Generators Profiling

Tim Martin, Warwick
Work-in-progress update
28th April 2020

Intro

- Working with some local installs of common generator SW
  - Pythia8 - installed & working, not had too much attention yet.
  - Sherpa + OpenLoops - working
  - MadGraph5_aMC@NLO - recently working
- Starting to look at some general profiling data, with aim to gain understanding of both generator structure and vtune. Look for leads.

Sherpa Z+jj

- Split into two jobs, integration and event gen.
- Modification to example run card to make the integration less taxing
  - `NJET:=4; LJET:=2,3,4; QCUT:=20.;` -> `NJET:=2; LJET:=2; QCUT:=20.;`

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in im application performance.

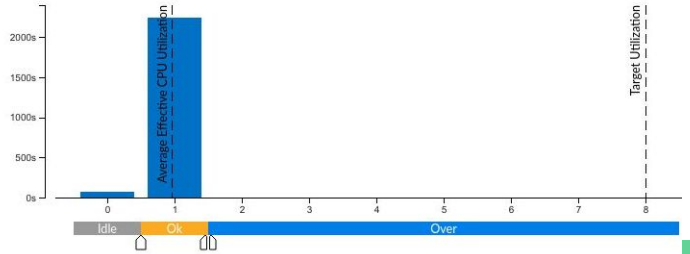
Function	Module	CPU Time
NNPDFDriver::lh_polint	libNNPDFSherpa.so	171.255s
__ieee754_log_avx	libm.so.6	160.965s
NNPDFDriver::xfx	libNNPDFSherpa.so	117.127s
_int_malloc	libc.so.6	77.189s
__muldc3	libgcc_s.so.1	54.670s
[Others]		1671.912s

*N/A is applied to non-summable metrics.

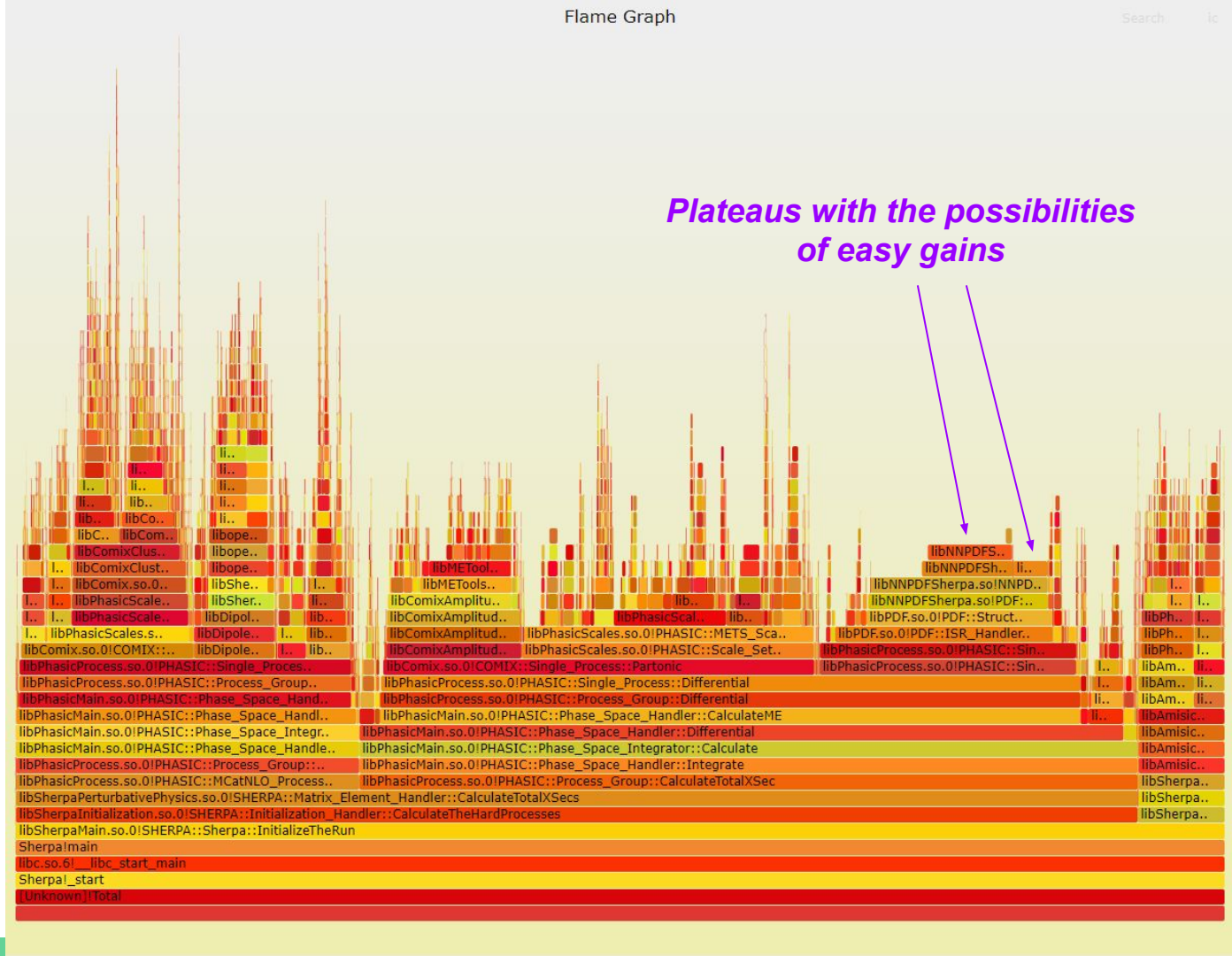
Single-threaded so no MT investigation here

