# Debugging

- **Integers! Integers everywhere!**
- **Strange behaviour with ending the program.**
- **Pointers about function pointers.**

# Stupid, stupid compiler creatures

- **Consider this code:**

```
__constant__ __device__ double dev_params[2];
thrust::device_vector<double> theEvents;

struct GaussianFunctor{
  __device__ double operator()(double x){
    double mean = dev_params[0];
    double sigma = dev_params[1];
    return (-log(sigma)-pow((x-mean)/sigma,2));
  }
};

void FitFcn (int &npar, double *deriv, double &fun,
             double *param, int flg){
  cudaMemcpyToSymbol("dev_params",
                     param,
                     2*sizeof(double),
                     0,
                     cudaMemcpyHostToDevice);
  fun = thrust::transform_reduce(theEvents.begin(),
                                 theEvents.end(),
                                 GaussianFunctor(),
                                 0,
                                 thrust::plus<double>());
}
```

```
int main(int argc, char** argv) {
  host_vector<double> theEventsH(10000,0);

  TRandom mygaus(118);
  TMinuit minuit(2);
  minuit.DefineParameter(0,"mean", 0, 0.1, -1,1);
  minuit.DefineParameter(1,"sigma",1,0.1,0.5,1.5);

  for (int i=0; i<theEventsH.size(); ++i) {
    theEventsH[i]=mygaus.Gaus(0,1);//mean, sigma
  }
  theEvents = theEventsH;

  minuit.SetFCN(&FitFcn);
  minuit.Migrad();

  return 0;
}
```

- **No flaw obvious to the naked eye. Yet somehow the fit fails to converge. Minuit throws up its hands and says "could not find convergence" and "STATUS FAILED" and other *mean, nasty* things that hurts our feelings, Precious!:**

```
FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-04
FCN=-6667 FROM MIGRAD    STATUS=INITIATE       74 CALLS          75 TOTAL
                     EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
```

```
 EXT PARAMETER                CURRENT GUESS       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean        -3.99357e-04   1.00000e-01   0.00000e+00   3.75603e+03
  2  sigma        1.00000e+00   1.00000e-01   0.00000e+00   0.00000e+00
MIGRAD FAILS TO FIND IMPROVEMENT
EIGENVALUES OF SECOND-DERIVATIVE MATRIX:
       -1.2678e+00  3.2678e+00
MINUIT WARNING IN HESSE
============== MATRIX FORCED POS-DEF BY ADDING 1.271055 TO DIAGONAL.
FCN=-57283 FROM HESSE     STATUS=NOT POSDEF     10 CALLS         119 TOTAL
                    EDM=1.10218e+06    STRATEGY= 1      ERR MATRIX NOT POS-DE
 EXT PARAMETER                 APPROXIMATE        STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean         8.99336e-01   4.01389e-02   1.38982e-02  -2.86369e+04
  2  sigma        5.22747e-01   2.06442e-03   1.58252e-03   3.85461e+04
MIGRAD FAILS TO FIND IMPROVEMENT
MIGRAD TERMINATED WITHOUT CONVERGENCE.
FCN=-70606 FROM MIGRAD    STATUS=FAILED        167 CALLS         168 TOTAL
                    EDM=0.897104    STRATEGY= 1      ERR MATRIX NOT POS-DEF
 EXT PARAMETER                 APPROXIMATE        STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean         9.99998e-01   1.26098e-05  -0.00000e+00   7.19530e+01
  2  sigma        5.00000e-01   2.19047e-06  -0.00000e+00   0.00000e+00
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=1
 8.043e-11  2.389e-12
 2.389e-12  3.663e-12
ERR MATRIX NOT POS-DEF
```

```
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2
        1  0.13920   1.000  0.139
        2  0.13920   0.139  1.000
 ERR MATRIX NOT POS-DEF

terminate called after throwing an instance of 'thrust::system::system_error'
  what():  unload of CUDA runtime failed
Aborted (core dumped)
```

- **Throw it into Mount Doom!**

# Debugging in gory detail

- **Debugging involves lots of failures and asking many stupid questions; these are mine.**
- **First question: That log expression is complex. Possibly a mistake was made? Rewrite for simplicity:**

```
struct GaussianFunctor{
  __device__ double operator() (double x){
    double mean = dev_params[0];
    double sigma = dev_params[1];
    double nll = (x - mean);
    nll /= sigma;
    nll *= nll;
    nll  = exp(-0.5*nll);
    nll /= sqrt(2*3.14159)*sigma;
    return -2*log(nll);
  }
};
```

- **That's not it; same results.**
- **Could we have made a mistake in transferring data or parameters? Let's print some numbers.**

```
#include "cuPrintf.cu"
// ...
```

```
    printf("Gaussian: %f %f %f %f %f\n", x, mean,
           sigma, nll, -2*log(nll))
```

- **Representative output:**

```
Gaussian: 0.380885 -0.000077 1.000000 0.371018 1.983008
Gaussian: -0.720619 -0.000077 1.000000 0.307731 2.357058
Gaussian: 1.392728 -0.000077 1.000000 0.151239 3.777782
Gaussian: 3.267297 -0.000077 1.000000 0.001917 12.513608
Gaussian: 0.017704 -0.000077 1.000000 0.398879 1.838192
Gaussian: 0.287120 -0.000077 1.000000 0.382824 1.920358
```

- **Well, that looks reasonable.**
- **Ok. Stupid question. What happens if we calculate the thing on the CPU?**

```
void FitFcn (int &npar, double *deriv, double &fun,
             double *param, int flg) {
  //cudaMemcpyToSymbol("dev_params", param, 2*sizeof(double),0, cudaMemcpyHost
  //fun = thrust::transform_reduce(theEvents.begin(), theEvents.end(), Gaussia

  double ret = 0;
  double mean = param[0];
  double sigma = param[1];
  // NB, theEventsH made global
  for (int i = 0; i < theEventsH.size(); ++i) {
    double nll = (theEventsH[i] - mean);
    nll /= sigma;
```

```
    nll *= nll;
    nll = exp(-0.5*nll);
    nll /= sqrt(2*3.14159265)*sigma;
    ret -= 2*log(nll);
  }
  fun = ret;
}
```

- **Now it converges!**

```
FCN=28389.4 FROM MIGRAD    STATUS=INITIATE       10 CALLS          11 TOTAL
                     EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
 EXT PARAMETER                CURRENT GUESS       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean         0.00000e+00   1.00000e-01   1.00167e-01   1.05880e+02
  2  sigma        1.00000e+00   1.00000e-01   2.01358e-01  -1.06039e+01
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=28389.1 FROM MIGRAD    STATUS=CONVERGED      32 CALLS          33 TOTAL
                     EDM=4.42776e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean        -5.29407e-03   1.00050e-02   8.23170e-04  -1.31161e-03
  2  sigma        1.00052e+00   7.07448e-03   1.16415e-03   6.58625e-03
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=1
 1.001e-04  4.114e-08
 4.114e-08  5.005e-05
```

```
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2
        1  0.00058   1.000  0.001
        2  0.00058   0.001  1.000
```

- **So what is the difference between CPU and GPU? We already checked that the GPU calculations look reasonable, but are they the same as the CPU ones? Limit the data set to ten events and look again.**

```
void FitFcn (int &npar, double *deriv,
             double &fun, double *param, int flg) {
  static int callnum = 0;

  cudaMemcpyToSymbol("dev_params",
                     param,
                     2*sizeof(double),
                     0,
                     cudaMemcpyHostToDevice);
  double funDev = thrust::transform_reduce(theEvents.begin(),
                                           theEvents.end(),
                                           GaussianFunctor(),
                                           0,
                                           thrust::plus<double>());

  double ret = 0;
  double mean = param[0];
  double sigma = param[1];
  for (int i = 0; i < theEventsH.size(); ++i) {
```

```
      double nll = (theEventsH[i] - mean);
      nll /= sigma;
      nll *= nll;
      nll = exp(-0.5*nll);
      nll /= sqrt(2*3.14159265)*sigma;
      ret -= 2*log(nll);
      std::cout << "Gaussian CPU: "
                << theEventsH[i] << " "
                << mean << " "
                << sigma << " "
                << nll << " "
                << -2*log(nll) << " "
                << std::endl;
    }
    fun = ret;
    std::cout << "Call " << callnum++ << " : " << funDev << " " << fun << std::e
  }
```

- **Numbers are the same:**

```
Gaussian: -0.504328 0.000000 1.000000 0.351301 2.092223
Gaussian: 1.564153 0.000000 1.000000 0.117393 4.284450
Gaussian: -0.001761 0.000000 1.000000 0.398942 1.837879
Gaussian: 1.937444 0.000000 1.000000 0.061067 5.591566
Gaussian: 1.348653 0.000000 1.000000 0.160675 3.656742
Gaussian: -0.767503 0.000000 1.000000 0.297165 2.426937
Gaussian: -1.197120 0.000000 1.000000 0.194858 3.270973
Gaussian: -0.293744 0.000000 1.000000 0.382097 1.924162
```

```
Gaussian: -0.230165 0.000000 1.000000 0.388514 1.890852
Gaussian: -2.112619 0.000000 1.000000 0.042830 6.301034
Gaussian CPU: -0.504328 0 1 0.351301 2.09222
Gaussian CPU: 1.56415 0 1 0.117393 4.28445
Gaussian CPU: -0.00176133 0 1 0.398942 1.83788
Gaussian CPU: 1.93744 0 1 0.061067 5.59157
Gaussian CPU: 1.34865 0 1 0.160675 3.65674
Gaussian CPU: -0.767503 0 1 0.297165 2.42694
Gaussian CPU: -1.19712 0 1 0.194858 3.27097
Gaussian CPU: -0.293744 0 1 0.382097 1.92416
Gaussian CPU: -0.230165 0 1 0.388514 1.89085
Gaussian CPU: -2.11262 0 1 0.04283 6.30104
```

- **But hold up! Why are `fun` and `funDev` different?**

```
Call 0 : 28 33.2768
```

- **28 is the sum of the integers 2, 4, 1, 5, 3, 2, 3, 1, 1, 6...**
- **Puzzlement! Confusion! Why is the GPU doing integer addition?**

```
double funDev = thrust::transform_reduce(theEvents.begin(),
                                         theEvents.end(),
                                         GaussianFunctor(),
                                         0,
                                         thrust::plus<double>());
```

- **Could it be an issue with recreating the functors every time?**

```
static GaussianFunctor gauss;
static thrust::plus<double> plus;
double funDev = thrust::transform_reduce(theEvents.begin(),
                                         theEvents.end(),
                                         gauss,
                                         0,
                                         plus);
```

- **No, that's not it.**
- **Ok, this is silly, but how about that zero?**

```
static GaussianFunctor gauss;
static thrust::plus<double> plus;
double dummy = 0;
double funDev = thrust::transform_reduce(theEvents.begin(),
                                         theEvents.end(),
                                         gauss,
                                         dummy,
                                         plus);
```

- **That's it! The fit now converges using the GPU number! (Tears hair, goes for long walk to cool down.)**
- **So what on earth happened here? The compiler has two functions to choose from:**

```
transform_reduce(iterator<double>, iterator<double>,
```

```
                  double unary_op<double>,
                  double,
                  double binary_op<double, double>);
transform_reduce(iterator<double>, iterator<double>,
                  int unary_op<double>,
                  int,
                  int binary_op<int, int>);
```

**but we are giving it this:**

```
transform_reduce(iterator<double>, iterator<double>,
                  double unary_op<double>,
                  int,
                  double binary_op<double, double>);
```

- **The compiler (silently!) picks the second option! It converts doubles to ints without saying a word about it!**
- **Now, one minor problem remains:**

```
terminate called after throwing an instance of 'thrust::system::system_error'
  what():  unload of CUDA runtime failed
Aborted (core dumped)
```

- **Debug this by commenting things out!**
- **Turns out to be caused by the `theEvents` device vector.**
- **Conjecture: Global variables are destroyed after the CUDA runtime quits? Let's see what happens if we make it a function-local variable:**

```
void FitFcn(int &npar, double *deriv, double &fun, double *param, int flg){
  static int callnum = 0;
  static thrust::device_vector<double> theEvents;
  if (0 == callnum) theEvents = theEventsH;
```

- **That works. But static variables are in some sense globals too:**

  > **Destructors (12.4) for initialized objects of static storage duration (declared at block scope or at namespace scope) are called as a result of returning from main and as a result of calling exit (18.3). These objects are destroyed in the reverse order of the completion of their constructor or of the completion of their dynamic initialization.**

- **Something to do with the order of initialisation? Let's try getting this under control:**

```
thrust::device_vector<double>* theEvents = 0;
// ...
  double funDev = thrust::transform_reduce(theEvents->begin(),
                                           theEvents->end(),
                                           gauss,
                                           dummy,
                                           plus);
// ...
  theEvents = new thrust::device_vector<double>();
  (*theEvents) = theEventsH;
```

- **Now it doesn't crash. But...I didn't invoke the destructor!**

```
delete theEvents; // Still works.
```

- **Thrust memory management is unfortunately a black box.**

# Function pointer clarification

- **Function pointers cannot be passed around. Hence the indirection of creating `void*` and doing a `reinterpret_cast`:**

```
typedef double (*dev_fcn_ptr) (double, double*, int*);
__constant__ __device__ double dev_params[100];
__device__ int idxs[1000];

struct GeneralFunctor {

double operator (double x) {
  return (*(reinterpret_cast<dev_fcn_ptr>(func1)))
           (x, dev_params, idxs + myIndex);
}

void* func1;
int myIndex;
};
```

- **An alternative: Work strictly with structs:**

```
struct FunctionBase {
  double operator (double x, double* params, int* idxs) = 0;
};

struct FunctionImplementation
  : public FunctionBase
```

```
{
  double operator (double x, double* params, int* idxs) {
    return x*params[idxs[0]];
  }
};

struct GeneralFunctor {
  double operator (double x) {
    return (*fcn)(x, dev_params, idxs + myIndex);
  }

  FunctionBase* fcn;
  int myIndex;
};
```

# Request for proposals

- **For the last lecture: What is unclear?**
- **What point should be re-iterated or covered in more detail?**