STAR My report

V. Perevoztchikov
Brookhaven National Laboratory,USA

STAF

At the beginning of PHENIX and STAR as a frame work was selected STAF.

I was responsible guy to support it in both experiments. STAF itself was an eclectic mixture of:

- Fortran with structures support;
- Standard C;
- Kuip, which was used as a glue.
- Additional modules, providing Fortran & C calls.;
- The input/output data arrays of structures, accessible to Fortran and C as well .

The main idea was to provide the way to work with old Fortran and new C and C++ software.

It was huge number of bugs. Steering KUIP code becomes very big (tens thousand of lines) and unsupportable.

After bug fixes we run test jobs. No more 50 events was possible to proceed, because of memory leeks.

It became clear for both experiments that no sense to continue.

StMaker framework

Star Maker framework was mainly developed by me. ROOT and TdataSet class(Valery Fine) was widely used.   It is based on C++ StMaker class This class represents a module. These modules (Makers) organized the hierarchy(TDataSet). Mostly it is linear chain, but not always. Real work is done by inheriting the user class by StMaker. There a set of methods which could be overloaded by user class. Like:

- Init(),InitRun(), Make(), Clear(),Finish();
- AddObject(...) - add data object for communications with other makers. It is deleted on each event
- AddConst(...) - add data object which is permanent for the job
- GetData(...) - get data object created in another maker. You can use information from it and/or modify it
- GetConst(...) get permanent data object;
- And lot more...

StMaker continue

All the makers organized a chain or hierarchy. When chain is started :

- All maker->Clear() are called, to delete all the created data objects;
- Then all make->Make() are called to create and fill data objects;
- Again from the beginning
- At the end, all maker->Finish() are called .

The data objects all inherited from TDataSet. Makers themselves also inherited from TDataSet. Actually all the StMaker framework is a hierarchy of TDataSet's.

Each Maker could have the attributes. It is the simple way to manage the different type of jobs. The value of attribute could be number, character string or CINT function. STAF modules are implemented, so the STAF codes could be reused.

Star I/O

I/O evidently is important part of any framework. Several makers are equipped as input. The main one is StIOMaker. It can read:

- FZ , Geant3 output data;
- Daq data;
- ROOT file
- And even TTree file, which is overkill.

The output data are ROOT and TTree files. Very soon we got the problem with Schema Evolution. We need modify the format of data, but in the same time we should read old files. It was no Schema Evolution in ROOT those days. So I modified ROOT and implemented it. It was reported on ROOT workshop and Rene also implemented it with Philippe Canal and partially with my code.

Star I/O continue

Them I received request to write tracking data and related hits in separate files. But relations between hits and tracks must be kept. So we can read only tracks or tracks and hits. It is implementing of permanent pointers. I did it, and soon Rene also implemented it. I think he did it better.

When TTree was implemented in ROOT, it becomes clear that for DST this format is much better than usual ROOT one. StDSTMaker was developped, but not by me.

I/O summary:

- Input: DAQ, FZ, ROOT, Ttree;
- Output: ROOT, Ttree;
- ROOT Schema evolution is used;
- Permanent pointers are implemented.

Star DataBase maker

At the beginning, real Data Base was not implemented yet. I implemented StDBMaker, where set of CINT and ROOT files in the set of sub-directories was used as a data. This approach allows to very easily modify data by simple editor. Especially it is convenient for developing, when the data is not yet defined well. First productions was made with this “DataBase”. When the real DB was implemented, this maker was modified accordingly. In result StDBMaker:

- Can work with ROOT and CINT files in the directories tree;
- Can work with real MySql data objects;
- If there is a clash which object to use, objects in file-store have priority. So the data is merged from two sources;
- DB maker loads the new data only if new event has wrong validity. Only the outdated data is reloaded.
- Data is selected by event time, run number and/or by flavors

Star Reconstruction

As a first STAR reconstruction was used TPT, heavily modified ALEPH FORTRAN code. It was working well, but was limited to only one detector.

