Assignment 3 Instructions

Paolo Joseph Baioni

December 27, 2022

For assignment 3 you have to implement a parallel program inspired to max pooling method for square matrices; you can rely on a provided Matrix class. Your implementation has to take into account whether given matrices are row-major or column-major.

Remark Row-major order and column-major order are methods for storing multidimensional arrays in linear storage. The difference between the orders lies in which elements of an array are accessed one after another. In row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in column-major order. For example, the provided `ordering-example.cpp` source file:

```cpp
#include "Matrix.h"
#include <iostream>

int main(){
  std::cout<<"M:\n";
  Matrix m("row-major",2,2,{1,2,3,4});
  m.print()<<std::endl;
  std::cout<<"N:\n";
  Matrix n("col-major",2,2,{1,2,3,4});
  n.print()<<std::endl;
  std::cout<<"M after order switch:\n";
  m.switch_ordering();
  m.print();
return 0;
}
```

which can be compiled with:

```
g++ -std=c++11 -Wall Matrix.cpp ordering-example.cpp -o ordering-example
```

and run with:

```
./ordering-example
```

will produce the following output:

```
M:
2 x 2 row-major
1 2
3 4

N:
2 x 2 col-major
1 3
2 4

M after order switch:
2 x 2 col-major
1 3
2 4
```

□

In particular, your program has to:

- divide input matrix across the `size` processors in rows if row-major, in columns if column-major
- output a `size` long vector with the local maxima, column vector if matrix is row-major, row vector if the matrix is column-major (for vectors, row or column major ordering doesn't matter):

As examples, you can refer to the expected result datafiles, which follow:

- Mr-expected-result-size4.dat

```
Input matrix is:
4 x 4 row-major
10 11 12 13
14 15 16 17
18 19 20 21
22 23 24 25

Max pooling gives:
4 x 1
13
17
21
25
```

- Mr-expected-result-size2.dat

```
Input matrix is:
4 x 4 row-major
10 11 12 13
14 15 16 17
18 19 20 21
22 23 24 25

Max pooling gives:
2 x 1
17
25
```

- Mc-expected-result-size5.dat

```
Input matrix is:
5 x 5 col-major
22 27 32 37 42
23 28 33 38 43
24 29 34 39 44
25 30 35 40 45
26 31 36 41 46

Max pooling gives:
1 x 5
26 31 36 41 46
```

- Mc-expected-result-size1.dat

```
Input matrix is:
5 x 5 col-major
22 27 32 37 42
23 28 33 38 43
24 29 34 39 44
25 30 35 40 45
26 31 36 41 46

Max pooling gives:
1 x 1
46
```

It's worth noting that the same matrix stored in a different major will produce a different output, even with the same number of parallel processes:

Mr-expected-result-size4.dat

```
Input matrix is:
4 x 4 row-major
10 11 12 13
14 15 16 17
18 19 20 21
22 23 24 25

Max pooling gives:
4 x 1
13
17
21
25
```

Mrc-expected-result-size4.dat

```
Input matrix is:
4 x 4 col-major
10 11 12 13
14 15 16 17
18 19 20 21
22 23 24 25

Max pooling gives:
1 x 4
22 23 24 25
```

You can assume that each matrix dimension divides the number of parallel processes, so for any matrix the code should work with `-np 1`, for 4x4 matrices with also `-np 2` and `-np 4`, for 5x5 matrices with `-np 1` and `-np 5` and so on; you can neglect pathological cases (such as empty matrices, non-square matrices, matrices with more elements in a row or column than in the other ones...), only real square matrices will be used for testing.

The provide code zip archive `Assignment3-initial.zip` contains:

- some examples, among which the ones used above and the input matrices to produce them
- the Matrix class implementation
- a `main-initial.cpp` file to be completed
- this documentation in pdf format

The `main-initial.cpp` file is the only one that has to be modified, as written in the comments thereby. The function `double matrix_max(const Matrix & M)` is not implemented, its implementation can be useful but it is optional. Mind that, as can be seen from the source file, rank 0 is the one that manages I/O, so is the only one knowing the full input matrix; data should be then divided properly to the other ranks, according also to the different input matrix major.
The code has to be compiled with (for example, with 4 processors using the provided `Mr.dat`):

```
mpicxx -std=c++11 -Wall Matrix.cpp main-initial.cpp -o program
```

And run with:

```
mpiexec -np 4 ./program Mr.dat
```

or with:

```
mpiexec -np 4 --oversubscribe ./program Mr.dat
```

if openmpi requires it on your configuration; in case you don't know, you can start with the first command, and if openmpi will complain about the fact that *There are not enough slots available in the system to satisfy the 4 slots that were requested by the application:...* you can retry with the `--oversubscribe` option.

Remark Nothing should be printed to the standard output, in particular no `std::cout` are allowed, with the only exception of the one already present in the initial code. For debugging purposes, you are strongly advised to write to the standard error, eg to use `std::cerr` instead, so to avoid the risk of forgetting to delete any additional `std::cout`s from the code you will upload. Codes producing an output different from the expected one will be considered wrong, even if the max-pooling is implemented correctly: a correct output formatting is part of the assignment too. To check if your output is correct you can redirect the standard output of your program to file, and then compare it with the expected one relying on the shell utility `diff`;

`mpiexec -np 2 ./program Mr.dat > program-output-file` (or with `--oversubscribe` if needed)
`diff program-output-file Mr-expected-result-size2.dat`

and the same for other provided examples. If nothing is printed it means there are no differences, so your output is correct; see `man diff` for more details. □

Your program should be portable, but consider that for grading it will be tested on both the docker container and the virtual machine, and it will be enough to get the correct result on one of them to get the point, so it is advisable to test it in one of those environments before your final upload. In case you can't, it's advisable to use a reasonably recent version of OpenMPI (the container runs Open MPI 4.1.4 and the VM Open MPI 3.4.2), besides a proper C++ compiler such as g++ or clang++.

Remark Assignment is not mandatory, if developed correctly can lead to a +1 point in the final grade. To get your program positively graded:

1. Only one file has to be uploaded, namely the main-initial.cpp file, **renamed** as PERSONCODE.cpp. For example, if your person code is 12345678, you have to upload one and only one file, named 12345678.cpp. Mind that person code is a eight digits number.
2. Such file has to be uploaded on WeBeep→Assignments(it: Consegne)→Assignment3; also file uploaded as drafts will be corrected.
3. Assignment is personal, no group or team work is allowed. For plagiarism, -1 penalty is foreseen; more details in the first day lecture slide.

In case of need, questions have be posted on the WeBeep Forum (Notice Board → Assignments Forum for prof. Ardagna students, Notice Board → Forum for prof. Tamburri students).
For WeBeep Assignments consult (once logged in):

https://webeep.polimi.it/mod/glossary/view.php?id=580&mode=cat&hook=32

Scheduled deadlines follow:

1. Code submission opens now, 27/12/2022 at 20:30, and closes on 04/01/2022 at 08:30 AM (Rome Time), so you have 7.5 days to submit your solution
2. Grades draft will be published on 04/01/2022 evening, you will have up to 06/01/2022 at 08:30 AM (Rome Time) to write in the code any question regarding your grade
3. Grades will be updated or confirmed on 06/01/2022

In case something would change you will be notified by an announcement (you will not have less time than written above.)