Discussion about GPU & GPUPWA

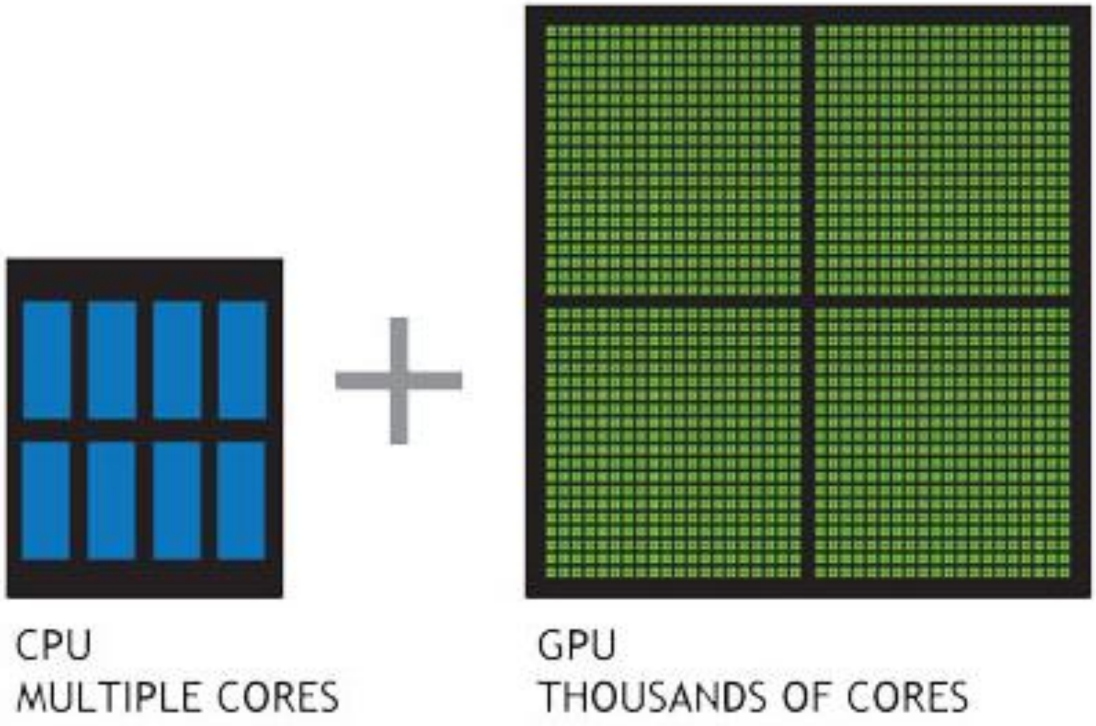
Yan Zhang
USTC

Outline

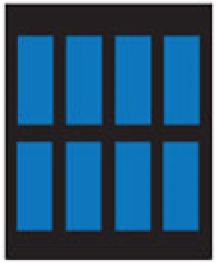
- Introduction to GPU
- Introduction to GPUPWA
- Process
  - Code
  - How to run

Introduction to GPU

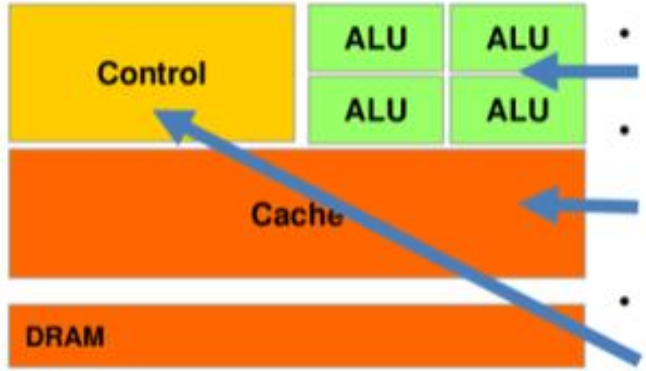
A simple way to understand the difference between a GPU and a CPU is to compare how they process tasks. A CPU consists of a few cores optimized for **sequential serial processing** while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for **handling multiple tasks simultaneously**.



