MpddstSelector

Kacper Skelnik, Wydział Fizyki Politechniki Warszawskiej
kontakt: kacske@wp.pl

27 sierpnia 2019

Spis treści

1 Wstęp

Poniższa instrukcja opisuję programy, napisane w celu umożliwienia wykonania analizy fizycznej za pomocą systemu PROOF. Omawia ona także strukturę tych programów, dzięki czemu czytelnik będzie w stanie z łatwością je modyfikować, lub napisać swoje wersję. Zaproponowany przeze mnie system zawiera pliki: "MpddstSelector.C", "MpddstSelector.h", "ReadAllRootFiles.C", "runAnalyze.sh".

2 PROOF

PROOF jest to nażędzie będące częścią środowiska ROOT, umożliwiające równoległe opracowywanie danych na wielu rdzeniach procesora. Mówiąc dokładniej PROOF rozdziela pracę zadaną przez użytkownika na wiele niezależnych od siebie podzadań. Jest to przeciwieństwo przetwarzania wsadowego. Niestety aby makro mogło wykorzystywać PROOFa należy je napisać w specyficzny sposób. Kolejne rozdziały zawierają opis makra MpddstSelector napisanego w ten sposób.

Aby uruchomić PROOFa należy wpisać w programie ROOT polecenie:

```
root [0] TProof::Open("")
 +++ Starting PROOF-Lite with 8 workers +++
Opening connections to workers: OK (8 workers)
Setting up worker servers: OK (8 workers)
PROOF set to parallel mode (8 workers)
```

```
(TProof *) 0x56173fce9970
root [1]
```

Możemy tu zmodyfikować np. liczbę rdzeni, których ma używać nasza analiza. Robimy to dopisując:

```
root [1] TProof::Open("","workers=2")
 +++ Starting PROOF-Lite with 2 workers +++
Opening connections to workers: OK (2 workers)
Setting up worker servers: OK (2 workers)
PROOF set to parallel mode (2 workers)
(TProof *) 0x56173fce9970
root [2]
```

Więcej informacji pod linkiem `https://root.cern.ch/starting-proof`

3 Selector

Selektor jest to program niezbędny do korzystania z systemu PROOF. Zawiera on dwa pliki: Selector.C, oraz Selector.h. TSelector jest to klasa będąca częścią środowiska ROOT, która umożliwia nam wykorzystanie funkcji które PROOF umie wykonać. Zarys pliku nagłówkowego powinien wyglądać następująco:

```
#ifndef MpddstSelector_h
#define MpddstSelector_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TSelector.h>
#include <TTreeReader.h>
#include <TTreeReaderValue.h>

class MpddstSelector : public TSelector {
public :
        TTreeReader     fReader;
        TTree          *fChain = 0; // deklaracja wskaznika do analizy TTree lub TChain

        TH1D *hpt; // Deklaracja histogramow

        TTreeReaderValue<MpdEvent> MPDEvent_; // deklaracja wskaznika do branch'a TTree

        TFile *opfile; // deklaracja pliku wyjsciowego

        Dwie_galezie(TTree * /*tree*/ =0) MPDEvent_(fReader, "MPDEvent."),
        hpt(0) { } // konstruktor domyslny, nalezy pamietac aby zdefiniowac wszystkie obiekty
        virtual ~Dwie_galezie() { }
        virtual Int_t   Version() const { return 2; }
        virtual void    Begin(TTree *tree);
        virtual void    SlaveBegin(TTree *tree);
        virtual void    Init(TTree *tree);
        virtual Bool_t  Notify();
        virtual Bool_t  Process(Long64_t entry);
```

```
        virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0) { return fChain ?
        fChain->GetTree()->GetEntry(entry, getall) : 0; }
        virtual void    SetOption(const char *option) { fOption = option; }
        virtual void    SetObject(TObject *obj) { fObject = obj; }
        virtual void    SetInputList(TList *input) { fInput = input; }
        virtual TList  *GetOutputList() const { return fOutput; }
        virtual void    SlaveTerminate();
        virtual void    Terminate();

        ClassDef(MpddstSelector,0);
};
#endif

#ifdef MpddstSelector_cxx
void MpddstSelector::Init(TTree *tree){
        mpdloadlibs(); // Ladowanie bibliotek
        if (!tree) return;
                fChain = tree;
        fReader.SetTree(tree); // przypisanie wskaznikowi odpowiedni obiekt TTree
    }
Bool_t MpddstSelector::Notify(){
        return kTRUE; // mozna tu pobrac wskazniki do branch'y
    }
#endif
```