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin an

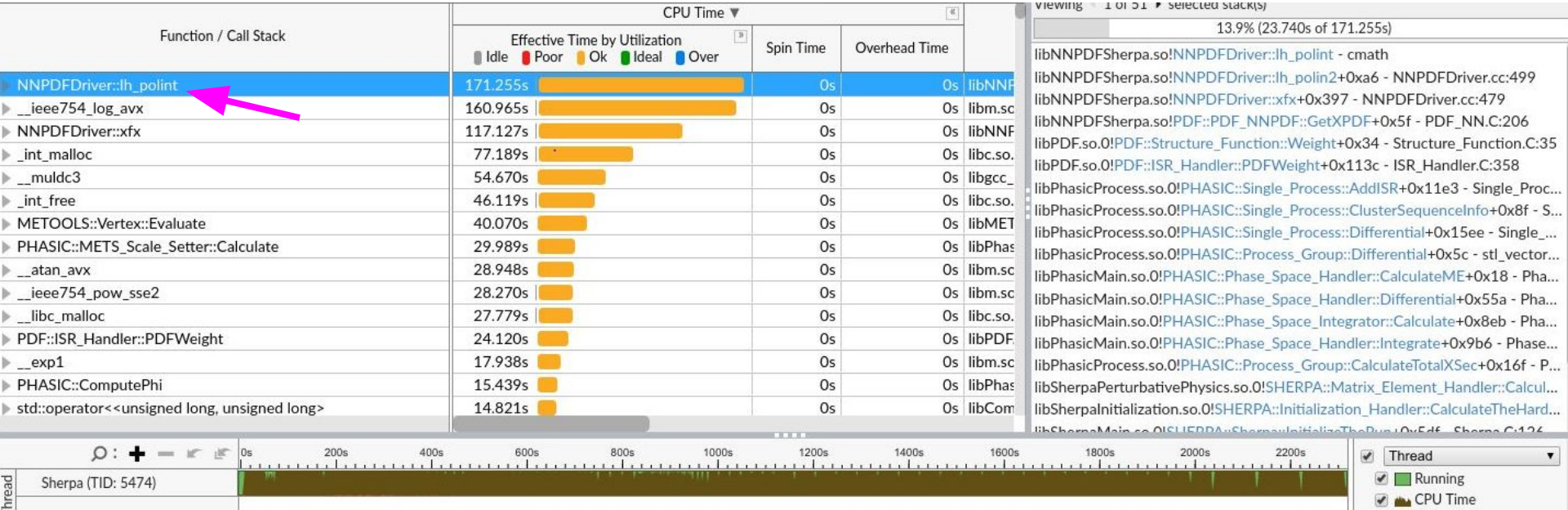


Sherpa Integrate



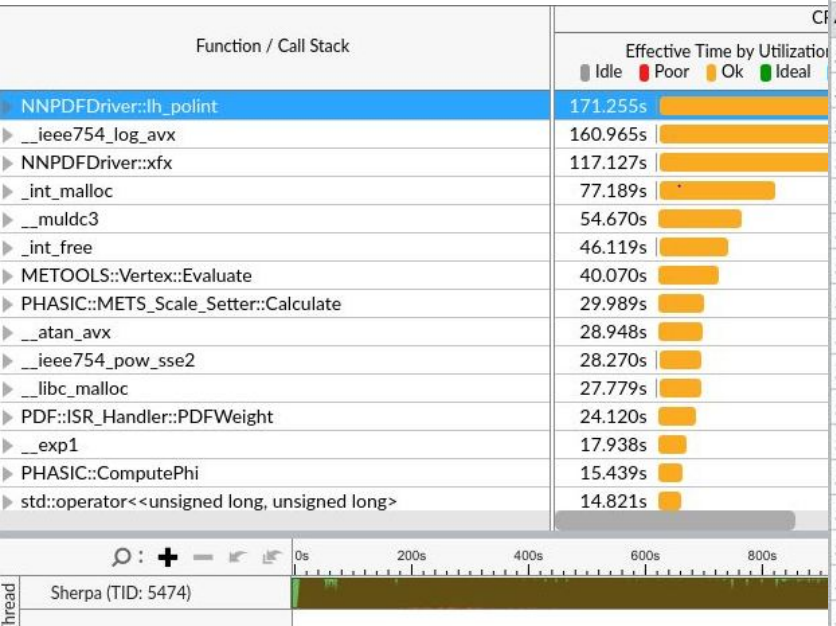
Sherpa Integrate Hotspots #1

- Possible hotspot in PDF evaludation
- **`NNPDFDriver::lh_polin2 <- NNPDFDriver::xfx <- PDF::PDF_NNPDF::GetXPDF`**



Sherpa Integrate Hotsp

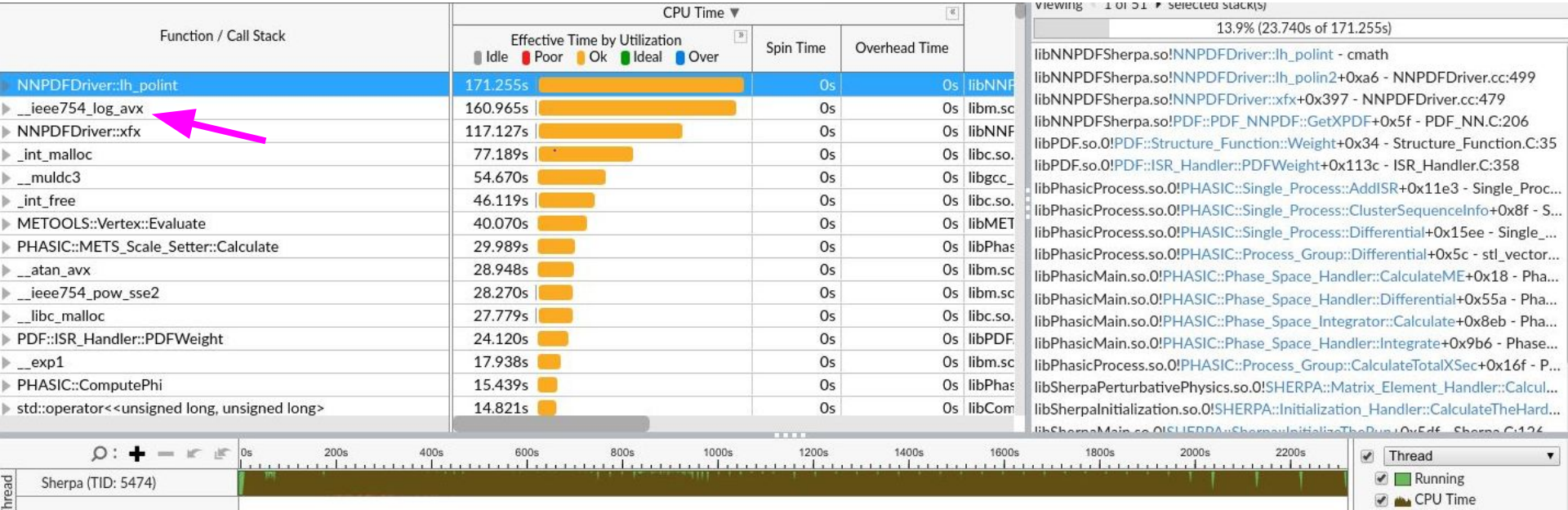
- Possible hotspot in PDF evalud
- **`NNPDFDriver::lh_polin2`**
  `PDF::PDF_NNPDF::GetXPDF`




```
486  void NNPDFDriver::lh_polin2(double x1a[], double x2a[],
487                  double ya[][fN],
488                  double x1, double x2,
489                  double& y, double& dy)
490  {
491    double yntmp[fN];
492    double ymtmp[fM];
493
494    for (int j = 0; j < fM; j++)
495      {
496        for(int k = 0; k < fN; k++)
497      yntmp[k] = ya[j][k];
498
499        lh_polint(x2a,yntmp,fN,x2,ymtmp[j],dy);
500      }
501    lh_polint(x1a,ymtmp,fM,x1,y,dy);
502  }
503
504  void NNPDFDriver::lh_polint(double xa[], double ya[], int n, double x,
505                  double& y, double& dy)
506  {
507    int ns = 0;
508    double dif = abs(x-xa[0]);
509    double c[fM > fN ? fM : fN];
510    double d[fM > fN ? fM : fN];
511
512    for (int i = 0; i < n; i++)
513      {
514        double dift = abs(x-xa[i]);
515        if (dift < dif)
516      {
```