Then in STAR, was developed Sti C++ code(C.Prunot at all), based on ALICE reconstruction software. When I became a responsible for this project, I found that there are a lot of bugs, some from ALICE but many are local ones. Performance was force than the previous one. For real production it was not ready. After big modification and bug fixes Sti became better performance the TPT and ready for production. The main features:

- Several but TPC-like detectors;
- Semi automatic conversion from TGeo to Sti-geometry(Y.Fisyak)
- Magnetic field non constant but smooth;
- Kalman refit or smoother was implemented for better precision;
- Multiple seed finders added, including CA(Germany) one;

StarSim framework

StarSim(AtlSim) framework was developed by P.NevsiT. It is very powerful interface for Geant3. It allows:

- Very easy to create Geant3 geometry;
- Very reach geometry versioning;
- And a lot more.

When I became responsible for STAR simulation, it was working well in ATLAS then in STAR. I found and fix some not too important bugs and did some improvements. The runs many simulation jobs by requests of STAR physicists.

StMCFilterMaker

This maker allows to filtered out, reject simulation on different stages.

It could increase simulation speed in thousand time. Rejecting on

- Rejection inside of EventGenerator (Pythia);
- Rejection of GEANT Tracking;
- Rejection at GEANT End, No GEANT output

:

New STAR tracker(Stv)

Sti tracker is good enough, but there are unpleasant limitations.

- Sti geometry description is too simple;
- Sensitive volumes MUST be aligned along Z;
- Magnetic field also along Z;
- And many other limitations.

In Stv:

:

- For geometry description the standard TGeo is used.
- Arbitrary sensitive volume orientation;
- GEANT is used to propagate tracks through material;
- Fitting plane is always normal to track (accuracy);
- Local coordinate system is oriented along the track;
- Arbitrary magnetic field;
- Automatically assignment hits to sensitive volumes.
- Etc...

Runge-Kutta

Non constant magnetic field is not a field calculation. This is simple.

It is not a tracking, using Runge-Kutta method. This is more complicated but not too much. For the track fitting derivatives must be used. For the helix track, derivatives are not simple but analytically calculated. For Runge-Kutta it is not possible. It is possible to calculate derivatives numerically. But it is equal to calculate 5 more tracks, which is not became faster. I implemented approksimation, based on Helix derivatives. For the tests it works reasonably well. How it will work in real life we will see.

Track parametrization

Typical track parametrization is Lambda,Phi,Curvature. It is good for non forward tracks. For track close to beam, Phi became undefined. It leads to fit singularity. It is rather easy to use another parametrization, TanPhiX,TanPhiY, Curvature. Is is good for forward tracking but again singularity for track orthogonal beam.

So I chose rather complicated approach. For each step I used coordinate system where X is along the track. It is complicated bot for the fit is the best. Now I feel that there is more simple solution, but it is already implemented now,

New tracker continue

New tracker was implemented and tested. Quality was better than Sti but not too much. But speed was significantly slower. So it was decided do not speed it up, but just stop the development. For current set of detectors Sti is good enough.

Later, the idea of forward tracker arises, Sti here is not fitted at all.

So some work with Stv is going on again

:

Hit errors fitting

Hit errors parametrization formula for TPC is rather complicated.

To obtain it the fit is the only way. Direct use of track fitting is not possible. How it was solved?

- Select track whit high momentum, where multiple scattering is negligible;
- Helix is is replaced by parabolic line;
- For such simple track parametrization we can get formula for Chi2 and use it in the fit together with error parameters.
- Then start collective fit with a many tracks with the all track and errors parameters.
- Using obtained error parameters do normal track fitting.
- Repeating procedure up to error modifications will be small

Such fit is used for both, Sti and Stv

32 bits ==> 64 bits conversion

To convert code in C,C++ form 32 bits to 64 usually not to complicated. Nuch more complicated to do it with Fortran. Because there is no pointers in Fortran, good fortran programmers found the way how to use pointers without pointers. And their pointers are represented by integers . These are Zerbra, Geant, Comis, Starsim...

All these codes was heavily modified by me, especially Comis.

It was not an easy job but it is done.

: