Introduction to STAR software and makers

Leszek Kosarzewski

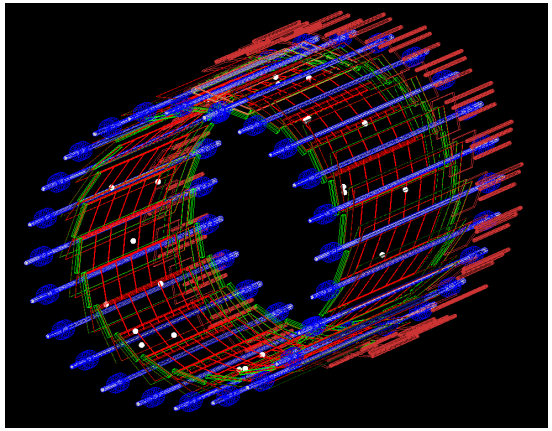
Warsaw University of Technology

STAR Collaboration Meeting 2015
Stony Brook University, 1-6.6.2015

Outline

Reconstruction in STAR - Example with IST

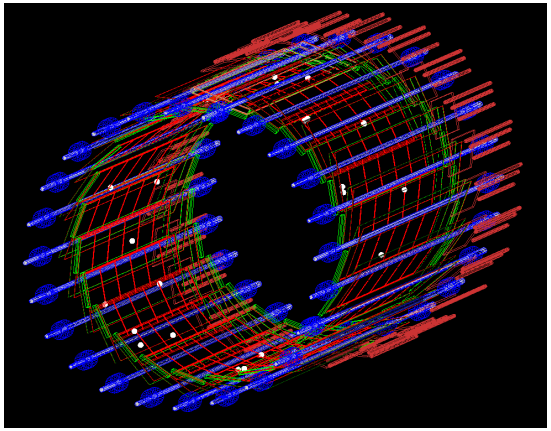
Raw ADC data



pedestal subtraction / zero supression

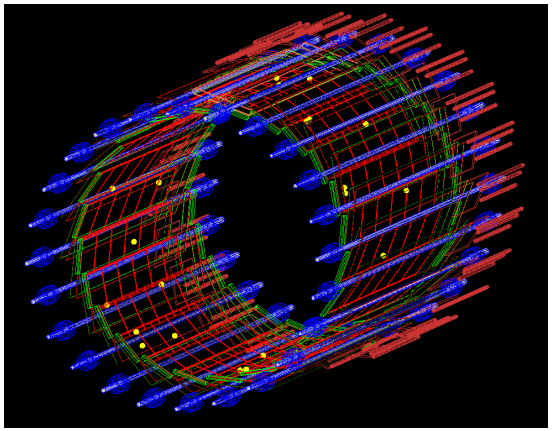
online/offline

Raw hits



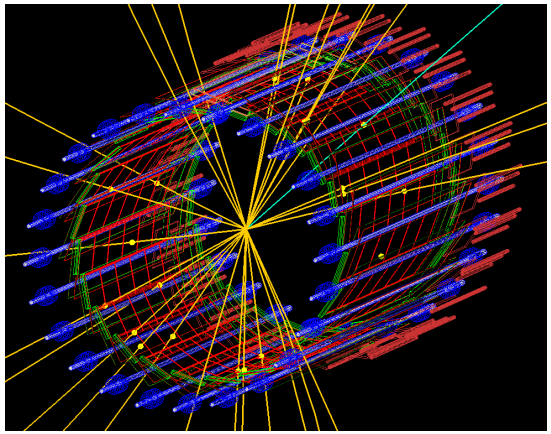
clustering

Hits



track reconstruction

Hits+tracks

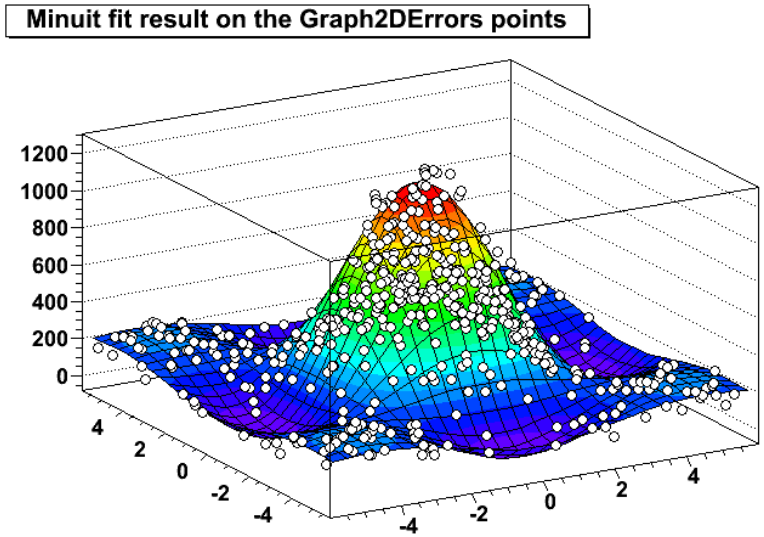


Each operation is done by its corresponding maker (pedestal subtraction, clustering...)

STAR software

Most of STAR software is based on libraries of ROOT data analysis framework.

- ROOT is widely used in Heavy Ion and High Energy Physics community
- STAR ROOT libraries contain classes specific to STAR:
  - Detector geometry classes
  - Event data
  - Specific makers
  - Database classes
- Look for information and class description on the ROOT webpage.



ROOT webpage

`https://root.cern.ch/drupal/`

Listing 1 : To run STAR ROOT

```
root4star
```

STAR Database

- Contains calibrations, gains, geometry transformations, etc.
- Access to Database information is done via StMaker::GetDataBase() method which calls St_db_Maker.
- St_db_Maker reads DB tables with a timestamp automatically set from the data or with SdtYYYYMMDD.HHMMSS command for simulation

DB tables are read in order:

1. ./StarDb - local directory
2. $STAR/StarDb - from current STAR Library version through AFS
3. MySQL database

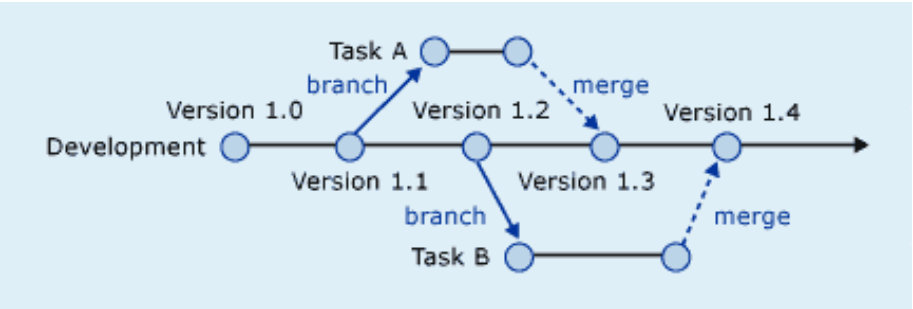
This allows to substitute official Db entries by a private version in order to check/modify calibration.

More information on STAR Database

https://drupal.star.bnl.gov/STAR/comp/db

STAR software - versions

- STAR software is constantly being updated as software developers make modifications or new detectors are added (HFT)
- Available remotely through an AFS cell
- The versions of STAR Libraries are labeled (listed only few):
  - SLYYx - where YY is the year, and x is a letter, corresponds to data production PYYix
  - DEV - latest development version, may be buggy or unstable
  - PRO - latest production version, default after login
  - NEW - a few weeks old version for full integration testing, more stable than DEV




```
starver SL15c // To select production P15ic:
stardev // To select DEV version
starpro // To select production version
starnew // to select NEW version
```

Listing 2 : Don't use above commands in login scripts!! Use command below instead:

```
if ($?STAR_LEVEL == 0) setenv STAR_LEVEL XXX
```

Library release structure and policy

https://drupal.star.bnl.gov/STAR/comp/sofi/soft-n-libs/release

Where to get it? CVS repository

CVS repository

- An archive of all codes and changes to the STAR software
- Source of codes for users

Listing 3 : To get a local copy of code from CVS repository:

```
cvs co <path_to_files>
```

Listing 4 : Compile source code:

```
cons
```

Quick guide to CVS in STAR

`https://drupal.star.bnl.gov/STAR/comp/sofi/tutorials/cvs`

CVS repository online viewer

`http://www.star.bnl.gov/cgi-bin/protected/cvsweb.cgi/`

Doxygen

- Automatically generates online code documentation
- Good for quick reference
- Contains links to the source code

Doxygen webpage

`http://www.star.bnl.gov/webdata/dox/html/index.html`

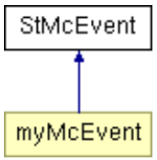
Main Page | Related Pages | Modules | Namespaces | Classes | Files | Examples | Search

Class List | Class Index | Class Hierarchy | Class Members

StMcEvent Class Reference

Event data structure to hold all information from a Monte Carlo simulation. This class is the interface to access all information from the GEANT simulations in STAR. From here one can get all vertices, tracks, hits in the event. More...

Inheritance diagram for StMcEvent:



List of all members.

Public Member Functions

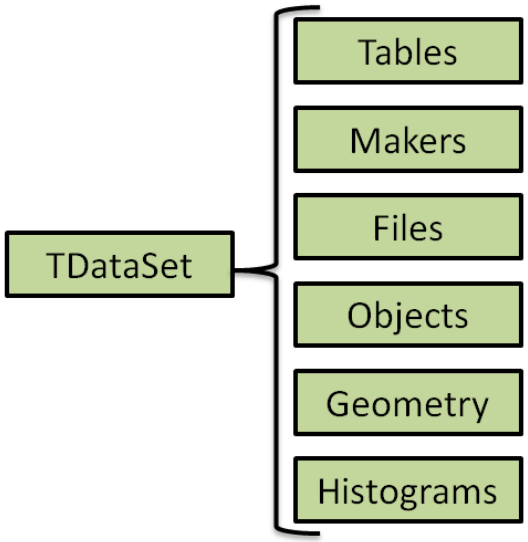
	StMcEvent (g2t_event_st *)
virtual void	**Browse** (TBrowser *b)
virtual bool	**IsFolder** () const
int	**operator==** (const **StMcEvent** &) const
int	**operator!=** (const **StMcEvent** &) const
unsigned long	**eventGeneratorEventLabel** () const

What is dataset?

Dataset

Dataset is a named list of dataset objects which allows to organize themselves in a tree. These objects can be almost any data!

Figure : Dataset can be a set of any of these:



TDataSet

- TDataSet (base class) is a part of basic ROOT
- Unique names for datasets are important, because they are accessed by their name.
- Datasets form a basis of STAR software for:
  - Data analysis
  - Simulation
  - Reconstruction

Listing 5 : Run an example chain and check it yourself!

```
root4star -l 'bfc.C(1,1,"gstar")'
```

What is dataset? See for yourself!

Figure : 1. Open a TBrowser and see a list of datasets

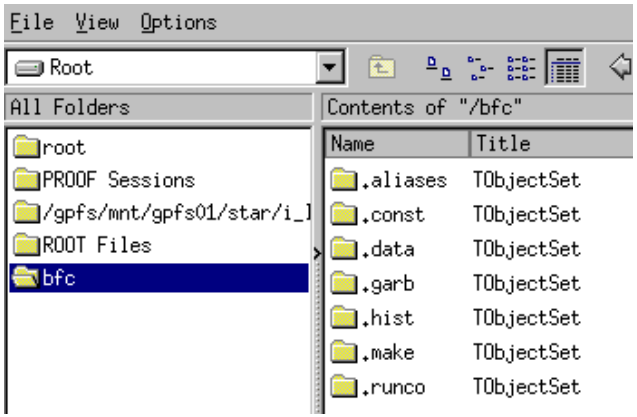


Figure : 2. Click on .make and check a list of makers

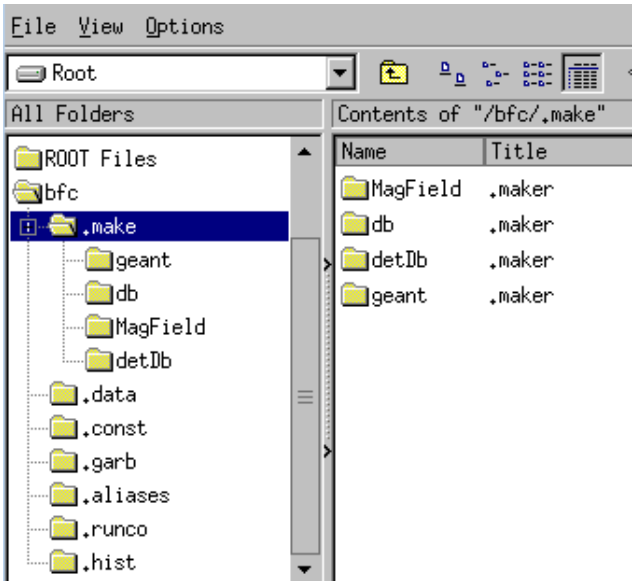


Figure : 3. Click on a maker for a list of it's datasets

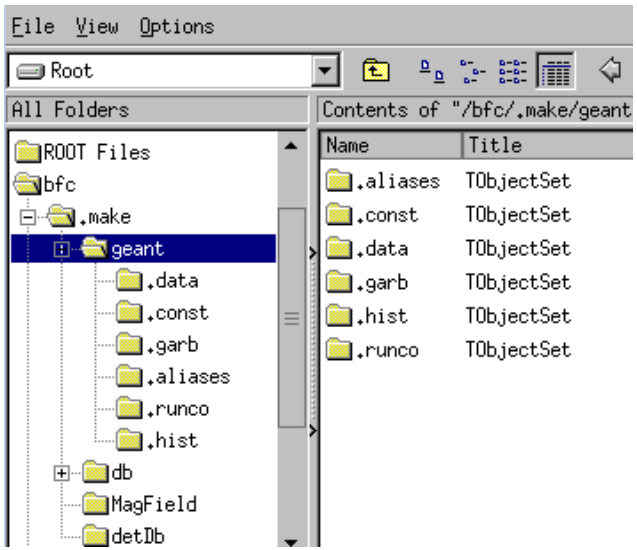
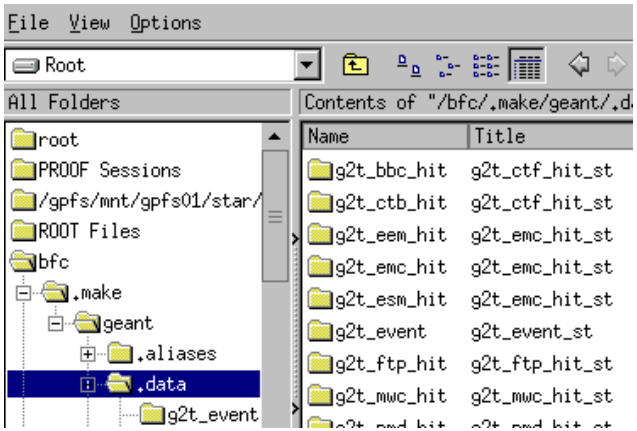


Figure : 4. Click on .data to see a list of data tables



Maker

- Maker is an owner of a dataset.
- StMaker (base class) is also a TDataSet itself.
- Makes operations on data:
  - Clustering
  - Applying corrections
  - Reconstruction
  - Data analysis
  - More...

Contents of a maker

- .make - list of child makers
- .data - list of owned datasets
- .hist - list of histograms created in StMaker::Init();
- .const - collection of constants used by the maker

Listing 6 : User defined functions - overwrite them with your own code

```
virtual Int_t Init(); // Called at the beginning, initialize histograms etc...
virtual Int_t InitRun(); // Called at the start of a new run
virtual Int_t Make(); // Called for every event, do your analysis here, fill histograms
virtual Int_t Clear(); // Used for clearing collections after each Make();
virtual void Finish(); // Finish
virtual Int_t FinishRun(Int_t oldrunumber); // Executed at the end of a run
```

Listing 7 : Get methods

```
virtual TDataSet *GetData(const char *name, const char *dir=".data") const;
virtual TDataSet *GetDataSet(const char *logInput) const {return FindDataSet(logInput);}
virtual TDataSet *DataSet(const char *logInput) const
                        {return GetDataSet(logInput);};
virtual TDataSet *GetInputDS (const char *logInput) const
                        {return GetDataSet(logInput);};

virtual TDataSet *GetDataBase(const char *logInput,const TDatime *td=0);
virtual TDataSet *GetInputDB(const char *logInput)
                       {return GetDataBase(logInput);};
```

Listing 8 : Add methods

```
virtual void AddData (TDataSet *data,const char *dir=".data");
virtual void AddConst(TDataSet *data=0){AddData(data,".const");}
virtual void AddHist(TH1 *h,const char *dir=0);
virtual void AddGarb (TDataSet *data=0){AddData(data,".garb");};
virtual void AddRunco (TDataSet *data=0){AddData(data,".runco");};
virtual void AddRunco (Double_t par,const char *name,const char *comment);
void AddRunCont (TDataSet *data=0){AddRunco(data);}; //alias
```

Chain

- A special StMaker ("Top maker") and a TDataSet as well
- Contains an ordered list of makers
- Makers are added to the chain automatically in order they are created
- Allows to combine makers as components of a complex analysis, reconstruction or simulation

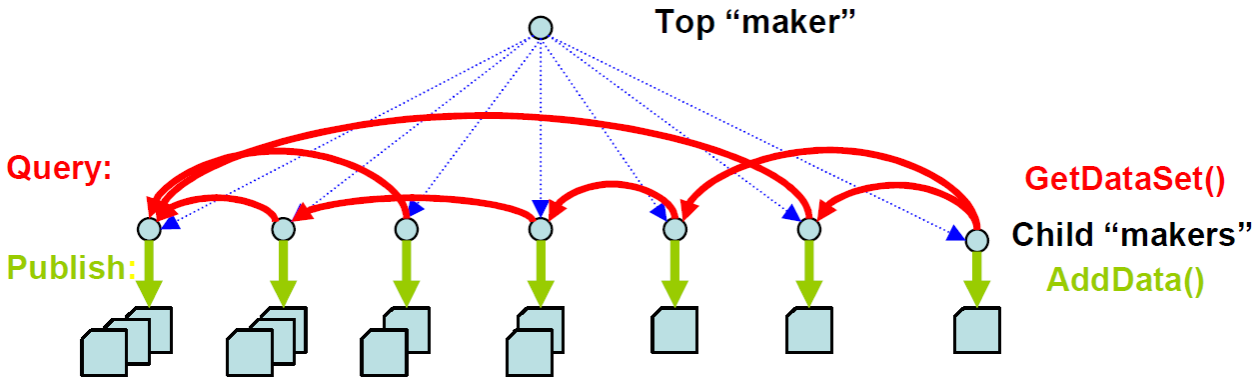
Listing 9 : Example MuDST analysis chain

```
StChain *chain = new StChain;
StMuDstMaker *muDstMk = new StMuDstMaker(0, 0, "", file_list.data(), ".MuDst.root", 200, "MuDst"); // reads MuDST
St_db_Maker *dbMk = new St_db_Maker("StarDb", "MySQL:StarDb"); // DB maker
StEEmcDbMaker *eemcDb = new StEEmcDbMaker("EEmcDbMaker"); // Additional EEMC DB maker

StAnalysisMaker *analysis = new StAnalysisMaker(muDstMk); // Pass StMuDstMaker pointer to your analysis maker

Bool_t status = chain->Init(); // Call Init() for all makers
if (status != kStOK) {
cout << "ERROR: Chain initialisation failed (error " << status << "), not running!" << endl;
return 0;
}
status = chain->EventLoop(); //Process all events in the chain
if ((status != kStOK) && (status != kStEOF)) cout << "WARNING: EventLoop returned error " << status << endl;
status = chain->Finish(); // Call Finish() for all makers, write output data
if (status != kStOK) cout << "WARNING: Chain finished with error " << status << endl;
delete chain;
```

Communication between makers

- Provider consumer model
- Makers are both providers and consumers of data
- The order of communication is fixed (providers before consumers) and is controlled by the order of makers instantiation

1. Maker "consumes" dataset found in a chain by its name
2. Maker fills the output dataset and "provides" it to the chain



Each maker can query the data published by the other makers.

PoS ACAT08 040 (2008)

Listing 10 : Communication functions

```
virtual TDataSet* GetDataSet(const char *logInput) const // Get dataset by its name
virtual void AddData(TDataSet *data, const char *dir=".data") // Publish data
```

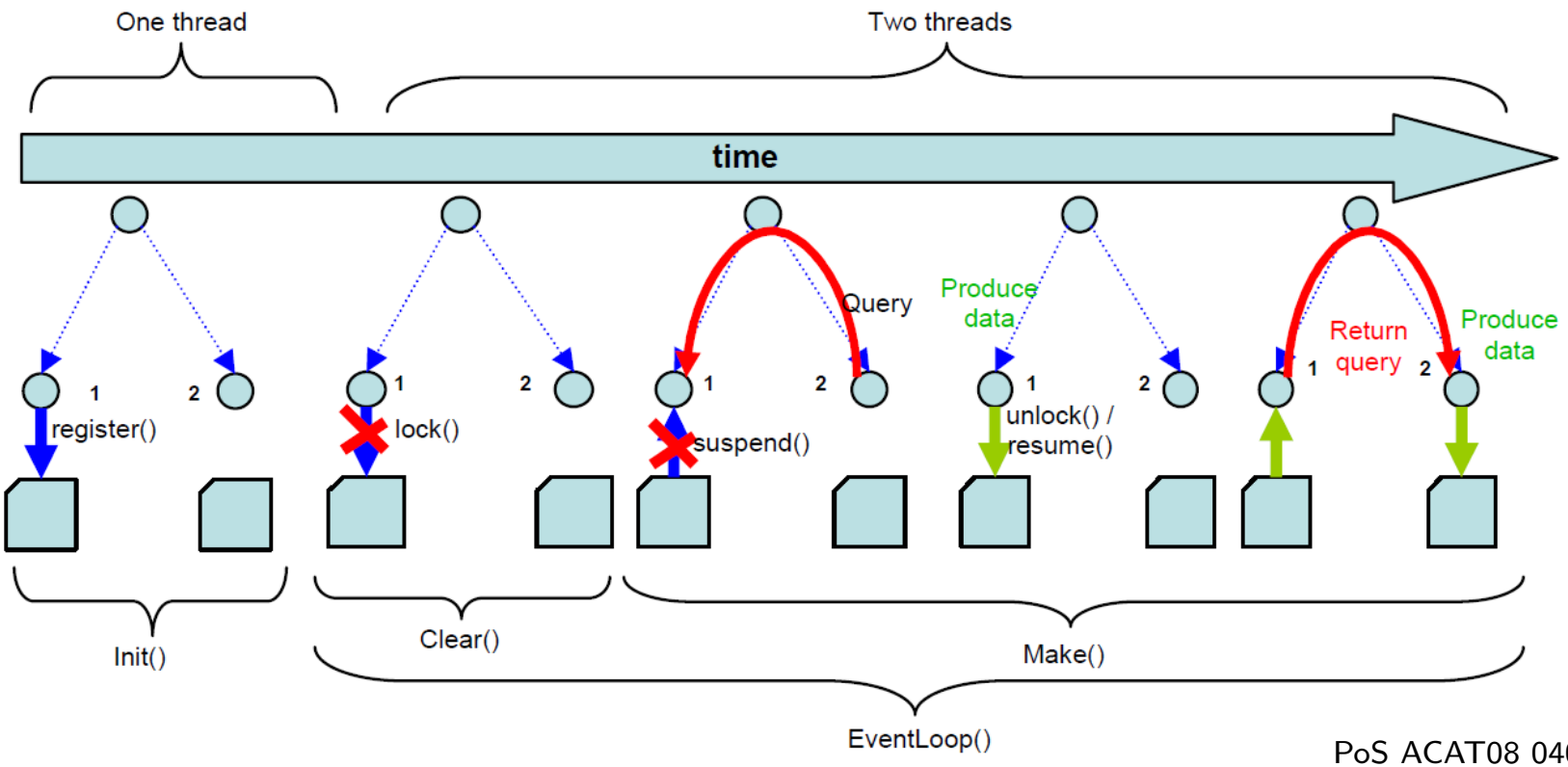
Parallel maker threads possibility

Threads

- Not implemented currently due to limited parallelization - but potentially available
- Makers can run in different threads to take advantage of multi-core CPUs
- StMaker::Make() function executed in parallel
- Pipeline-like execution

Pipeline-like execution

- Maker 1 finishes Make() for event 1 and passes data to the maker 2
- Maker 1 executes Make() for event 2, while maker 2 executes Make() for event 1
- ...



PoS ACAT08 040 (2008)

Efficiency of threads

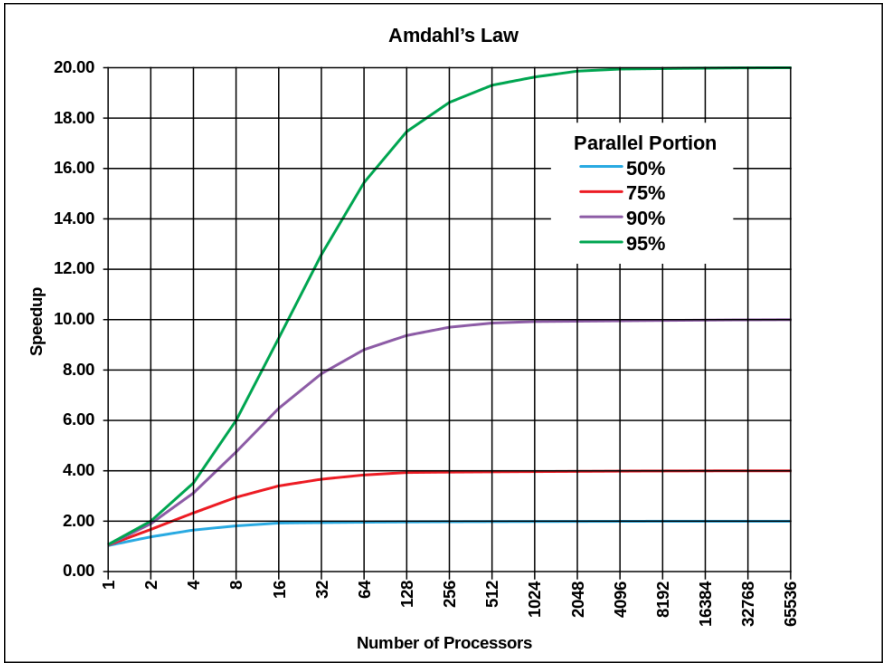
Amdahl's Law

$S(N) = \frac{1}{(1-P)+\frac{P}{N}}$, where:

- $S(N)$ - speedup compared to executing a serial code on a single core, in fact a **theoretical limit**
- P - fraction of the code, which can be made parallel
- N - number of processors used

Think about efficiency!

- Embarassingly parallel mode - we run independent serial jobs (1 job = 1 core) over multiple data (CPU use efficiency is 98%! in case of reconstruction)
- Now with 60% parallel code run on 16 cores, we get a maximum speedup of 2.28, but we occupy 16 cores for that! The efficiency is 14% compared to embarassingly parallel.
- More resources used, so efficiency drops!
- Tracking can be parallelized, but it takes 60% of the time.
- Parallel threads implementation abandoned in favor of high efficiency.



$E(N)$ - efficiency compared to embarassingly parallel

Theoretical maximum values for 16 cores

P	$S(16)$	$E(16)$
99%	13.9	87%
95%	9.14	57%
60%	2.28	14%

Reconstruction Chain - BFC

Reconstruction requires running a few makers per subsystem(detector):

$\Downarrow$

Total number: $\sim$ 100 makers

- Easy with Big Full Chain(BFC), StBFChain class
- Just choose options you need(subsystems and corrections) and go
- BFC can run reconstruction for data or simulation with makers in predefined, fixed order
- Option and maker dependencies form a tree eg:
  istHit option runs *StIstHitMaker*,
  which requires **istCluster**, which runs *StIstClusterMaker*,
  which also requires **istRaw** and runs *StIstRawHitMaker* and **istDB**, running *StIstDbMaker*
  This forms IST reconstruction chain (but no tracking!) and ensures no maker is missing (in most cases)

More details on BFC here:

`https://drupal.star.bnl.gov/STAR/system/files/BFC_HQPC_LK_2015_2_12.pdf`

After setting options, BFC:

1. Creates ordered list of makers
2. Loads shared libraries
3. Runs

If no options are set it will print the list of available options

Listing 11 : How to run BFC?

```
root4star bfc.C
root -l bfc.C
```

Listing 12 : BFC run arguments

```
root4star -l -b -q bfc.C(First, Last, *Chain, *infile) | tee bfc.log

bfc(Int_t First, Int_t Last, // first and last event
 const Char_t *Chain, // BFC options, separated by spaces or commas, case insensitive, order
       irrelevant, put "-" in front to turn off
 const Char_t *infile, // input file .daq (data) or .fzd (simulation)
 const Char_t *outfile, // output file
 const Char_t *TreeFile)
   }
```

List of all BFC options

http://www.star.bnl.gov/cgi-bin/protected/cvsweb.cgi/~checkout~/StRoot/StBFChain/doc/index.html

MuDST analysis

- Write a macro similar to one shown in Listing 9
- Create a chain, StMuDstMaker and your own maker in this macro
- Pass file list to the StMuDstMaker
- The macro is used to run the analysis chain
- Write your own analysis maker and put it in StRoot/StXxxMaker
- Compile (cons) and run macro when finished

Listing 13 : In the Make() function of your maker

```
StMuDst* mMuDst = mMuDstMaker->muDst(); // Get StMuDst
StMuEvent* muEvent = mMuDst->event(); // Get StMuEvent
mMuDst->primaryVertex()->position(); // Get primary vertex position
mMuDst->primaryVertex()->refMult(); // Get reference multiplicity
TObjArray* tracks = mMuDst->primaryTracks(); // Create a TObject array containing the primary
      tracks
TObjArrayIter GetTracks(tracks); // Create an iterator to step through the tracks
StMuTrack* track; // Pointer to a track
while((track = (StMuTrack*)GetTracks.Next())) // Main loop for iterating over tracks
    {histogram[1]->Fill(track->pt());} // Do analysis, fill histograms
```

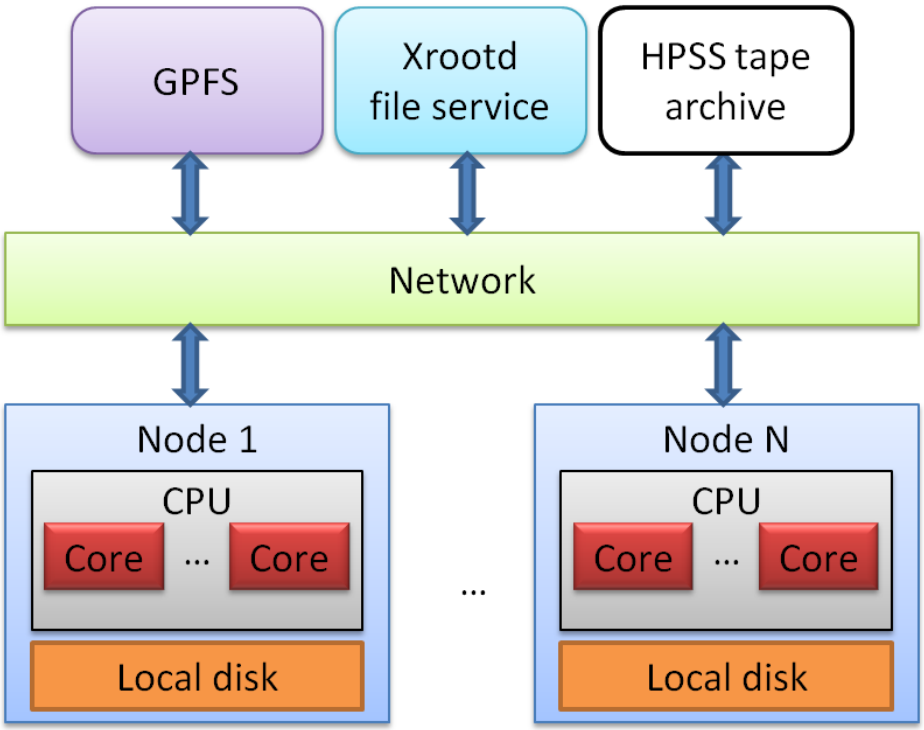
MuDST tutorial

http://rnc.lbl.gov/~jhthomas/public/MuDstTutorial2011.pdf
http://www.star.bnl.gov/public/comp/train/tut/MuDstTutorial06/