Sherpa Integrate Hotspots #2

- Also in NNPDFDriver, 1% of CPU spent on **log** function (**`libm.so`**)



Sherpa Integrate Hotspots #3

- **`Malloc`** quite high, large number of calls from **`std::vector`**
- Identify vectors which should be calling **`reserve()`**, using **`emplace`**

```
105   struct Cluster_Sequence_Info {
106   public:
107     double m_pdfwgt, m_flux, m_ct;
108     std::vector<Cluster_Step> m_txfl;
109
110     Cluster_Sequence_Info(const double &w=1.0, const double &f=1.0,
111                           const double &c=0.0) :
112       m_pdfwgt(w), m_flux(f), m_ct(c) {}
113     inline void AddSplitting(const double& t,
114                              const double& xa, const double& xb,
115                              const int& fla, const int& flb)
116   >   { m_txfl.push_back(Cluster_Step(t,xa,xb,fla,flb)); }
```

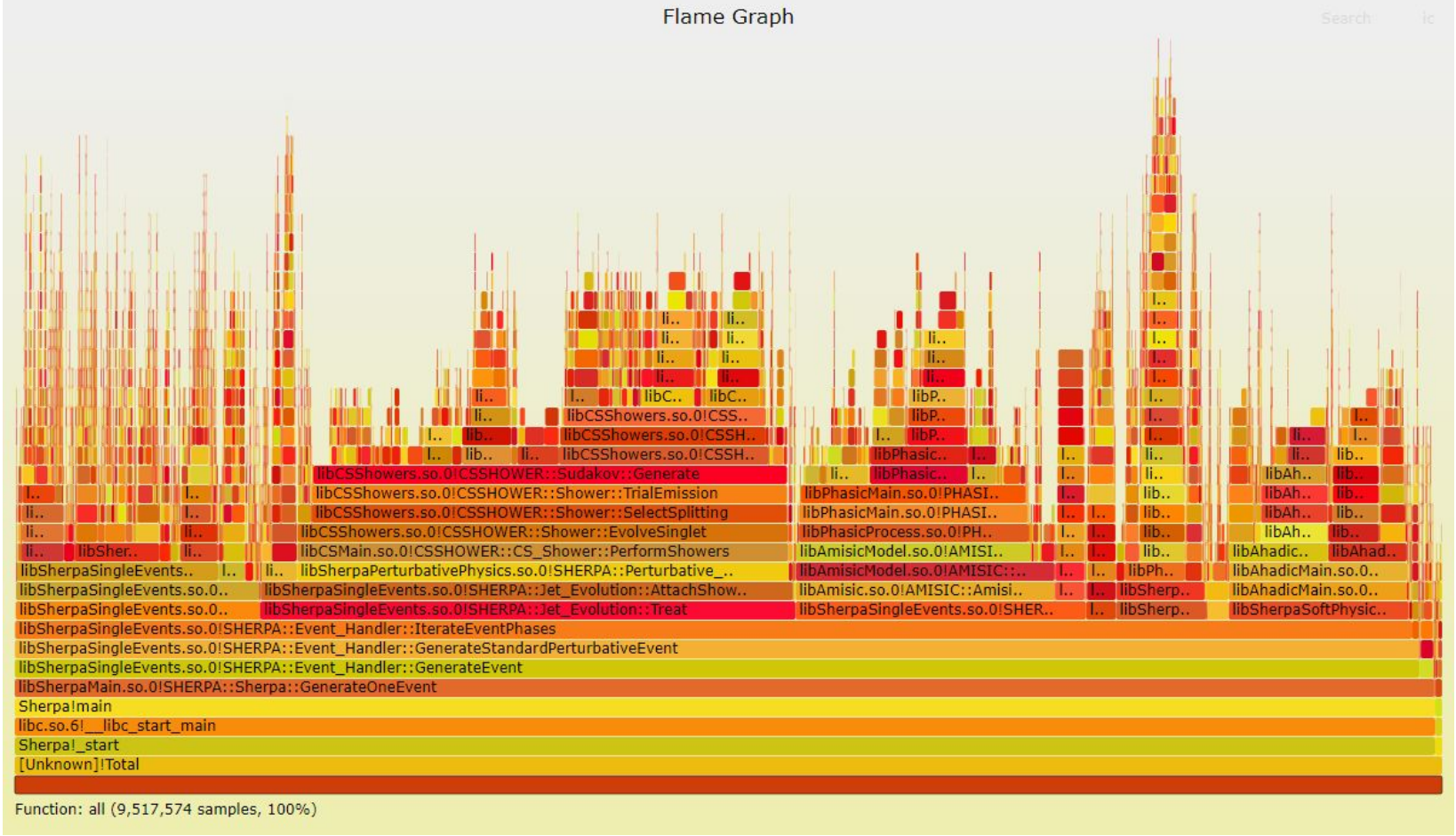
Function / Call Stack	CPU Time ▼ (Effective Time by Utilization: Idle, Poor, Ok, Ideal, Over)
NNPDFDriver::lh_polint	171.255s
__ieee754_log_avx	160.965s
NNPDFDriver::xfx	117.127s
_int_malloc	77.189s
__muldc3	54.670s
_int_free	46.119s
METOOLS::Vertex::Evaluate	40.070s
PHASIC::METS_Scale_Setter::Calculate	29.989s
__atan_avx	28.948s
__ieee754_pow_sse2	28.270s
__libc_malloc	27.779s
PDF::ISR_Handler::PDFWeight	24.120s
__exp1	17.938s
PHASIC::ComputePhi	15.439s
std::operator<<unsigned long, unsigned long>	14.821s

```
libc.so.6!_int_malloc - [unknown source file]
libc.so.6!__libc_malloc+0x4b - [unknown source file]
libstdc++.so.6!operator new+0x17 - [unknown source file]
libPhasicProcess.so.0!__gnu_cxx::new_allocator<ATOOLS::Cluster_Step>::all...
libPhasicProcess.so.0!std::allocator_traits<std::allocator<ATOOLS::Cluster_St...
libPhasicProcess.so.0!std::_Vector_base<ATOOLS::Cluster_Step, std::allocato...
libPhasicProcess.so.0!std::vector<ATOOLS::Cluster_Step, std::allocator<ATO...
libPhasicProcess.so.0!std::vector<ATOOLS::Cluster_Step, std::allocator<ATO...
libPhasicProcess.so.0!ATOOLS::Cluster_Sequence_Info::AddSplitting+0x8 - ...
libPhasicProcess.so.0!PHASIC::Single_Process::AddISR+0x808 - Single_Proce...
```

```
0s  200s  400s  600s  800s  1000s  1200s  1400s  1600s  1800s  2000s  2200s
Sherpa (TID: 5474)
```