Introduction to GPU



CPU

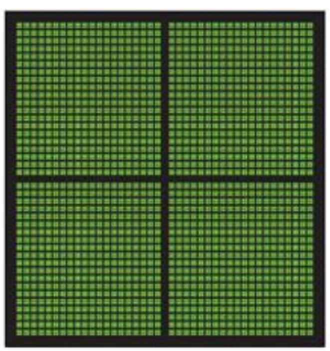


- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

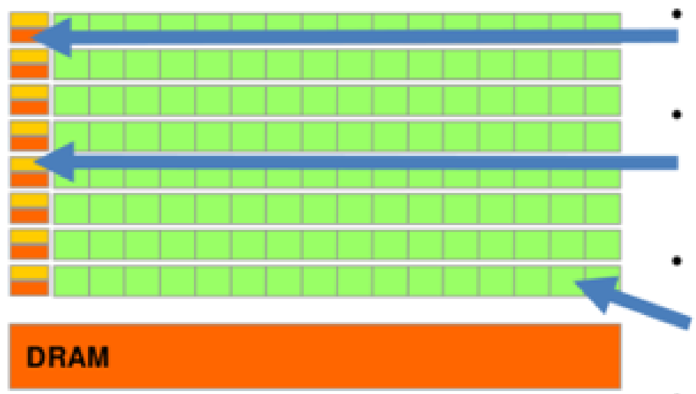
ALU: Arithmetic and logic unit

Cache(SRAM): Static Random Access Memory

DRAM: Dynamic Random Access Memory



GPU

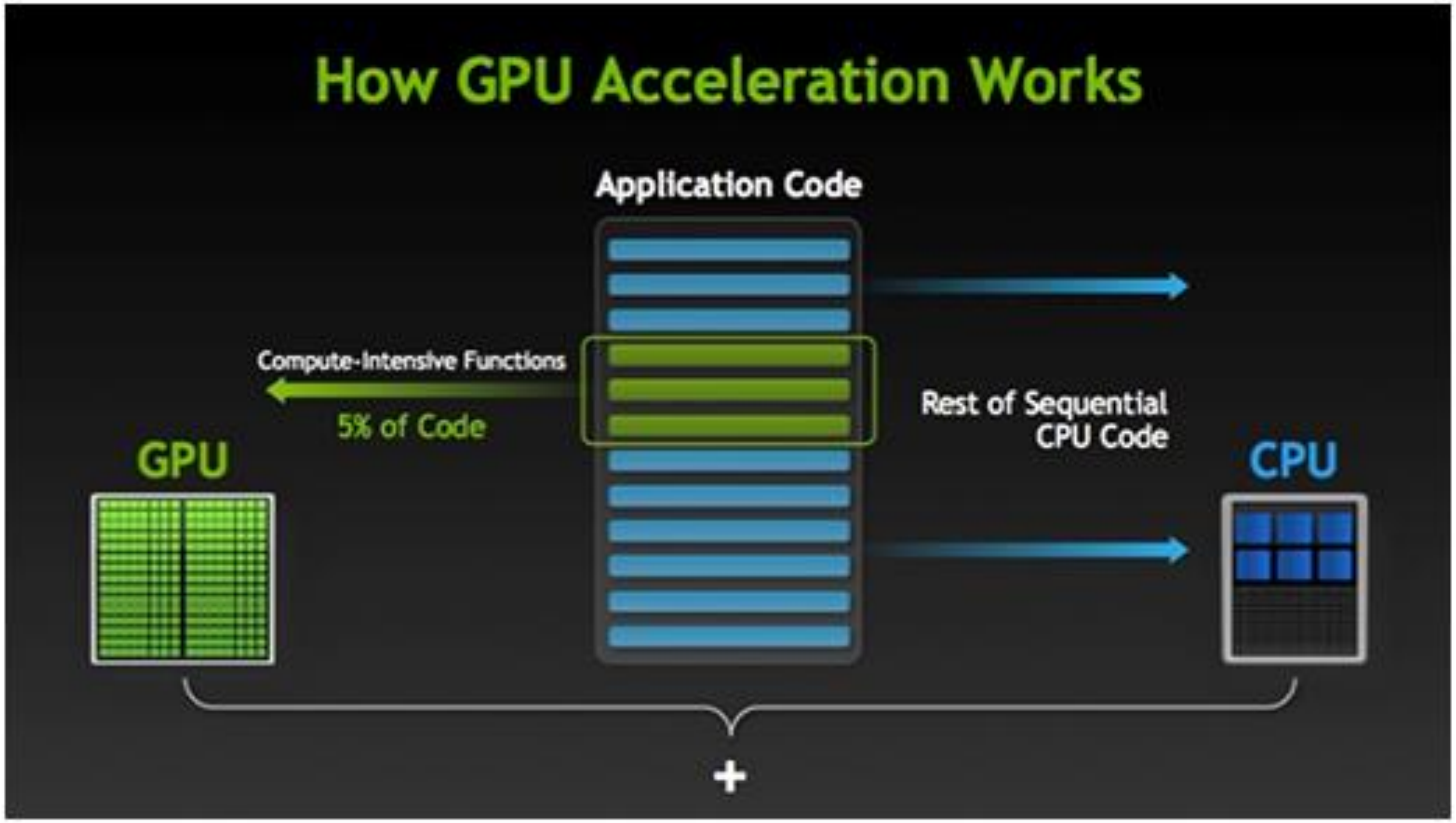


- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

Introduction to GPU

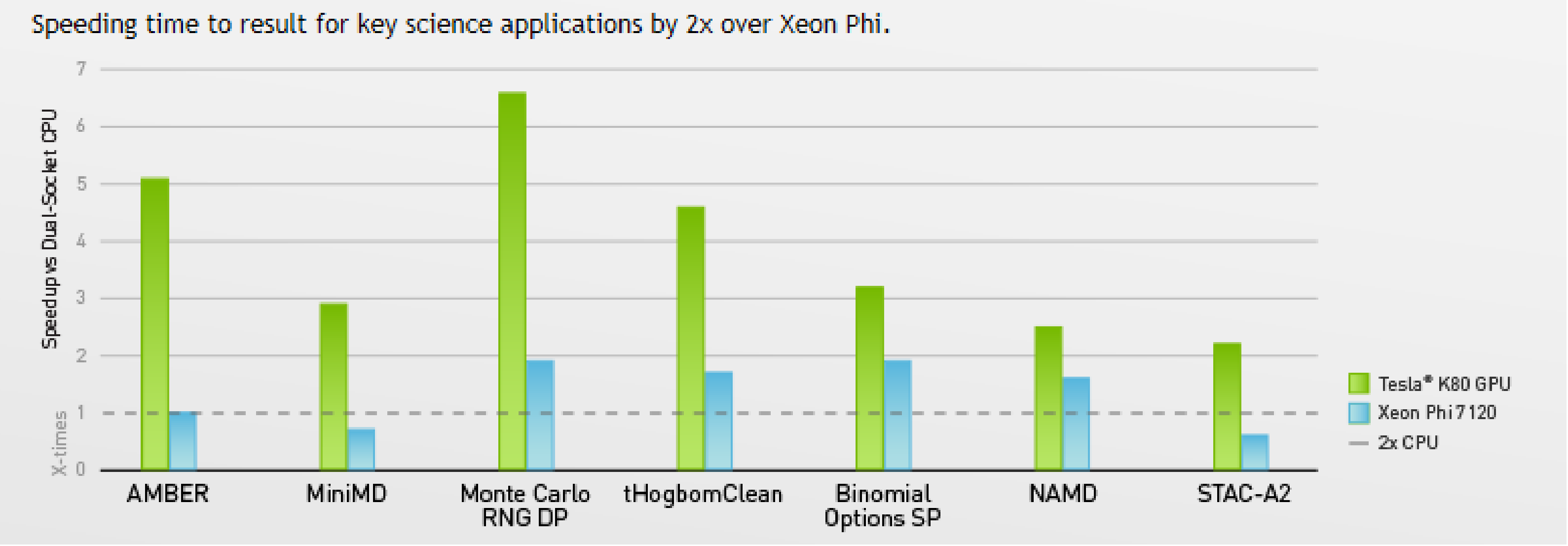
HOW GPUs ACCELERATE SOFTWARE APPLICATIONS

GPU-accelerated computing **offloads compute-intensive portions** of the application to the GPU, while the **remainder of the code** still runs on the CPU. From a user's perspective, applications simply run much faster.



