High Performance Computer Architectures Practical Course - Exercise 5 -

Tutorium 1

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

May 23, 2023

1 First Third

First and foremost, we must decide which data should be grouped and how it should be grouped in order to vectorize the track fitting procedure. To achieve maximum independence, M tracks can be handled simultaneously. The procedure involves:

1. Initializing M tracks.
2. Extrapolating M tracks to the first station.
3. Estimating track parameters using M hits (one hit per track).
4. Extrapolating M tracks to the second station.
5. Continuing extrapolation to the z-coordinate of the last mc point, shared by all tracks.

To execute these steps, hits from different tracks should be grouped into a single vector, where each group corresponds to a specific station. The class representing the group should include a vector of M x-coordinates for the hits, along with a scalar representing their z-coordinate. The general type for grouping M floats together is denoted as T, allowing for the substitution of fvec and float types using templates (see ”struct LFPoint” in code).
-> M track parameters in one vectorized parameters class

The x parameters and covariance elements are grouped together, while the z-coordinate remains scalar. The grouping allows for efficient processing of M floats, denoted as T, using templates.

Because of the different chi-squared deviation and the NDF for the different tracks, we need a vector type. You can see this done here:

```
template< typename T, typename I >
struct LFTrack {
  vector< LFPoint<T> > hits;

  LFTrackParam<T> rParam;
  LFTrackCovMatrix<T> rCovMatrix;
  T chi2;
  I ndf;

  vector< LFTrackParam<T> > mcPoints;
};
```

File 1: KFLineFitter.cpp

Second Third

The Code in this excerpt of the program simulates and fits tracks in the program's tracking system. The LFGeometry struct stores the geometry of the tracking system, that is the z-value of each of the stations. A staion here refers to the individual measurement points in the tracking system where hits can be made. The LFSimulator class simulates the tracks using the given set of parameters. its simulation functions generate the simulated tracks, by creating MOnte Carlo points for each station and distributing the potential hits based on the sigma value. The thereby simulated tracks are stored in the LFTrack objects. The LFFitter class fits the simulated tracks. Fitting here describes the process of finding the best-fit parameters of the underlying model that can describe the trajectory of the particle being tracked by the system. The Initialise function initialises the track parameters and the covariance matrix of the track object. It does this before the fitting process. The Fit function takes a track object as an argument, and utiliyes the simulated hits stored in the track object to refine the track parameters and thereby obtain a best fit. The Extrapolate function extrapolates the track parameters to a given z-value. The Filter function compares the simulated hit position with the predicted position from the track parameters, calculates the difference between the two, and updates the covariance matrix on the basis of this comparison. The LFFitter classapplies the fitting algorithm iteratively, with the goal of finding the best fit to the observed hits and thereby leading to a better description of the particle's trajectory. The whole excerpt defines the necessary data structures for simulating tracks and fitting the simulated tracks with a Fitter.

```
 class LFSimulator
{
public:
  // the constructor
  LFSimulator() : fGeometry(), fSigma(DSigma)
  {
    fGeometry.zs.resize(DNStations);
    // loop to calculate the z coordinate for each
       station,
    // giving the geometric information necessary for
    // further tracking
    for (int i = 0; i < DNStations; ++i)
      fGeometry.zs[i] = (DDZ * static_cast<float>(i));
  };
  // Destructor
  ~LFSimulator(){};
  ...
```

Third Third

First of all the geometry is set up, containing the number of stations / particles, the distance between those stations and the hit distribution. Those values will result in a linearly spaced distribution of stations along the z-axis. This simulates a set of tracks with random starting positions and slopes.

Next up, we need to get our tracks fitted, to determine the most probable trajectory of a particle. For this we need to define the SIMD-ized version of the fitting process or it will be computed in a scalar way.

```
 1    const int NVTracks = NTracks/fvecLen;
 2    LFTrack<fvec,fvec> vTracks[NVTracks];
 3
 4    CopyTrackHits( tracks, vTracks, NVTracks );
 5
 6
 7    LFFitter<fvec,fvec> fit;
 8
 9    fit.SetSigma( Sigma );
10
11  #ifdef TIME
12    timer.Start(1);
13  #endif
14    for ( int i = 0; i < NVTracks; ++i ) {
15      LFTrack<fvec,fvec> &track = vTracks[i];
16      fit.Fit( track );
17    }
18  #ifdef TIME
19    timer.Stop();
20  #endif
21
22
23    CopyTrackParams( vTracks, tracks, NVTracks );
24
25  #endif
```

File 2: KFLineFitter.cpp

In the first line we calculate the number of vectorized tracks, to assign it to our list of vectorized tracks in the next line (as the length). In the following steps and lines of code, we transform our scalar track data into a SIMD vector, fit the tracks parameter to the measure data points and add Sigma (uncertainty) to our simulation. The for-loop then iterates over every vectorized track to fit multiple tracks simultaneously. Last but not least we copy our tracks back to their original format (line 23). Especially the functions CopyTrackHits and CopyTrackParams are of interest. This function iterates over the grouped tracks and for those over each hit for every track. All tracks have the same number of hits, those correspond to the number of stations, therefore we can treat this parameters as a constant. For CopyTrackParams we need to iterate over the vectorized tracks and their entries, as well as the elements of the covariance matrix.

The last lines of code save our results. If an output file is defined, then the program will output the parameters and covariance matrices for every track. The output will be stored in the defined file and will look like this:

```
1  0
2  90 -9.13228 -0.0736924
3  90 -9.03232 -0.0722956
4  0.0862335 0.00135828 3.00843e-05
```

File 3: Output

Here we will list the line and its corresponding explanation (index number corresponds to line in the output):

1. Track index, which increments by one for every track
2. True values of the parameters (Monte Carlo truth)
3. Reconstructed / fitted values for the same parameters
4. Covariance matrix for the fitted parameters

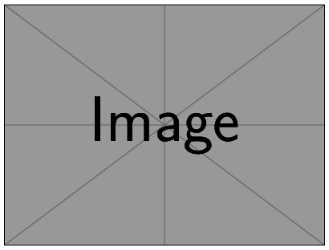

Figure 1: Output