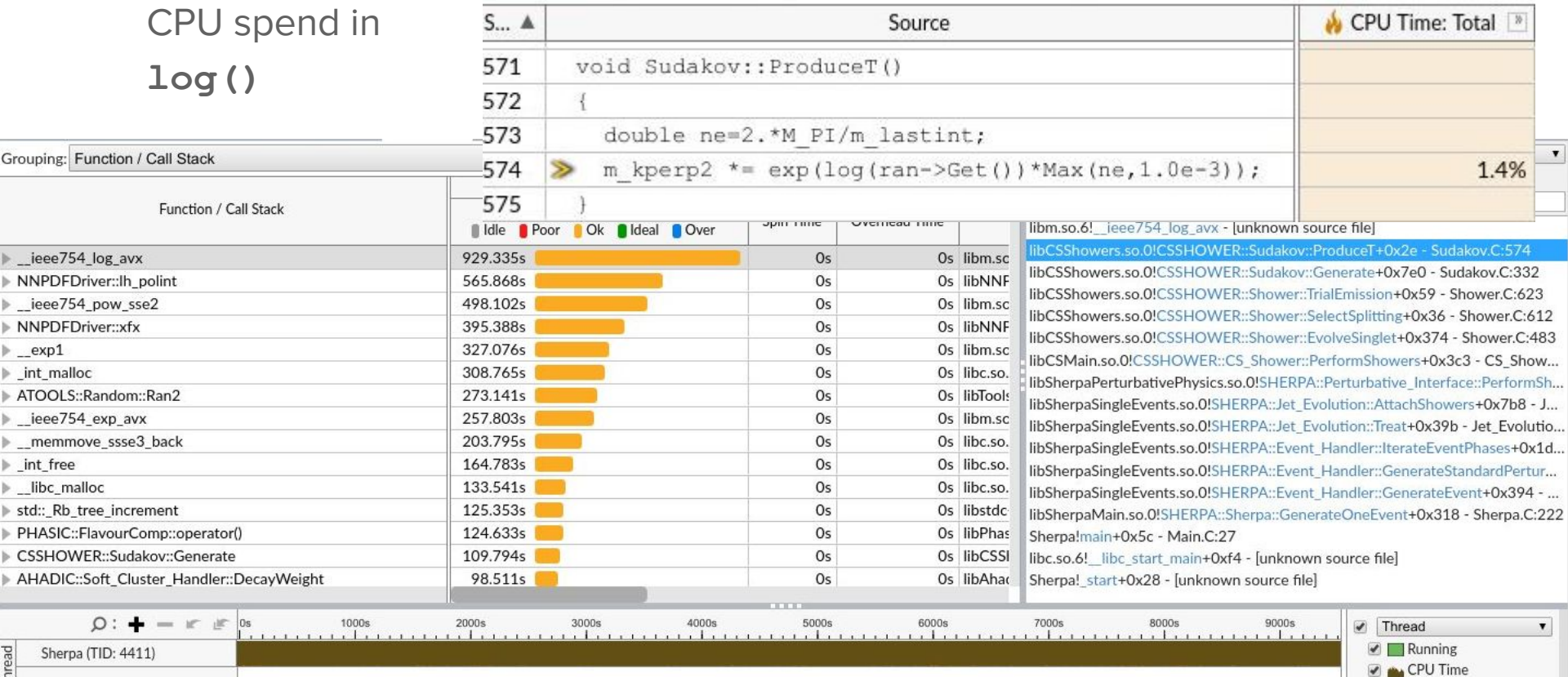
Thread ▾ — Running — CPU Time

Sherpa Evgen

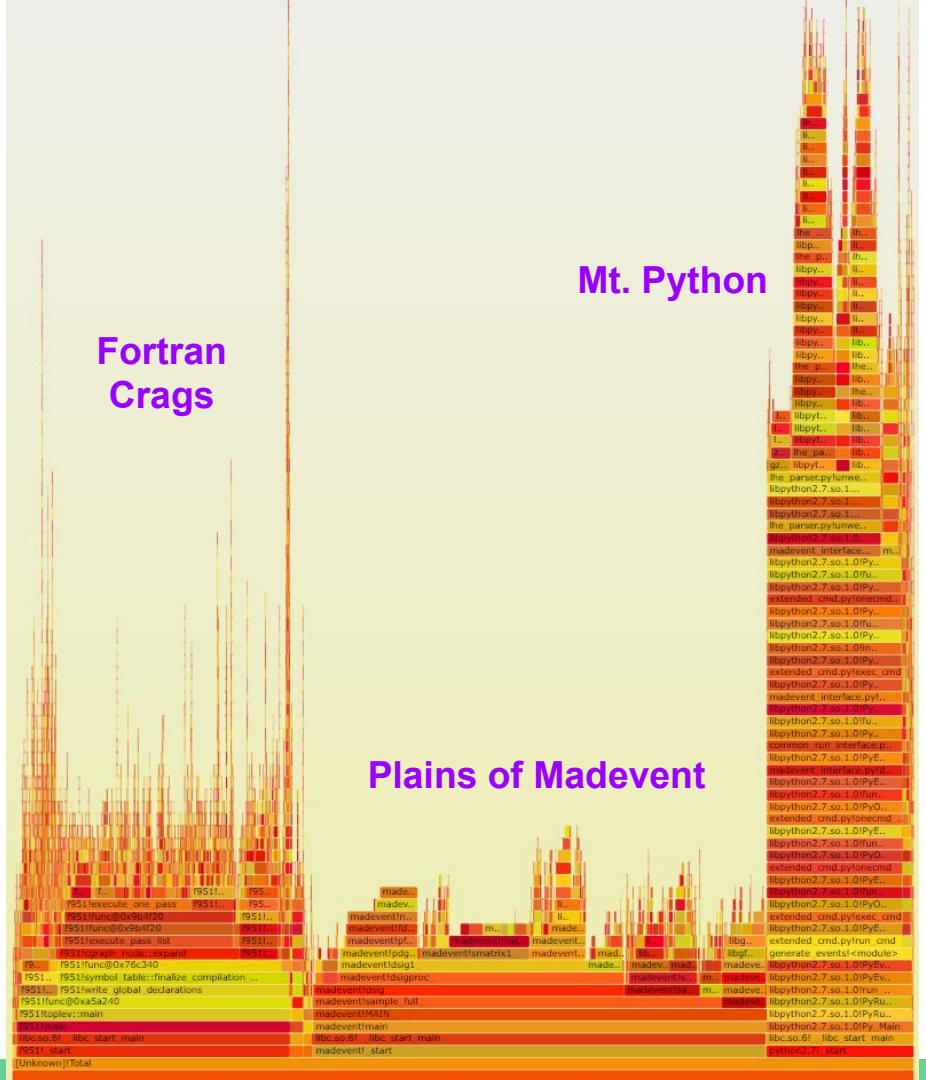


Sherpa Evgen Hotspots

- **OpenLoops** did **not** get picked up - *need to understand why…*
- Sherpa parts share similarities to Integration phase. Another area with large CPU spend in `log()`



Madgraph



Madgraph

- Quick tutorial job (10,000 events tt, matrix only)
- **`lh_polint`** showing up here too…
- MG is using multi-core

Elapsed Time: 54.982s

CPU Time: 33.390s
Total Thread Count: 907
Paused Time: 0s

Top Hotspots

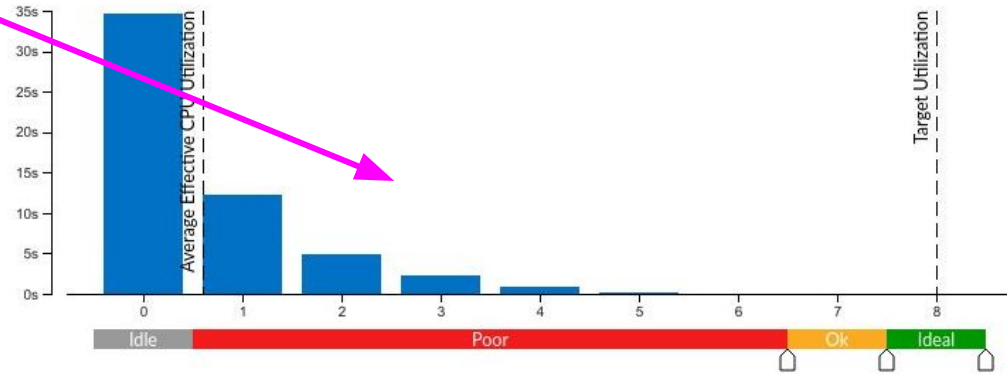
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in impr application performance.