RCF cluster

RHIC Computing Facility (RCF)

- Made of multiple nodes connected by network (657 nodes, 15,576 cores)
- Each node has CPU with multiple cores, so a few jobs can run on the same node (1 core = 1 job)
- Provides resources to analyze massive amounts of data in parallel on different nodes
- Nodes accessed through a gatekeeper rssh.rhic.bnl.gov



Remote desktop NX server

https://www.racf.bnl.gov/docs/services/nx

Nodes have access to:

- Local disk, the fastest - use these!
- General Parallel File System (GPFS), disk, slower access than local disk
- High Performance Storage System (HPSS), tape archive, the slowest but stores all data
- Files using Xrootd file server, which redirects connections to the physical location of the file

RCF nodes monitoring table

`http://www.star.bnl.gov/cgi-bin/protected/nova/showMachines.pl`

File catalog

- Allows to locate files and to create file lists
- Very flexible, can return number of events or calculate total number

Listing 14 : Usage

```
get_file_list.pl [-all] -keys keyword[,keyword,...] \
  [-cond keyword=value[,keyword=value,...]] [-start #] [-limit #] [-delim $St] \
  [-onefile] [-o outputfile]
```

Listing 15 : Example - get MuDST files from Run 14

```
get_file_list.pl -keys 'path,filename' -cond 'trgsetupname=AuAu_200_production_low_2014,
      available=1,filetype=daq_reco_MuDst,sanity=1,filename~st_physics_14,storage=local' -
      onefile -delim '/'
```

File catalog manual:

https://drupal.star.bnl.gov/STAR/comp/sofi/filecatalog/user-manual

STAR Scheduler - SUMS

- Used for submitting batch jobs to RCF cluster
- Automatically distributes input files between multiple jobs

Listing 16 : Typical XML job description file

```
<?xml version="1.0" encoding="utf-8" ?>
<job minFilesPerProcess="14" maxFilesPerProcess="20" filesPerHour="3" fileListSyntax="xrootd">
<command> starver SL11d
       cons
       root4star -q -b RunTrackSampleMaker.C\(1e8,\"$FILELIST\",\"$SCRATCH\",\"$JOBID\"\)
       </command>
<SandBox installer="ZIP"> <Package> <File>file:./RunTrackSampleMaker.C</File>
  <File>file:./StRoot</File> </Package> </SandBox>
<input URL="catalog:star.bnl.gov?production=P11id,trgsetupname~pp500_production_2011,filetype=daq_reco_MuDst,sanity
       =1,storage!=HPSS" nFiles="all"/>
<stdout URL="file:./output/stdout/$JOBID.out"/>
<stderr URL="file:./output/stderr/$JOBID.err"/>
<output fromScratch="*.root" toURL="file:/star/institutions/lbl/lkosarz/Upsilon/nSigmaCode/output/output/" />
</job>
```

```
star-submit jobs.xml // Submit jobs described in jobs.xml file
```

Scheduler manual and FAQ:

http://www.star.bnl.gov/public/comp/Grid/scheduler/
http://www.star.bnl.gov/public/comp/Grid/scheduler/faq.html

Good practice

- Test your code locally in interactive mode on a small data sample.
- Once tested submit jobs to RCF.
- Write output of jobs locally to avoid unnecessary traffic over the network.
- Turn off unused MuDST branches to speed up analysis.

Listing 17 : Turn of MuDST branches in the run macro

```
mudst_mk->SetStatus("*",0); // Turn off all MuDST branches
mudst_mk->SetStatus("MuEvent",1); // Turn on MuEvent branch
mudst_mk->SetStatus("PrimaryTracks",1); // Turn on Primary Tracks
```

Turning off MuDST branches

http://www.star.bnl.gov/public/comp/train/tut/MuDstTutorial/select_branches.html

STAR coding and naming standards - Update in preparation! - see Jerome's talk

https://drupal.star.bnl.gov/STAR/comp/sofi/soft-n-libs/standards

https://drupal.star.bnl.gov/STAR/blog/jeromel/c11-and-star-coding-standard-commitee-charges

More tutorials

https://drupal.star.bnl.gov/STAR/comp/sofi/tutorials

Summary

- Presented STAR software basics and crucial topics
- Good start for juniors
- Included useful links
- Everything you need to start your analysis

Thank you!

BACKUP