Lecture “Programming”
SS 2021

Take-home exam 2 – C++, OpenMP, MPI, Python

Simon Scheidegger

The solutions to this exam sheet are due **Friday, June 4th, 2021, 08.59h.**

The individual solutions should be submitted as follows:
1) Take a snapshot of your instance named "HT2_SurnameName" in Nuvolos as explained in the file "Introduction to Nuvolos.pdf" on Moodle. **The last version of your snapshot taken before the deadline will be corrected**.
2) As a backup solution to 1), please upload a folder named "HT2_SurnameName" that contains your solutions on the following Dropbox link in a single file (e.g., a zip file):
https://www.dropbox.com/sh/ldl8e13gezjgaow/AAB7j1pGHu5B5b4GoYXq_2ZMa?dl=0

No late hand-in is accepted. A late hand-in will result in a grade of 1 (zero points) for the take-home exam.

You have to solve this take-home exam alone and therefore you have to submit your own solutions. **No work in teams is permitted. Plagiarism will result in a grade of 1** (zero points) for this take-home exam. Note that we may run a plagiarism checker to find similarities across your solutions.

You have to submit your work as individual source code files that can be run (e.g., mysolution1.cpp, mysolution2.cpp, mysolution1.py, etc.) in respective folders that are provided to you.
→ All exam questions need to be solved by using the C++ or the Python programming language.
→ JUPYTER NOTEBOOKS are NOT allowed as way to submit your solutions. Work submitted in a Jupyter Notebook will not be graded, i.e., will get 0 points for those answers.
→ All submitted codes need to be able to compile and run on the Nuvolos compute cluster directly. Any submitted work that does not compile and run correctly will immediately get a discount of 50% of the total points that could be obtained in that particular exam question.

Don’t forget to include comments into your source code. This will help us understand your thoughts when we correct it.

Good luck! Go for it!

Hint:
- You should use the app **“Programming 2021”** by default.
- Please also note that you need to **connect to the HPC** cluster to run efficiently in parallel.
To do so, please type on the terminal the commands below, as learned during the lecture
connect_cluster
then type
source /opt/miniconda3/bin/activate
- Files that should be synchronized between the Nuvolos app and the cluster should be put into the folder
files_for_cluster.
- More hints (for pybind11) are at the end of this exam (on page 12).

Note: there are a maximum of 75 Points that can be achieved in this exam (plus 5 Bonus points)

Question 1 (10 Points)

Templates in C++, makefile, slurm

!!! NOTICE THE SKELETON CODE AT THE END OF THIS QUESTION !!!

a) (3 Points) Implement a template function ellipsoid_volume in C++ that returns

$$\mathrm{V}_{ellipsoid} = \frac{4}{3}\pi * abc$$

the function of an ellipsoid of half-axes *a*, *b*, and *c*.
The function should compile with all types (int, float, double) used in the skeleton code and give the following output similar to this one:

```
4/3 * pi * 3.f * 2.f * 1.f = 25.1327
```

Store above's function in a file called ellipsoid_volume.cpp.
Make sure that the function is callable from a function “int main()” within the same file (see the code skeleton at the end of this question).
Evaluate this function by calling it in the main() function, and set a = 3, b =2, and c = 1 (notice that it should be possible that the types can be modified from double to float and int, and the function should be evaluated for all those cases separately).

b) (3 Points) Implement a makefile that compiles above’s code. Once written, the following sequence of commands executed on the terminal should permit the user to compile and run the software.

> make
> ./your_executable_name.exec (run your “executable”)

c) (2 Points) Write a short README.md that documents how to compile and run the entire code.

d) (2 Points) Fix floats a = 5.0, b = 3.0, c = 23.05 inside the main function in your ellipsoid_volume.cpp. Then, implement a script to run your executable (“your_executable_name.exec”) on a cluster. To do so, use “slurm”. The slurm.sh file that you are supposed to implement should request

- 1 cpu,
- 2 minute of runtime
- have a job name “ellipse”
- write the error into a file ellipse_error.err
- write the output into a file ellipse_output.out

Submit a job via this “slurm.sh” script on the nuvolos cluster. Collect the ellipse_error.err and ellipse_output.out files, and submit those files to the rest of the exam files. To run your scripts, you need – as show in the lecture – connect to the cluster.

********** CODE LISTING **********

```cpp
#include <iostream>
#include <cmath>

const double pi = M_PI;

// Implement the function(s) ellipsoid_volume for question 9 here

/// DO NOT CHANGE any of the following code.
int main() {
    float float_v = ellipsoid_volume(3.f, 2.f, 1.f);
    std::cout << "4/3 * pi * 3.f * 2.f * 1.f = " << float_v << std::
        endl;

    double double_v = ellipsoid_volume(3., 2., 1.);
    std::cout << "4/3 * pi * 3.  * 2.  * 1.  = " << double_v << std::
        endl;

    double_v = ellipsoid_volume(3, 2, 1);
    std::cout << "4/3 * pi * 3   * 2   * 1   = " << double_v << std::
        endl;
}
```

Listing 1: Code skeleton for Question 1.

********** END CODE LISTING **********

Question 2 (10 Points)

Implementing a class in C++, makefile, slurm

The following **main** below function reads a sequence of integers from standard input until the sum of their absolute values exceeds a threshold (here 10) and in the end prints the maximum of their absolute values. It uses a **MaxAfterThreshold** class, which is initialized with the threshold.

********** CODE LISTING **********

```cpp
#include <cmath>
#include <iostream>
#include <algorithm>

// Define and implement the class MaxAfterThreshold here



/// DO NOT CHANGE any of the following code.
int main()
{
  MaxAfterThreshold max_thresh(10);

  std::cout << "Enter numbers:" << std::endl;
  do{
    std::cin >> max_thresh;
  } while(!max_thresh.limit_exceeded());

  std::cout << "The maximum of the absolute values is " << max_thresh()
            << "." << std::endl;

  return 0;
}
```

Listing 2: Code skeleton for Question 2.

********** END CODE LISTING **********

a) (6 Points) Define and implement this class in C++ such that the program compiles and runs correctly. Pay attention to data encapsulation and declare members as constant where possible.

b) (2 Points) Implement a makefile that compiles the program and generates an executable called “max.exec”.

c) (2 Points) Implement a slurm.sh script that can run the executable generated in b), and ensure that it runs correctly on Nuvolos with 1 CPU, and a walltime of 1 minute.

Question 3 (10 Points)

Creating a library in C++, Cmake

The following code below defines a **scale** function scaling a vector by a factor, and a **rotate** function rotating a vector around the origin, along with a main function which tests these functions.

a) (4 Points) Refactor this code by creating a library for the **scale** and **rotate** functions and change main.cpp to use this library.

b) (2 Points) Write a Makefile which

- builds a (static or dynamic) operations library containing the **scale** and **rotate** functions and
- creates a main program by compiling **main.cpp** and linking to the library.

c) (4 Points) Write a CMakeLists.txt file which

- builds a (static or dynamic) operations library containing the scale and rotate functions,
- builds a test_operations program by compiling main.cpp and linking to the library,
- installs header, library and test program into the include, lib and bin subdirectories of the location specified by the variable CMAKE_INSTALL_PREFIX.

********** CODE LISTING **********

```cpp
#include <vector>
#include <cmath>
#include <iostream>

void scale(std::vector<double>& x, const double scale_factor){
        x[0] *= scale_factor;
        x[1] *= scale_factor;
}

void rotate(std::vector<double>& x, const double degrees){
        double radians = degrees*M_PI/180.;
        double temp = x[0];
        x[0] = std::cos(radians)*x[0] - std::sin(radians)*x[1];
        x[1] = std::sin(radians)*temp + std::cos(radians)*x[1];
}

int main(){
        const double sf = 1./std::sqrt(2), angle1 = 45., angle2 = 135.;
        std::vector<double> x (2, 1.), y (2, 1.);

        y[1] = -1;

        scale(x,sf);
        scale(y,sf);

        std::cout << "x = [" << x[0] << ", " << x[1] << "]\n";
        std::cout << "y = [" << y[0] << ", " << y[1] << "]\n";

        rotate(x, angle1);
        rotate(y, angle2);

        std::cout << "x = [" << x[0] << ", " << x[1] << "]\n";
        std::cout << "y = [" << y[0] << ", " << y[1] << "]\n";

        return 0;
}
```

Listing 3: Code skeleton for Question 3.

********** END CODE LISTING **********

Question 4 (10 Points – FYI: this is a harder question)

Inheritance in C++

In the displayed main function a std::vector is allocated with pointers to fruit-type objects. Both fruit-type objects are passed to the same print function, which prints the sugar content of the fruit.

Using inheritance, modify the code below without changing the print or main function bodies so that it compiles and prints the following output when running the executable named **inheritance.exec:**

```
3 5 7 6 11 16
```

De-duplicate/simplify as much code as possible and make sure that the memory cleanup is done properly!

********** CODE LISTING **********

```
#include <iostream>
#include <vector>

class Apple {
  public:
  Apple(int ripeness) : ripeness_(ripeness) {}
  int sugar_content() const { return 2*ripeness_+3; }

  private:
  int ripeness_;
};

class Orange {
  public:
  Orange(int ripeness) : ripeness_(ripeness) {}
  int sugar_content() const { return 5*ripeness_+6; }

  private:
  int ripeness_;
};

void print(/* to be completed ... */) {
/// DO NOT CHANGE any code from this line on.
  std::cout << fruit.sugar_content()  << " ";
}

int main() {
  std::vector<Fruit*> fruits;
  for(int i = 0; i < 6; ++i)
    i < 3 ? fruits.push_back(new Apple(i)) :
        fruits.push_back(new Orange(i-3));
  for(int i = 0; i < 6; ++i )
    print(*fruits[i]);
  std::cout << std::endl;
  for(int i = 0; i < 6; ++i )
    delete fruits[i];
  return 0;
}
```

Listing 4: Code skeleton for Question 4.

********** END CODE LISTING **********

Question 5 (15 Points – FYI: this is a harder question)

Standard Library

The 10% trimmed (or truncated) mean of a set of numbers v = {v_0, v_1 , . . . , v_{N-1} } is the average after discarding the 10% of the lowest and highest values. In particular, you shall discard the M lowest and highest values, where M is the integer part of the product of N with the trimming fraction (10%). Aim for a solution with complexity O(N log N). Make sure that the function trimmed_mean cannot modify the input. Handle appropriately the case of an empty input set (N = 0) in trimmed_mean.

a) (5 Points) The following function computes the trimmed mean of elements in a **std::vector<double>**.
The second argument is the trimming fraction. Complete the function implementation without using any explicit loops, but use standard library algorithms instead.

********** CODE LISTING **********

```
#include <iostream>
#include <vector>

/// Preconditions: 0<=trimming_fraction<0.5
/// Postconditions: the input data will not be altered
double trimmed_mean(/*to be completed*/, double trimming_fraction)
{
  //to be completed ...
}

int main()
{
  std::vector<double> numbers;
  for(int i = 0; i < 10; ++i)
    numbers.push_back(i);

  std::cout << trimmed_mean(numbers,0.1) << std::endl;

  return 0;
}
```

Listing 5: Code skeleton for Question 5.

********** END CODE LISTING **********

The output of the code should be as follows:

```
4.5
```

b) (5 Points) The following function computes the trimmed mean of the elements of a given type value_type between two forward iterators. The last argument is the trimming fraction. Complete the function implementation using iterator traits and without using any explicit loops.

********** CODE LISTING **********

```
#include <iostream>
#include <iterator>
#include <vector>

/// Preconditions: 0<=trimming_fraction<0.5
/// Requirements: ForwIt is a forward iterator
/// Postconditions: the elements in the range between the iterators
     will not be altered
template<typename ForwIt>
double trimmed_mean(/*to be completed*/, double trimming_fraction)
{
  using value_type = typename std::iterator_traits<ForwIt>::
      value_type;
  //value_type is the type of values the iterator ForwIt points to

  //to be completed ...
}

int main()
{
  std::vector<int> numbers;
  for(int i = 0; i < 10; ++i)
    numbers.push_back(i);

  const std::vector<int> v0(numbers.begin(),numbers.end());
  const std::vector<float> v1(numbers.begin(),numbers.end());

  std::cout << trimmed_mean(v0.begin(),v0.end(),0.1) << ' ' <<
      trimmed_mean(v1.begin(),v1.end(),0.1) << std::endl;

  return 0;
}
```

Listing 6: Code skeleton for Question 5.

********** END CODE LISTING **********

The output of the code should be as follows:

```
4.5 4.5
```

c) (5 Points) Implement deduction of the return type for the trimmed_mean function according to the **value_type**: the return type shall be the **value_type** with the exception of integers, in which case it shall be **double**.

Question 6 (10 Points)

Pybind11

a) (2 Points) Implement a function in C++ that computes the solutions to the quadratic equation

$$ax^2 + bx + c = 0$$

Take care that your function can deal both with real as well as with imaginary solutions.

b) (2 Points) Use Pybind11 to create Python bindings for this function. The module name you can use in python should be “quad_fun”.

c) (2 Points) Compile the implemented code to generate a binary module that can be imported in Python.

d) (2 Points) Implement a docstring in the C++ code so you can use the help function in Python to see how this module can be used.

e) (2 Points) Import the generated library in Python (via “import quad_fun) in a file called eval_quad_fun.py, and evaluate the function quad_fun for two settings:

a =2, b=3, c = 4
a = -1, b = -2, c = -5

Question 8 (10 Points)

Scalability

You have a program that you would like to parallelize. You know that 0.015 of the total runtime is used to start up the program and parse some input data—that is, it cannot be parallelized. The rest of the program is embarrassingly parallel.

a) (3 Points) What is the maximum speedup you could expect if you parallelize such a code? What is the maximum number of CPUs you could use if you want to keep the parallel efficiency above 60%, i.e., the minimum level of efficiency you would need to run on CSCS?

b) (3 Points) In the Lecture 10 of this class, we looked at an implementation of an economic model that is solved with value function iteration. The presented implementation was parallelized with OpenMP. The location of the code on Nuvolos relative to your lecture materials was here:

Lecture10/Projects/DynamicProgramming/openmp_DP

→ In **params.cpp**, set nk = 7000; and Numits = 3.