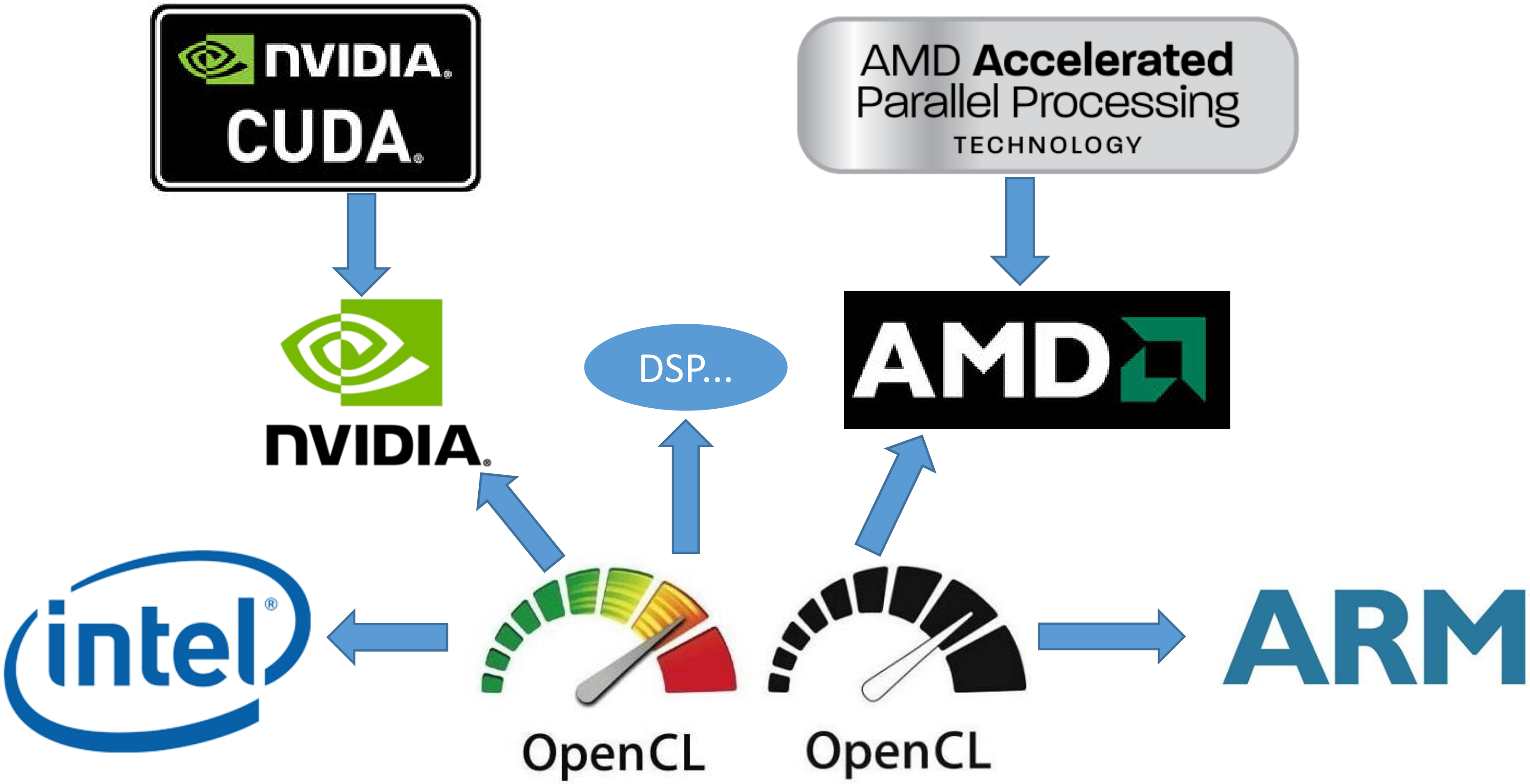
Introduction to GPU

GPU Accelerated Computing is revolution in High Performance Computing



Organization	Application	GPU Speed-up over Xeon Phi
Tokyo Institute of Technology	CFD Diffusion	2.6x
Xcelerit	Monte-Carlo LIBOR Swap Pricing	2.2x - 4x
Georgia Tech	Synthetic Aperture Radar	2.1x
CGGVeritas	Reverse Time Migration	2.0x
Paralution	BLAS & SpMV	2.0x
Univ. of Wisconsin-Madison	WRF (Weather Forecasting)	1.8x
University Erlangen-Nuremberg	Medical Imaging- 3D Image Reconstruction	7x
Delft University	Drug Discovery	3x

Introduction to CUDA & OpenCL



CUDA is NVIDIA's **parallel computing architecture** that enables dramatic increases in computing performance by harnessing the power of the GPU (graphics processing unit).

Introduction to our machine

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.26                 Driver Version: 375.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K40m          Off  | 0000:03:00.0     Off |                    0 |
| N/A   24C    P0    66W / 235W |      0MiB / 11439MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K40m          Off  | 0000:04:00.0     Off |                    0 |
| N/A   22C    P0    65W / 235W |      0MiB / 11439MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K40m          Off  | 0000:82:00.0     Off |                    0 |
| N/A   23C    P0    64W / 235W |      0MiB / 11439MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K40m          Off  | 0000:83:00.0     Off |                    0 |
| N/A   21C    P0    70W / 235W |      0MiB / 11439MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2609 v4 @ 1.70GHz
stepping        : 1
microcode       : 184549407
cpu MHz         : 1699.956
cache size      : 20480 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 8
```

× 16

Environment for GPUPWA

GPUPWA provides a C++ interface to **covariant tensor manipulation** and PWA fits without bothering the user with GPU internals.

```
Distributor ID: Scientific
Description:    Scientific Linux release 6.9 (Carbon)
Release:        6.9
Codename:       Carbon
```

```
gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-18)
Copyright (C) 2010 Free Software Foundation, Inc.
```

```
ROOT 5.34/32 (v5-34-32@v5-34-32, Jun 23 2015, 17:58:02 on linuxx8664gcc)
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue_Jan_10_13:22:03_CST_2017
Cuda compilation tools, release 8.0, V8.0.61
```

```
export ROOTSYS=/root534
export CUDAROOT=/usr/local/cuda-8.0
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${CUDAROOT}/lib64:${ROOTSYS}/lib:/usr/local/lib
export PATH=${PATH}:${CUDAROOT}/bin:$ROOTSYS/bin:
export GPUPWA=~/gpu
export GPUPWA_GPU_NR=2
export DISPLAY=0.0
export _NVIDIA=1
```

Environment for GPUPWA

1. You need to set the path of OpenCL driver:
   setenv CUDAROOT /usr/local/cuda-8.0 (tcsh)
   export CUDAROOT=/usr/local/cuda-8.0 (bash)
2. You need to set the path of Root (different to those installed in lxslc):
   export ROOTSYS=/root/root534
3. You should add the two libs to LD_LIBRARY_PATH.
   export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${CUDAROOT}/lib64:${ROOTSYS}/lib:/usr/local/lib
4. You need to set the GPUPWA path.
   export GPUPWA=/xxx/xxx/the/path/to/your/GPUPWA/project/
5. You need to set DISPLAY.
   export DISPLAY=0.0
6. You should set the below environment variables if required when apply a front-test gpu account.
   export GPUPWA_GPU_NR=your_val
   export _NVIDIA=your_val
   (export GPU_DEVICE_ORDINAL=your_val
   export BRT_ADAPTER=your_val)
7. The below set may be also useful:
   export PATH=${PATH}:${CUDAROOT}/bin:$ROOTSYS/bin

GPUPWA

/besfs/users/zhangyan/gpupwa-f17422302b0012facb46505d3c9d3c39788c4492.zip

```
-rwxr-xr-x. 1 zhangyan zhangyan 17515 Jul 12  2016 Changelog.txt
-rwxr-xr-x. 1 zhangyan zhangyan   340 Jul 12  2016 commands.mk
-rwxr-xr-x. 1 zhangyan zhangyan   387 Jul 12  2016 depends.mk
-rwxr-xr-x. 1 zhangyan zhangyan  8943 Jul 12  2016 details.txt
-rwxr-xr-x. 1 zhangyan zhangyan  7668 Jul 12  2016 Doxyfile
-rwxr-xr-x. 1 zhangyan zhangyan 10179 Jul 12  2016 Doxyfile_win
-rwxr-xr-x. 1 zhangyan zhangyan 22481 Jul 12  2016 example.txt
-rwxr-xr-x. 1 zhangyan zhangyan   819 Jul 12  2016 flags.mk
drwxrwxr-x. 4 zhangyan zhangyan 69632 Sep 21 22:52 GammaKK
drwxrwxr-x. 3 zhangyan zhangyan  4096 Sep 15 20:52 GammaKKpi
drwxrwxr-x. 4 zhangyan zhangyan  4096 Sep 15 20:52 GammaKKUserAmpInterface
drwxrwxr-x. 5 zhangyan zhangyan 12288 Sep 21 22:53 GPUPWA
-rwxr-xr-x. 1 zhangyan zhangyan  2817 Jul 12  2016 gpupwafiles.txt
-rwxr-xr-x. 1 zhangyan zhangyan  1789 Jul 12  2016 GPUPWA.sln
-rwxr-xr-x. 1 zhangyan zhangyan 11422 Jul 12  2016 GPUPWA.vcproj
-rwxr-xr-x. 1 zhangyan zhangyan 11692 Jul 12  2016 howtos.txt
-rwxr-xr-x. 1 zhangyan zhangyan 48799 Jul 12  2016 log
-rwxr-xr-x. 1 zhangyan zhangyan 10093 Jul 12  2016 mainpage.txt
-rwxr-xr-x. 1 zhangyan zhangyan  1077 Jul 12  2016 Makefile
-rwxr-xr-x. 1 zhangyan zhangyan   776 Jul 12  2016 paths.mk
drwxrwxr-x. 3 zhangyan zhangyan  4096 Sep 15 20:52 PiPiPi
-rwxr-xr-x. 1 zhangyan zhangyan   795 Jul 12  2016 target.mk
drwxrwxr-x. 3 zhangyan zhangyan  4096 Sep 15 20:52 Testanalysis
```

GPUPWA

```
-rwxr-xr-x. 1 zhangyan zhangyan  2096 Jul 12  2016 GPUStreamInputVector.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  5425 Jul 12  2016 GPUStreamInputVector.h
-rwxr-xr-x. 1 zhangyan zhangyan 22763 Jul 12  2016 GPUStreamTensor.cpp
-rwxr-xr-x. 1 zhangyan zhangyan 23917 Jul 12  2016 GPUStreamTensor.h
-rwxr-xr-x. 1 zhangyan zhangyan    23 Jul 12  2016 GPUTensor.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  1069 Jul 12  2016 GPUTensor.h
-rwxr-xr-x. 1 zhangyan zhangyan  4058 Jul 12  2016 GPUUnFactorizedPartialWave.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  3179 Jul 12  2016 GPUUnFactorizedPartialWave.h
-rwxr-xr-x. 1 zhangyan zhangyan  2132 Jul 12  2016 GPUUnFactorizedRadiativePartialWave.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  1811 Jul 12  2016 GPUUnFactorizedRadiativePartialWave.h
-rwxr-xr-x. 1 zhangyan zhangyan   962 Jul 12  2016 GPUUserBasicPartialWave.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  2382 Jul 12  2016 GPUUserBasicPartialWave.h
-rwxr-xr-x. 1 zhangyan zhangyan  6031 Jul 12  2016 GPUUserPartialWave.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  2071 Jul 12  2016 GPUUserPartialWave.h
-rwxr-xr-x. 1 zhangyan zhangyan  1042 Jul 12  2016 GPUUserPropagator.cpp
-rwxr-xr-x. 1 zhangyan zhangyan   933 Jul 12  2016 GPUUserPropagator.h
-rwxr-xr-x. 1 zhangyan zhangyan  6933 Jul 12  2016 GPUUserRadiativePartialWave.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  2284 Jul 12  2016 GPUUserRadiativePartialWave.h
-rwxr-xr-x. 1 zhangyan zhangyan  5561 Jul 12  2016 Makefile
-rwxr-xr-x. 1 zhangyan zhangyan 16375 Jul 12  2016 MnFunctionCross2.cxx
-rwxr-xr-x. 1 zhangyan zhangyan  1381 Jul 12  2016 MnFunctionCross2.h
drwxrwxr-x. 3 zhangyan zhangyan  4096 Sep 21 22:29 Opencl_interface
-rwxr-xr-x. 1 zhangyan zhangyan 64727 Jul 12  2016 Orbitals.cl
-rwxr-xr-x. 1 zhangyan zhangyan   457 Jul 12  2016 ParaCfg.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  1655 Jul 12  2016 ParaCfg.h
-rwxr-xr-x. 1 zhangyan zhangyan  1478 Jul 12  2016 PrepareKernels.cpp
-rwxr-xr-x. 1 zhangyan zhangyan   392 Jul 12  2016 PrepareKernels.h
-rwxr-xr-x. 1 zhangyan zhangyan 29725 Jul 12  2016 Propagators.cl
-rwxr-xr-x. 1 zhangyan zhangyan   414 Jul 12  2016 ResCfg.cpp
-rwxr-xr-x. 1 zhangyan zhangyan  1179 Jul 12  2016 ResCfg.h
-rwxr-xr-x. 1 zhangyan zhangyan   186 Jul 12  2016 Status.h
-rwxr-xr-x. 1 zhangyan zhangyan  1266 Jul 12  2016 Style.h
-rwxr-xr-x. 1 zhangyan zhangyan 98650 Jul 12  2016 Tensors.cl
-rwxr-xr-x. 1 zhangyan zhangyan 44873 Jul 12  2016 UserAmplitude.cl
drwxrwxr-x. 2 zhangyan zhangyan  4096 Sep 18 21:25 _x86_64
```

DIY is ok here.

GPUPWA example : J/psi → Gamma K K

```
drwxr-xr-x. 2 zhangyan zhangyan  4096 Sep 21 22:10 data
-rwxr-xr-x. 1 zhangyan zhangyan   210 Sep 21 22:10 files.txt
-rwxr-xr-x. 1 zhangyan zhangyan 18838 Sep 21 22:46 GammaKK.cpp
-rwxr-xr-x. 1 zhangyan zhangyan   982 Sep 21 22:10 Makefile
-rwxr-xr-x. 1 zhangyan zhangyan   516 Sep 21 22:39 para.inp
-rwxr-xr-x. 1 zhangyan zhangyan   269 Sep 21 22:44 res.inp
```

files.txt

```
# files.txt
# Example configuration file for ConfigFile class
ParameterFile = para.inp
ResonanceFile = res.inp
DataFile = data/zeroplustwoplus_data_100k_01.root
MCFile1 = data/zeroplustwoplus_phsp_100k_01.root
```

para.inp

```
f0_mag = 2.26 2 0 500
f0_phase = 1.11 0.3 -3.14159 3.14159

f20_mag = 1 2 0 500
f20_phase = 1.0 -0.3 -3.14159 3.14159

f21_mag = 0.03 2 0 500
f21_phase = 1.0 -0.3 -3.14159 3.14159

f22_mag = 0.2 2 0 500
f22_phase = 1.0 -0.3 -3.14159 3.14159

f40_mag = 0.0 -2 0 500
f40_phase = 1.0 -0.3 -3.14159 3.14159

f41_mag = 0.0 -2 0 500
f41_phase = 1.0 -0.3 -3.14159 3.14159

f42_mag = 0.0 -2 0 500
f42_phase = 1.0 -0.3 -3.14159 3.14159

bg_mag = 10.0 -50 0 500
```

res.inp

```
f0_mass  = 2.15 -1 -20 20      # comment
f0_width = 0.0486  -1 999 999  # comment
f2_mass  = 2.0010 -1 999 999   # comment
f2_width = 0.133  -1 999 999   # comment
f4_mass  = 2.000  -1 999 999   # comment
f4_width = 0.03  -1 999 999    # comment
```

GPUPWA example : J/psi → Gamma K K

```cpp
#include "../GPUPWA/GPUStreamTensor.h"
#include "../GPUPWA/GPUComputedTensor.h"
#include "../GPUPWA/GPUMetricTensor.h"
#include "../GPUPWA/GPUOrbitalTensors.h"
#include "../GPUPWA/GPUPropagatorBreitWigner.h"
#include "../GPUPWA/GPUPropagatorMassDependentBreitWigner.h"
#include "../GPUPWA/GPUPartialWaveAnalysis.h"
#include "../GPUPWA/GPUPWAAmplitudeCalculator.h"
#include "../GPUPWA/GPUStreamInputRootFileVector.h"
#include "../GPUPWA/GPUStreamInputTextFileVector.h"
#include "../GPUPWA/GPUPlotset.h"
#include "../GPUPWA/GPUChi2FitConstraint.h"
#include "../GPUPWA/GPUDataDependentObjectType.h"
#include "../GPUPWA/GPUFactorizedRadiativePartialWave.h"

// We also need some stuff from root
#include "TFile.h"
#include "TRandom3.h"

// And some general C/C++ stuff
#include <ctime>
#include <iomanip>
#include <fstream>
#include <iostream>
```

```
2016 GPUTensor.h
2016 GPUUnFactorizedPartialWave.cpp
2016 GPUUnFactorizedPartialWave.h
2016 GPUUnFactorizedRadiativePartialWave.cpp
2016 GPUUnFactorizedRadiativePartialWave.h
2016 GPUUserBasicPartialWave.cpp
2016 GPUUserBasicPartialWave.h
2016 GPUUserPartialWave.cpp
2016 GPUUserPartialWave.h
2016 GPUUserPropagator.cpp
2016 GPUUserPropagator.h
2016 GPUUserRadiativePartialWave.cpp
2016 GPUUserRadiativePartialWave.h
2016 Makefile
```

GPUPWA example : J/psi → Gamma K K

```
GPUPartialWaveAnalysis * myanalysis = new GPUPartialWaveAnalysis("Gamma KK Analysis","files.txt",2);


// For now we will store and use MC at index 1
myanalysis->SetMCIndex(1);

GPUStreamInputRootFileVector & k_plus = * new GPUStreamInputRootFileVector(myanalysis,myanalysis->GetDataFile(), "t","p
x1","py1","pz1","E1");
GPUStreamInputRootFileVector & k_minus = * new GPUStreamInputRootFileVector(myanalysis,myanalysis->GetDataFile(), "t","
px2","py2","pz2","E2");




/* We can use weights for the data events, e.g. to do a background subtraction. Here we just set the weights to 1 for a
ll
data used*/
myanalysis->SetEventWeights(1);
//      myanalysis->SetEventWeights(0.2,1);

vector<int> ivec;
vector<float> fvec;
ivec.push_back(28434);
fvec.push_back(1.0f);
ivec.push_back(2173);
fvec.push_back(-0.5f);
ivec.push_back(885);
fvec.push_back(0.25f);
myanalysis->SetEventWeights(ivec, fvec, 0);
```

GPUPWA example : J/psi → Gamma K K

```
/**********kshort1pion mode*****************************************/
GPUPropagatorBreitWigner & propagator_k892zero1   = * new GPUPropagatorBreitWigner("k892zero", vec_kshort1pion2);
GPUPropagatorBreitWigner & propagator_kappazero1   = * new GPUPropagatorBreitWigner("kappazero", vec_kshort1pion2);

//        GPUPropagatorBreitWigner & propagator_k892phsp1   = * new GPUPropagatorBreitWigner("k892phsp", vec_kshort1pion2);
/*********kshort2pion mode******************************************/
GPUPropagatorBreitWigner & propagator_k892zero2 = * new GPUPropagatorBreitWigner("k892zero", vec_kshort2pion2);
GPUPropagatorBreitWigner & propagator_kappazero2 = * new GPUPropagatorBreitWigner("kappazero", vec_kshort2pion2);

//        GPUPropagatorBreitWigner & propagator_k892phsp2 = * new GPUPropagatorBreitWigner("k892phsp", vec_kshort2pion2);

/*********kshort1kshort2 mode****************************************/
//      GPUPropagatorBreitWigner & propagator_a980   = * new GPUPropagatorBreitWigner("a980", vec_kshort1kshort2);
GPUPropagatorBreitWigner & propagator_a2   = * new GPUPropagatorBreitWigner("a2", vec_kshort1kshort22);

GPUPropagatorA980 & propagator_a980 = * new GPUPropagatorA980("a980", vec_kshort1kshort22, meta, mpi, mK, mK,mKs, mKs,m
etap, mpi);


// so we end up with the complete orbital part of the amplitudes
GPUStreamTensor2 & Orbital_f2_0_MN = t2_mn;
GPUStreamTensor2 & Orbital_f2_1_MN = -g * (((Jpsi%Jpsi)|t2_mn )* B2_psi_gamma_f2);
GPUStreamTensor2 & Orbital_f2_2_MN = (Gamma % (t2_Nm|jpsi))*  B2_psi_gamma_f2;
        /* The '%' charcter denotes the outer product of two tensors - you can find the complete
catalogue of permitted operations on the GPUPWA Wiki*/

// And the same for the f4s...
GPUStreamTensor4 &  t4_mnuv = xorbitals.Spin4OrbitalTensor();
GPUStreamTensor4 &  t4_mnUV = moveindices(t4_mnuv);
GPUStreamTensor4 &  t4_UVmn = trans_3412(t4_mnUV);
GPUStreamTensor4 &  t4_mnuV = movelastindex(t4_mnuv);
GPUStreamTensor4 &  t4_Vmnu = trans_2341(t4_mnuV);

GPUStreamScalar &  B4_psi_gamma_f4 = jpsiorbitals.Barrier4();
GPUStreamTensor2 & Orbital_f4_0_MN = (t4_UVmn|(jpsi%jpsi)) * B2_psi_gamma_f2;
GPUStreamTensor2 & Orbital_f4_1_MN = -g * (t4_mnuv|((jpsi%jpsi)%(jpsi%jpsi))) * B4_psi_gamma_f4;
GPUStreamTensor2 & Orbital_f4_2_MN = Gamma %(t4_Vmnu|(jpsi%jpsi%jpsi)) * B4_psi_gamma_f4;
```

/besfs/users/zhangyan/GPUPWA_2_1.pdf ------Section10.3

GPUPWA example : J/psi → Gamma K K

```
// A scalar
GPUFactorizedRadiativePartialWave & wave0 = * new GPUFactorizedRadiativePartialWave(Orbital_f0_MN,propagator2,"f0");
// And a 2+ resonance, with three waves.
GPUFactorizedRadiativePartialWave & wave1 = * new GPUFactorizedRadiativePartialWave(Orbital_f2_0_MN,propagator1,"f20");
GPUFactorizedRadiativePartialWave & wave2 = * new GPUFactorizedRadiativePartialWave(Orbital_f2_1_MN,propagator1,"f21");
GPUFactorizedRadiativePartialWave & wave3 = * new GPUFactorizedRadiativePartialWave(Orbital_f2_2_MN,propagator1,"f22");
// And again the same for the 4+
GPUFactorizedRadiativePartialWave & wave4 = * new GPUFactorizedRadiativePartialWave(Orbital_f4_0_MN,propagator3,"f40");
GPUFactorizedRadiativePartialWave & wave5 = * new GPUFactorizedRadiativePartialWave(Orbital_f4_1_MN,propagator3,"f41");
GPUFactorizedRadiativePartialWave & wave6 = * new GPUFactorizedRadiativePartialWave(Orbital_f4_2_MN,propagator3,"f42");
```

```
// And we are ready to run - so lets check how long this took...
clock_t startup = clock();
/* Here we perform the preparations for the Monte Carlo calculation.
This will read the MC file, compute and sum all amplitude and interference
terms and write them to a file. This has to be called only once for a constant set
of resonances, as long as their masses and widths are not changed */
myanalysis->MCIntegral();

/* Reset the cache for the MC (at index 1) in order to free some memory. If you have "out of memory"
or "Cal ressource allocation" errors, remove the comment from the next line */
myanalysis->Reset(1);

...
```

```
/* Now we can do the fit. Currently you can use either of the following fitters:
- FUMILI (the Minuit2 implementation,
- OLDFUMILI (the BES II implementation, in general requires fewest iterations),
- MINUIT (with numerical gradients),
- MINUITGRAD (with analytical gradients,
- MINUITMINOS (MNUIT (numerical gradients) followed by a modified MINOS error estimation)
*/
//myanalysis->DoFit(GPUPartialWaveAnalysis::OLDFUMILI);
myanalysis->DoMultiFit(GPUPartialWaveAnalysis::OLDFUMILI, 1, 3);
```

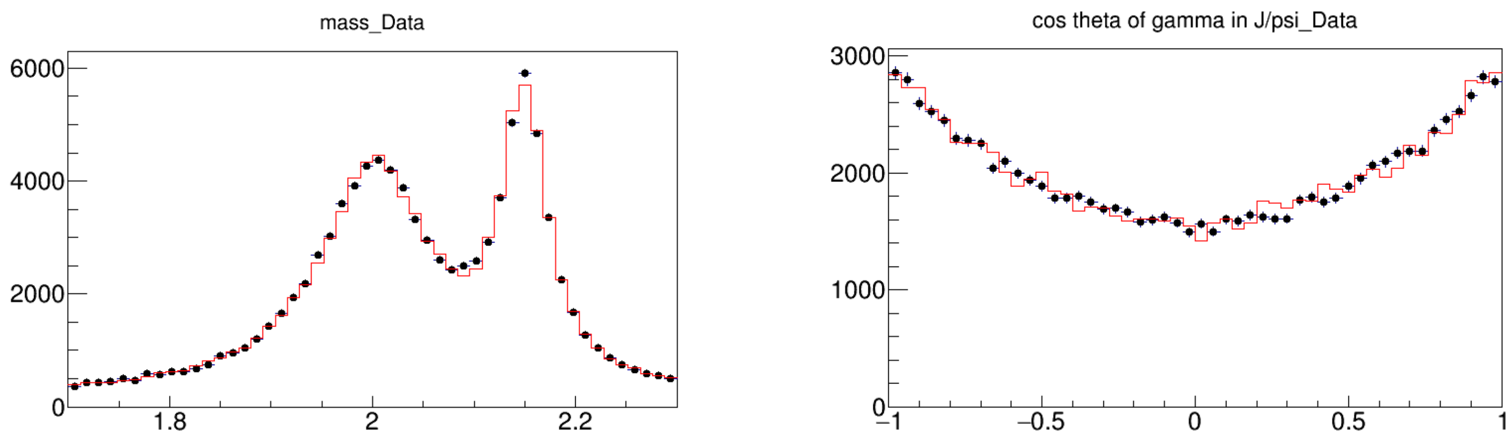
GPUPWA example : J/psi → Gamma K K

```
// Create the quantities to be plotted
GPUStreamScalar &ct_g=costheta(gamma);

/* .. yes, we can also rotate and boost vectors - this is of course meaningless for covariant amplitudes, but nice for
plotting */
GPUStreamVector & kr= lorentzrotation(k_plus,x);
GPUStreamVector & xr= lorentzrotation(x,x);
GPUStreamVector & kb= lorentzboost(kr,xr);
GPUStreamScalar & ct_k=costheta(kb);
GPUStreamScalar & ph_k=phi(kb);