Function	Module	CPU Time
lh_polint_	madevent	1.142s
nnevolvepdf_	madevent	0.831s
ffv1_2_	madevent	0.679s
longest_match	libz.so.1	0.611s
_int_malloc	libc.so.6	0.579s
[Others]		29.548s

*N/A is applied to non-summable metrics.

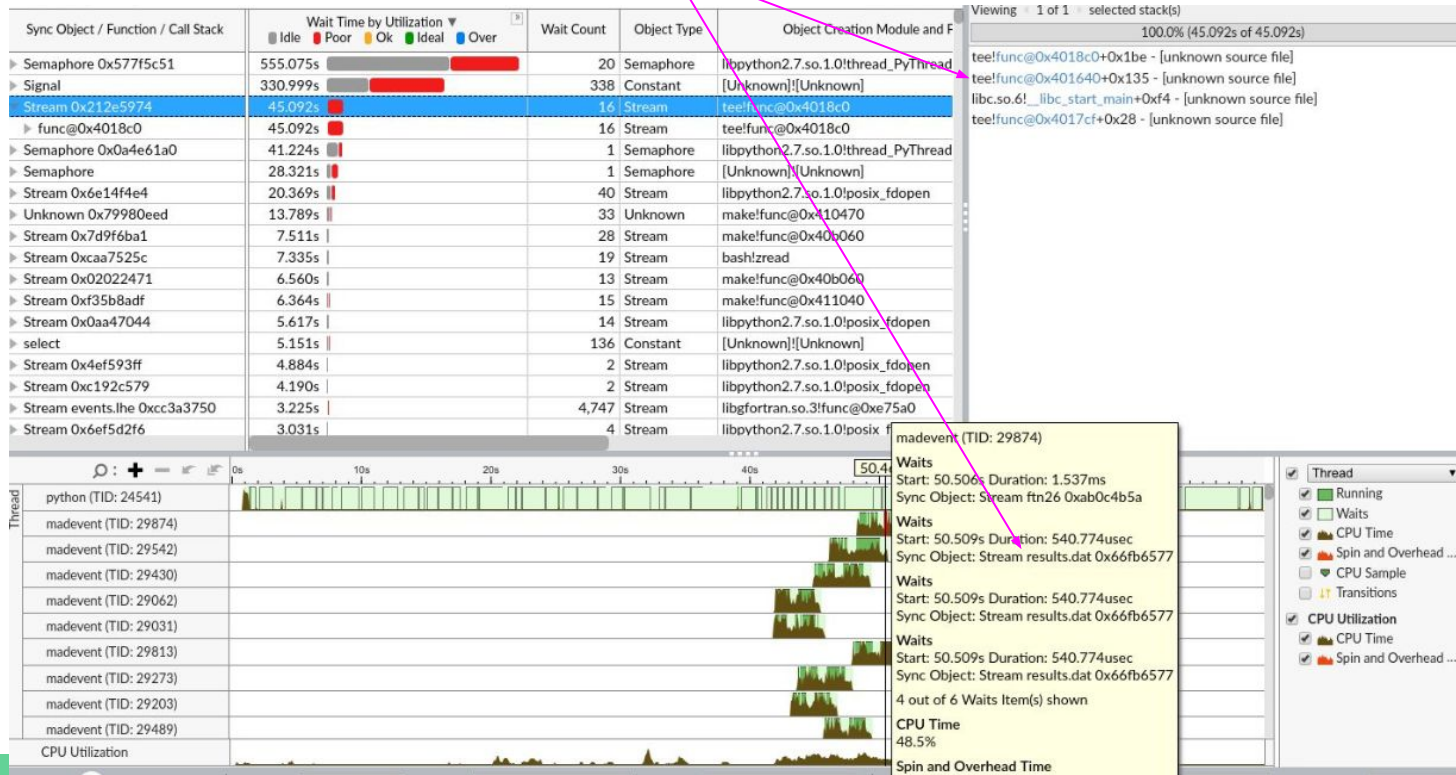
Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and



Madgraph Threading

- Should run more events, processes look too short-lived
- Blocking may relate at least in part to the I/O



Status Summary

- Started some “standard” profiling over Sherpa and Madgraph.
- Some potential easy fixes seen, but more in-depth dives into profile outputs are needed.
- Also started to look at the meta-level of ME code generation, post-compile optimisations - but no results here yet.