- Connect to the Nuvolos Cluster as shown in the class by typing connect_cluster.
- Compile the code, and run the executable with 1, 2, 4, and 8 threads (by using slurm – set a maximum wall-time of 4 minutes). Note: the run-times will be reported inside the output file that is generated for the different runs.
- Generate – as shown in the lecture – a figure that shows the parallel efficiency as a function of the number of threads.
- Generate – as shown in the lecture – a figure that shows the parallel speedup as a function of the number of threads.
- Attach both this figure as well as the generated slurm output files to the materials you submit as solution to this exam.

c) (3 Points) In the Lecture 11 of this class, we looked at an implementation of an economic model that is solved with value function iteration. The presented implementation was parallelized with MPI. The location of the code on Nuvolos relative to your lecture materials was here:

Lecture11/demo/DynamicProgramming/MPI_DP

→ In **params.cpp**, set nk = 8000; and Numits = 3.

- Connect to the Nuvolos Cluster as shown in the class by typing $ connect_cluster.
- Compile the code, and run the executable with 1, 2, 4, 8, and 16 MPI processes.
  To do so, implement a slurm script (set a maximum wall-time of 5 minutes). Note: the run-times will be reported inside the output file that is generated for the different runs.
- Generate – as shown in the lecture – a figure that shows the parallel efficiency as a function of the number of threads.
- Generate – as shown in the lecture – a figure that shows the parallel speedup as a function of the number of threads.
- Attach both this figure as well as the generated slurm output files to the materials you submit as solution to this exam.

d) (1 Points) Question: Do the value function iteration codes run in b) and c) seem to converge? Tell verbally, where the convergence is computed inside the codes discussed in b) and c), and generate graphs that illustrate how the value function iteration converges with iterations.

Bonus Question (5 Points)

MPI4PY

Below is a code listing that computes “pi” numerically in python with the Monte Carlo Method.

********** CODE LISTING **********

```
import math

def compute_pi(n):
    h = 1.0 / n
    s = 0.0
    for i in range(n):
        x = h * (i + 0.5)
        s += 4.0 / (1.0 + x**2)
    return s*h

n = 1000000
pi = compute_pi(n)
error = abs(pi - math.pi)
print ("pi is approximately %.16f, " "error is %.16f" % (pi, error))
```

********** END CODE LISTING **********

a) Use MPI4PY to parallelize this code with MPI.

b) Implement a slurm.sh file that can run this code with MPI. The output should be directed in a file entitled s**caling_mpi4py_no_procs.out** for every run, where **no_procs** should be replaced with the actual number of MPI processes with which the code is run.

c) Run the code with n = 10 billion points, and 1, 2, 4, and 8 MPI processes, and see whether you obtain any speedup over the 1 CPU baseline run. Is there any strong scaling? To assess the runtime, use the timer from MPI, i.e., MPI.Wtime().

========== END OF THE EXAM AND THE SEMESTER ==========

ADDITIONAL INFORMATION WHEN WORKING ON NUVOLOS

1. Working with pybind on Nuvolos

There are 2 steps of using pybind: the first step is the compilation of the C++ source, then the running of the compiled code. It is installed, so it should work per default. However, if not, you have two options.

1. Compiling the C++ source with pybind

You have 2 options to do this: you can either compile on Nuvolos or on the login node of the HPC cluster

A) Compile on Nuvolos: in a terminal of any of the JupyterLab applications, install pybind with

```
pip install pybind11
```

After that, the required header files will be available at

```
python3 -m pybind11 --includes
```

B) Compile on the login node: first do a connect_cluster from a Nuvolos terminal in any of the JupyterLab applications, then use

```
source /opt/miniconda3/bin/activate
```

to activate the base environment where pybind is already installed. After this, the required header files will be available at

```
python3 -m pybind11 --includes
```

2. Running the compiled code

A) Run on Nuvolos directly: once the code is compiled, you can run it in any of the JupyterLab applications. Note that student applications are constrained to 1 vCPU.

B) Run on the Slurm cluster: if you want to run the code on the cluster, it's recommended to compile the code on the cluster (option 1/B). To run python code on the cluster, you should import the

python-3.6.5-intel-17.0.6- module in your sbatch file, like this:

```
#!/bin/bash

#SBATCH ...


module load python-3.6.5-intel-17.0.6-adb26hh

python ...
```