// So first we create a set of plots, which takes care of the formatting and file handling
GPUPlotset * plotset = new GPUPlotset();
//plotset->AddGraph(mygraph);
int nwaves =  myanalysis->GetWaves()->GetNActiveWaves();

plotset->AddPlots(mX.Plot("mass","mass;mX [GeV]",50,1.7,2.3,dcs, nwaves,true));
plotset->AddPlots(ct_g.Plot("cos_theta_g","cos theta of gamma in J/psi;cos theta(gamma) ",50,-1,1,dcs, nwaves, true));
plotset->AddPlots(ct_k.Plot("cos_theta_K","cos theta of K in X;cos theta(K) ",20,-1,1,dcs, nwaves,true));
plotset->AddPlots(ph_k.Plot("phi_K","phi of K in X;phi(K) ",20,-3.1416,3.1416,dcs, nwaves,true));
// Nicely format the plots (root defaults are REALLY UGLY!)
plotset->Format();
// Write a postscript file with the plots. The erguments are currently ignored.
// -> Fix the argument issue
plotset->WritePsfile("testout1.ps",1,1);
// Write a rootfile with the plots
plotset->WriteRootfile("testout1.root");
```



GPUPWA example : J/psi → Gamma K K

```
lrwxrwxrwx. 1 zhangyan zhangyan       42 Sep 21 22:30 binfiles -> /home/zhangyan/gpu/GPUPWA/_common/binfiles
drwxr-xr-x. 2 zhangyan zhangyan     4096 Sep 21 22:10 data
-rwxr-xr-x. 1 zhangyan zhangyan      210 Sep 21 22:10 files.txt
-rwxrwxr-x. 1 zhangyan zhangyan 19441276 Sep 24 23:06 gammakk
-rw-rw-r--. 1 zhangyan zhangyan      517 Sep 24 23:06 GammaKKAnalysis_Amplitude_MC_Integral
-rw-rw-r--. 1 zhangyan zhangyan        4 Sep 24 23:06 GammaKKAnalysis_counter.cnt
-rwxr-xr-x. 1 zhangyan zhangyan    18838 Sep 21 22:46 GammaKK.cpp
-rw-rw-r--. 1 zhangyan zhangyan   730755 Sep 24 23:06 gmon.out
-rwxr-xr-x. 1 zhangyan zhangyan      982 Sep 21 22:10 Makefile
-rw-rw-r--. 1 zhangyan zhangyan      262 Sep 24 23:06 multifitresults_GammaKKAnalysis_0.txt
-rwxr-xr-x. 1 zhangyan zhangyan      516 Sep 21 22:39 para.inp
-rwxr-xr-x. 1 zhangyan zhangyan      269 Sep 21 22:44 res.inp
-rw-rw-r--. 1 zhangyan zhangyan   149612 Sep 24 23:06 testout1.ps
-rw-r--r--. 1 zhangyan zhangyan    35180 Sep 24 23:06 testout1.root
drwxrwxr-x. 2 zhangyan zhangyan     4096 Sep 24 23:05 _x86_64
```

Multifitresults_GammaKKAnalysis_0.txt

```
Fit                    1: Minimum Likelihood : -38180.139639743472799
Fit                    2: Minimum Likelihood : -38180.11549846563139
Fit                    3: Minimum Likelihood : -38180.126488532609073
Best Likelihood obtained was: -38180.139639743472799
```

```
************************************************************
       Fit coverged

       Minimum Likelihood: -38180.1

       Estimated Distance to Minimum: -0.000133672

************************************************************
Gamma KK Analysis
Using the following partial waves:
           Wave Name  Magn. in Magn. out  Phase in Phase outDynamic inDynamic outDynamic inDynamic out
________________________________________________________________________________________________________________
                 f20         1  0.957244         1    (fixed)     2.001    (fixed)     0.133    (fixed)
                 f21      0.03 0.0303596         1    (fixed)     2.001    (fixed)     0.133    (fixed)
                 f22       0.2  0.202545         1    (fixed)     2.001    (fixed)     0.133    (fixed)
                  f0      2.26   1.99345      1.11    1.04418      2.15    (fixed)    0.0486    (fixed)
________________________________________________________________________________________________________________
************************************************************
```

GPUPWA example : J/psi → Gamma K K

```
drwxrwxr-x. 8 ustc ustc      4096 Jul 12  2016 gpupwa
-rw-r--r--. 1 ustc ustc       288 Sep 25 14:59 gpupwa2.1
```

If you have got a GPUPWA copy, and set all the environment, then

you can try to run a gpupwa program:

Source gpupwa2.1

cd gpupwa

make

cd GammaKK

cp _x86_64/gammakk ./

./gammakk

And wait for the program finish

P.S : If you made some modification in gpupwa/GammaKK/, you can make in GammaKK directory (if you have done make at the top directory at least one time).

GPUPWA example : J/psi → Gamma K K

If you want to submit jobs to GPU queue, you can:

Source gpupwa2.1

cd gpupwa

make

cd GammaKK

cp _x86_64/gammakk ./

nohup gammakk & or nohup gammakk > *.out 2>&1 &

And wait for the program finish

ssh ustc@210.45.78.29

Passwd: P@ss#p0rt

BACK UP

● 构造振幅的符号约定和宇称限制

表 2.1: 构造协变耦合振幅所用的主要符号约定

	母粒子	子粒子1	子粒子2
自旋	J	s	σ
宇称	η_J	η_s	η_σ
Helicity	δ	λ	ν
自旋z−分量	m	m_s	m_σ
动量	p	q	k
能量	p_0	q_0	k_0
质量	W	m	μ
能量/质量	γ_J	γ_s	γ_σ
波函数	$\phi^*(\delta)$	$\omega(\lambda)$	$\epsilon(-\nu)$

$(J+s+\sigma+l)=2$，是偶数，则不需要 $\epsilon_{\mu\nu\sigma\gamma}p^\mu$的出现。

$[\eta_a\eta_b\eta_c(-1)^l=1]$

构造似然函数

$$P(\xi_1,\xi_2,\cdots,\xi_n)=\prod_{i=1}^{n}\frac{\omega(\xi_i)\varepsilon(\xi_i)}{\int d\xi\omega(\xi)\varepsilon(\xi_i)}$$

$\sigma=\int d\xi\omega(\xi)\varepsilon(\xi)$ 总截面

$\omega(\xi)=\frac{d\sigma}{d\phi}$ 产生几率密度

$\varepsilon(\xi)$ 被探测和选择到的几率

$$\ln P(\xi_1,\xi_2,\cdots\xi_n)=\sum_{i=1}^{n}\ln\left(\frac{\omega(\xi_i)}{\int d\xi\omega(\xi)\varepsilon(\xi)}\right)+\underbrace{\sum_{i=1}^{n}\ln\varepsilon(\xi_i)}^{\text{可以忽略}} \xrightarrow[L=P(\xi_1,\xi_2,\cdots,\xi_n)]{} \ln L=\sum_{i=1}^{n}\ln\left(\frac{d\sigma}{d\phi}\Big/\sigma\right)$$

Monte Carlo积分

$$\sigma = \int d\xi\, \omega(\xi)\varepsilon(\xi) = \sum_i \Delta\xi_i\, \omega(\xi_i)\varepsilon(\xi_i) = \frac{1}{N_{gen}} \sum_i N_{gen}\Delta\xi_i \omega(\xi_i)\varepsilon(\xi_i)$$

$$\sigma = \frac{1}{N_{gen}} \sum_i N_{\xi_i}\, \omega(\xi_i) = \frac{1}{N_{gen}} \sum_{k=1}^{N_{MC}} \omega(\xi_k)$$

$$\frac{d\sigma}{d\phi} = |A|^2$$

$$A = \sum_i \Lambda_i \sum_m A(i,m)$$

Λ_i 不同分波振幅的复耦合系数

$$A(i,m) = \frac{1}{\sqrt{N_m}} \phi_\mu(m) U_i^\mu$$

$\phi_\mu(m)$ 初态粒子的极化四矢量

N_m 磁量子数m可取值的个数

U_i^μ 协变张量振幅公式

$$\frac{d\sigma}{d\phi} = |A|^2 = \frac{1}{2}\sum_{i,j} \Lambda_i \, \Lambda_j^* \sum_{\mu=1,2} U_i^{\mu} U_j^{*\mu}$$

$$\left\{\begin{array}{l} \Lambda_i = a \cdot e^{ib} = A + iB \\ \Lambda_j^* = C - iD \end{array}\right. \longrightarrow P_{ij} = \Lambda_i \Lambda_j^* = \left(\Lambda_j \Lambda_i^*\right)^* = P_{ji}^*$$

$$\longrightarrow \mathrm{m}_{ij} = P_{ij} F_{ij} = \left(P_{ji} F_{ji}\right)^* = m_{ji}^*$$

$$\left\{\begin{array}{l} U_i^{\mu} = m + in \\ U_j^{*\mu} = p - iq \end{array}\right. \longrightarrow F_{ij} = \frac{1}{2}\sum_{\mu=1,2} U_i U_j^{*\mu} = \left(\frac{1}{2}\sum_{\mu=1,2} U_j U_i^{*\mu}\right)^* = F_{ji}^*$$

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{pmatrix} = M^{\dagger}$$

n个独立的分波

$$\frac{d\sigma}{d\phi} = \sum_{i,j=1}^{n} m_{ij} = \sum_{i=1}^{n} m_{ii} + 2\sum_{i<j} \mathrm{Re}(m_{ij}) = \sum_{i=1}^{n} P_{ii}F_{ii} + 2\sum_{i<j}\left[\mathrm{Re}(P_{ij}F_{ij})\right]$$

$$= \sum_{i=1}^{n} P_{ii}F_{ii} + \underbrace{2\sum_{i<j}\left[\mathrm{Re}(P_{ij})\mathrm{Re}(F_{ij}) - \mathrm{Im}(P_{ij})\mathrm{Im}(F_{ij})\right]}_{\text{i分波和j分波的干涉项}}$$

✓ 轨道角动量波函数协变形式

$$
\begin{aligned}
\tilde{T}^{(1)}_{\mu} &= \tilde{r}_{\mu} B_1(Q) \\
\tilde{T}^{(2)}_{\mu\nu} &= \{\tilde{r}_{\mu}\tilde{r}_{\nu} - \frac{1}{3}\tilde{r}^2 \tilde{g}_{\mu\nu}\} B_2(Q) \\
\tilde{T}^{(3)}_{\mu\nu\lambda} &= \{\tilde{r}_{\mu}\tilde{r}_{\nu}\tilde{r}_{\lambda} - \frac{1}{5}\tilde{r}^2 [\tilde{g}_{\mu\nu}\tilde{r}_{\lambda} + \tilde{g}_{\nu\lambda}\tilde{r}_{\mu} + \tilde{g}_{\lambda\mu}\tilde{r}_{\nu}]\} B_3(Q) \\
\tilde{T}^{(4)}_{\mu\nu\lambda\sigma} &= \{\tilde{r}_{\mu}\tilde{r}_{\nu}\tilde{r}_{\lambda}\tilde{r}_{\sigma} - \frac{1}{7}\tilde{r}^2 [\tilde{g}_{\mu\nu}\tilde{r}_{\lambda}\tilde{r}_{\sigma} + \tilde{g}_{\mu\lambda}\tilde{r}_{\nu}\tilde{r}_{\sigma} + \tilde{g}_{\mu\sigma}\tilde{r}_{\nu}\tilde{r}_{\lambda} + \tilde{g}_{\nu\lambda}\tilde{r}_{\mu}\tilde{r}_{\sigma} + \tilde{g}_{\nu\sigma}\tilde{r}_{\mu}\tilde{r}_{\lambda} + \tilde{g}_{\lambda\sigma}\tilde{r}_{\mu}\tilde{r}_{\nu}] \\
&\quad + \frac{1}{35}\tilde{r}^4 [\tilde{g}_{\mu\nu}\tilde{g}_{\lambda\sigma} + \tilde{g}_{\mu\lambda}\tilde{g}_{\nu\sigma} + \tilde{g}_{\mu\sigma}\tilde{g}_{\nu\lambda}]\} B_4(Q)
\end{aligned}
$$

✓ 势垒因子（分母部分）

$$
\begin{aligned}
B_1(Q) &= \sqrt{\frac{2}{Q^2+Q_0^2}} \qquad Q_0 = \frac{\hbar c}{R} = \frac{197.32697\mathrm{MeV\cdot fm}}{R} \\
B_2(Q) &= \sqrt{\frac{13}{Q^4+3Q^2Q_0^2+9Q_0^4}} \\
B_3(Q) &= \sqrt{\frac{227}{Q^6+6Q^4Q_0^2+45Q^2Q_0^4+225Q_0^6}}
\end{aligned}
$$

✓ 其他符号

$BW^{(a)}_{(bc)}$	$a \to bc$ 的 Breit-Wigner 传播子；
ε	全反对称张量；
g	Lorentz 矩阵；
$\tilde{g}_{\mu\nu}$	$g_{\mu\nu} - \frac{P_\mu P_\nu}{P_\psi^2}$；
$\tilde{p}_\mu$	$p^\nu \tilde{g}_{\mu\nu}$；
$B_L(Q_{abc})$	$a \to b + c$ 的 Blatt-Weisskopf 势垒因子 [36]，(其中L 为角动量)。

其中，对 $a \to bc$ 衰变来说，$r = p_b - p_c$(p_b, p_c 分别为 b，c 粒子的动量），$\tilde{r}.\tilde{r}$ 为四矢量的点乘：$\tilde{r}_0.\tilde{r}_0 - \tilde{r}_1.\tilde{r}_1 - \tilde{r}_2.\tilde{r}_2 - \tilde{r}_3.\tilde{r}_3$，Blatt-Weisskopf 势垒因子 $B_i(Q)$ 分别为：

$\psi_\mu(m)$	J/ψ 粒子的极化四矢量；
P	J/ψ 粒子的四动量；
p_1	赝标介子1 的四动量；
p_2	赝标介子2 的四动量；
p	$p_1 - p_2$；