High Performance Computer Architectures Practical Course - Exercise 4 -

Tutorium 1

David Jordan (6260776)
Florian Rüffer (7454628)
Michael Samjatin (7485765)

May 16, 2023

1 Matrix

2 Quadratic Equation

3 Newton

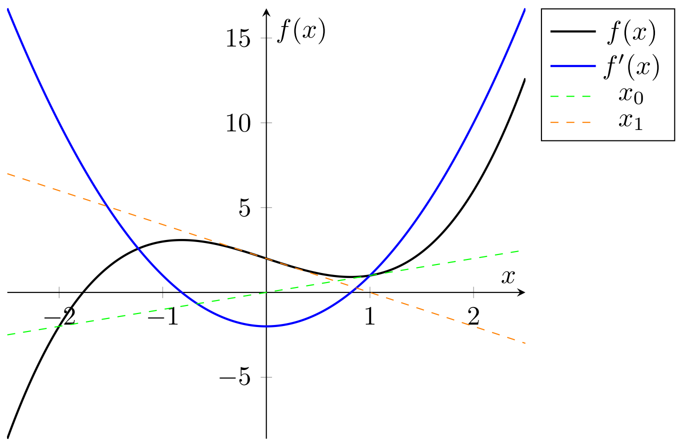
In this task we have to vectorize the Newton method. As mentioned in the task description the Newton Method is an interative process to approximate the x-axis intersection.

```cpp
float_v FindRootVector(const float_v& p1, const
   float_v& p2)
{
  float_v x = 1.f, x_new = 0.f;
  float_m mask(true);
  for( ; !mask.isEmpty(); ) {
      // for(int i = 0; i < 1000; ++i){
    x = x_new;
    x_new(mask) = x - F(x,p1,p2) / Fd(x,p1,p2);
    mask = abs((x_new - x)/x_new) > P;
  }
  return x_new;
}
```

File 1: Newton.cpp

The main idea of vectorizing the Newton method, is the calculation of multiple roots at once. This comes along with an issue, because if multiple roots are calculate, then also some of those will reach the desired precision earlier than others. To solve this problem we will use masks. The underlying principle of masks is a vector with each value mapped to a boolean. Each boolean value corresponds to the value of another vector. Using this methodology we can assign 'false' to a corresponding number, which already has our desired precision and therefore exclude it from further unneccessary calculations.

The first iteration will start with a mask of only 'true' values, as no value has yet reached the desired precision. This means every value will be updated accordingly with $x - F(x, p1, p2)/Fd(x, p1, p2)$. The next iteration will only update those values, which need further refinement ($\rightarrow$ mask value equals 'true') with $mask = abs((x_new - x)/x_new) > P$.


$f(x)$
x
15
10
5
-5
-2
-1
1
2
$f(x)$
$f'(x)$
x_0
x_1

4 Random Access

The first part of this exercise is done by gathering the data from the input arry to the tmp float_v variable according to the index array. This is done by initializing a uint_v vector and it's corresponding indexes. If this is done, the goal of the subtask can be achieved by running the gather function:

```
    tmp.gather(input, ind);
```

File 2: gathering data

Now we want to filter the data - from the tmp variable every value larger than 0.5. Therefore we create a specific mask with the following code: float_m mask = tmp > 0.5f;
With this mask, and a new initialized variable, where we want to safe the data, we can run the "gather" function again, with adapted parameters:

```
    tmp2.gather(input, ind, mask);
```

File 3: gathering filtered data

The last part of this exercise is to put the data from the tmp float_v variable to the ouput array, when it satisfies a given condition. The condition is, that all values from tmp, that are smaller than 0.5, are put to the array "ouput" at the places given by indices.
We could create a new mask, but because this mask's type is identical to the mask above (and because the solutions does it like this), we reuse the mask (of course with adjusted filter < 0.5).
Lastly one can just use the scatter function with the added parameters to get the wished results to the tmp variable.

```
    tmp.scatter(input, ind, mask);
```

File 4: scattering

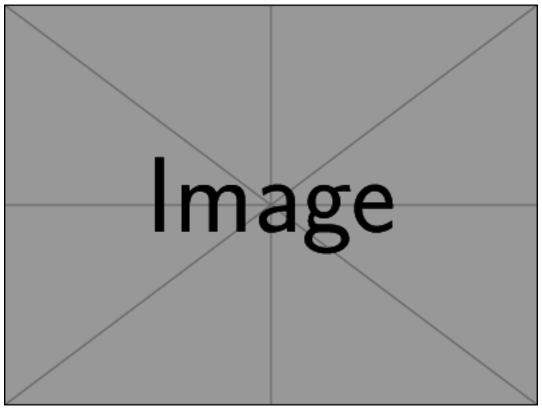


Figure 1: Output