4 Opis funkcji

Opis fuckcji niezbędnych do działania Selectora znajduję się np. pod linkami:

- `https://root.cern.ch/developing-tselector`
- `https://root.cern.ch/processing-proof`

Znajdują się tam też przykładowe programy napisane z użyciem PROOFa, oraz poradniki. Poniżej opiszę najważniejszę funkcję jeszcze raz:

- Begin() - Wywołuję się na początku działania makra. Można zdefiniować w niej np. plik wyjściowy.
- SlaveBegin() - Funkcja wywoływana po funkcji Begin() dla każdego "pracownika". Definiuje się w niej wszystkie histogramy, oraz dodaje się je do OutputList za pomocą funkcji GetOutputList().
- Process() - Funkcja która wywołuję się za każdym razem gdy pobieramy dane z drzewa(TTree) do obróbki. To tutaj powienien znajdować się trzon analizy. Funkcja zwraca warotść logiczną, która nie powinna być nigdzie wykorzystywana (domyślnie kTRUE).
- Terminate() - Funkcaj wywoływana jako ostatnia. Najczęściej wykorzystywana do zapisania gotowych danych do pliku, zaprezentowania ich na histogramie.
- SlaveTerminate() - Funkcja wywoływana na końcu pracy każdego "pracownika".

- Init() - Funkcja wywoływana przy inicjalizacji nowego drzewa(TTree), lub sieci(TChain). Można tu zainicjować wczytywanie bibliotek, sprawdzić istnienie drzewa, policzyć ilość event'ów. Należy pamiętać, aby zadeklarować tutaj drzewo dla TTreeReader'a.
- Notify() - Funkcja wywoływana gdy otfierany jest nowy plik.

5 Secletor.C

W pliku źródłowy powinny znajdować się funkcję odpowiedzialne za analizę danych i stworzenie histogramów:

```
#define MpddstSelector_cxx

#include "MpddstSelector.h"
#include <TH1.h>
#include <TStyle.h>

void MpddstSelector::Begin(TTree * /*tree*/){
    TString option = GetOption();

    opfile = new TFile("file.root", "RECREATE"); // definicja pliku wyjsciowego
}
void MpddstSelector::SlaveBegin(TTree * /*tree*/){
    TString option = GetOption();

    hpt = new TH1D(..definiowanie histogramu..);
    hpt->Sumw2(); // dodanie zarzadzania bledami
    GetOutputList()->Add(hpt); // dodanie histogramu do listy plikow wychodzacych
}
Bool_t MpddstSelector::Process(Long64_t entry){
    fReader.SetLocalEntry(entry);//okreslenie numeru wejscia(zastepuje petle "event loop")
    /* wlasciwa analiza */

    return kTRUE;
}
void MpddstSelector::Terminate(){
    opfile->cd();
    hpt->Write(); // zapisanie histogramu do pliku
}
```

6 Ładowanie danych z plików root

Aby załadować plik rootowy do TChain'a należy uruchomić roota, oraz wpisać w konsolę polecenie:

```
root[0] TChain* myChain = new TChain("cbmsim")
root[1] myChain->AddFile("plik.root")
```

Interesuję nas jednak możliwość analizy wielu plików rootowych na raz. W takim razie łatwo stwierdzić, że powyższa metoda ładowania plików jest nieefektywna. Makro "ReadAllRootFiles.C" służy do zautomatyzowania tego procesu. W zależności od potrzeb można tam ręcznie zdefiniować zakres folderów czytanych przez makro (first way to load root files):

```
for(Int_t iter=BEGIN; iter<=END; ++iter){
```

lub skorzystać z pliku tekstowego z zawierającego kolejne ścieżki dostępu do plików (secound way to load root files):

```
ifstream *istr = new ifstream("./urqmd34-11gev.list.txt");
```

W celu załadowania makra należy wpisać w konsolę:

```
root[1] .L ReadAllRootFiles.C
root[2] ReadAllRootFiles(myChain)
```

7 Uruchamianie procesu analizy danych

Podsumowując, wszystkie komendy które należy wpisać w konsolę, aby uruchomić analizę fizyczną to:

```
root -l -b
root [0] TProof::Open("")
root [1] TChain* myChain = new TChain("cbmsim")
root [2] .L ReadAllRootFiles.C
root [3] ReadAllRootFiles(myChain)
root [4] myChain->SetProof()
root [5] myChain->Process("MpddstSelectorOla.C")
```

W celu optymalizacji procesu pod względem użytkownika został napisany skrypt "runAnalyze.sh". Dzięki temu aby uruchomić analizę wystarczy wpisać w konsolę

```
./runAnalyze.sh
```