The CptnHook Profiler - A tool to investigate usage patterns of mathematical functions.

D. Piparo*, V. Innocente*

*** CERN, CH-1211 Geneva 23, Switzerland**

Abstract. Transcendental mathematical functions are one of the main hot-spots of scientific applications. The usage of highly optimised, general purpose mathematical libraries can mitigate this issue. On the other hand, a more comprehensive solution is represented by the replacement of the generic mathematical functions by specific implementations targeting particular sub-domains only.

CptnHook is a tool that helps achieving this goal allowing to monitor the input values of mathematical functions used in a given application, categorised according to the stacktraces leading to their invocations. In this contribution we describe the design of CptnHook, the data format of its profile and how it is possible to perform measurements without instrumenting the users code and imposing the need of recompilation.

We demonstrate that this approach scales on production workflows of LHC experiments and characterise a set of real life measurements, showing where opportunities for improvement lie and how the tool can be used for advanced debugging. We also illustrate how elegant summaries of the measurements can be produced and how ROOT based analysis of the profiles can be performed.

1. The ANR Metalibm Project

CptnHook has been developed within the framework of Metalibm [1]. This project is financed by the French National Research Agency (ANR) and is supported by a consortium of four partners: INRIA, UPMC, LIRMM and CERN.

Metalibm addresses the implementation of elementary mathematical functions such as exponential or trigonometric, but also other functions which are specific to certain applications or domains and cannot be found in existing present and future libraries. The standard mathematical library (libm [2]) is traditionally a rock-solid reference in scientific computing but offers a small set of elementary functions in a small set of precisions. In addition the performance of its implementations is not always ideal.

The runtime performance of mathematical functions is of critical importance, in particular in scientific and financial computing. In many contexts, these functions have always received special attention. For instance, processor manufacturers have teams of engineers who implement an optimized mathematical library version for each new processor released in order to take maximum advantage of the microarchitecture at disposal: e.g. registers, cache hierarchies, ports. This is a time-consuming and error-prone task.

A first objective of this project is therefore to automate the development of mathematical libraries, up to the point where libms are generated in a fraction of the time required by the

manual development procedure mentioned above. The performance and precision must also match the hand-coded ones not only when running on general purpose processors, e.g. for the server market segment, but for a wide range of targets including embedded processors, graphical processing units and even FPGA as well as digital circuits.

It is part of the program of work of Metalibm to provide the means for creating special versions of elementary functions, notably optimised for subdomains smaller than the full ones. A posterchild example is represented by the trigonometric functions for a restricted set of angles.

2. The Cost of Floating Point Operations

Floating point operations represent a considerable fraction of the computing time of scientific applications. High Energy Physics is not an exception in this respect: workflows are not exceptional for which 10% to 20% [3] of the time is spent in floating point operations, e.g. the applications performing maximum likelihood fits in presence of complex models and seizable datasets, detailed simulations of passage of particle through matter or simulation of many body systems such as particles in accelerators [4].

In general, the main portion of this time is accountable to mathematical functions: their cost in terms of runtime must be kept under control and minimised whenever possible if performance needs to be achieved.

3. A possible Solution

One possible way to speed up scientific application consists in the adoption of a highly optimised mathematical library such as VDT [3], AMD Libm [5] or Intel MKL [6]. On the other hand, this approach is less than optimal. A more comprehensive solution would be to identify on a case by case basis, e.g. for all sequences of function calls separately, a mathematical function implemented in terms of polyomials [7] tailored to provide just the right result's accuracy needed within the domain required for the calculation at hand.

Operations on the input values such as range reductions, e.g. in presence of trigonometrical functions, and validity checks are expensive calculations. In addition, guaranteeing accurate results over large intervals provides generality but penalizes performance of mathematical functions given the branching this kind of implementations often foresee.

The knowledge of the possible input values of mathematical functions can be immediately be translated in a performance improvement thanks to the optimisations relative to the aforementioned effects.

Unfortunately it is not trivial to determine the range of the input values of functions while implementing an algorithm. It is even harder to figure this out when dealing with large software systems like the software stacks of the LHC experiments.

A flexible, lightweight and easy to use profiler is needed to measure mathematical functions domains.

3.1. The CptnHook Profiler

Without the need of instrumenting nor recompiling the original code, the CptnHook profiler [1] allows to intercept and record for every call to any single argument mathematical function:

- the input value
- the stacktrace leading to the function call. This is necessary to isolate the contexts in which a particular function is invoked. It is indeed in general not enough in presence of a real life scientific application to record in a flat profile the input values of the mathematical functions.

[1] `https://github.com/dpiparo/cptnHook`

The hooking mechanism is based on the preload of the CptnHook shared library and not on code instrumentation. This preload happens transparently for the user (see 1). This library provides functions whose symbols have an identical signature to the one of the mathematical functions. Internally though, before redirecting to the actual mathematical function for the output value calculation, they store the input value as well as the back-trace which lead to the function call.

These features can be offered thanks to the preloading of a CptnHook component which "filters" all the calls to mathematical functions. Such a procedure implies a cost in terms of runtime overhead. It depends on the application and is quantified by a 2x-5x slowdown. The memory overhead is negligible. The data of the profile is stored in a columnwise compressed binary format for optimal usage of disk storage. The tool can be easily generalised: it is indeed straightforward to extend the hooking mechanism to apply to user defined functions with a `double(double)` or `float(float)` signature.

The invocation of CptnHook looks like the following command:

```
> cptnhook o myProfile  myProgram [arguments]
```
(1)

3.2. Analysis of the Results

Once the binary profile has been written, the `cptnhook-analyze` tool can be used to convert the binary profile into a web report or a ROOT [8] file.

The web report, see figure 1, provides high level information about the math function usage, such as:

- Minimum and maximum values of the arguments.
- Overall number of calls.
- Individual frames of the stacktraces.
- Grouping by stacktrace.

dpiparo.web.cern.ch

CptnHook Report gensimReport3/

Fcn Name	N Calls	Min	Max	Distinct Traces
exp	157879485	-1.00422817289e+27	50.7085064317	504
log	7189038	1.00208418e-292	3.46020761244e+11	4393
cos	5428826	-21.3653747467	6.23747672438	312
tanh	2907906	-42.0647244906	42.8540378648	14
acos	678211	-0.999986162326	1.0	23
atanf	600848	-90.3181304932	312.380096436	2
sin	256560	-3.14144142339	6.28012932983	19
asin	181246	-0.597844962924	1.0	11
tanf	138457	-1.55972445011	1.56335806847	2
acosf	135500	-0.999978899956	0.999981045723	2
tan	60056	-1.55569219857	3.09466767311	37
atan	31842	-95.4622003447	530285872.95	8
asinhf	24969	-204.965576172	204.965576172	10
cosh	5141	-9.22107623615	9.04837138739	4
cosf	2524	-0.261166542768	0.341942310333	25
sinh	494	-2.85403786485	25.273780312	4

Click stacktraces to inspect the stack traces.

Figure 1. Simulation of proton collisions in the CMS detector. Part of the full CptnHook webreport. A table summarising the name of the mathematical function, the number of times it was called, the minimum and maximum input value and the number of distinct stacktraces which led to the invocation of it.

CptnHook allows to obtain information about the single stack traces, figure 2, and the details of the single stack frames too, figure 3.

CptnHook Report log

N Calls	Min	Max	Trace ID
1641406	2.07315779511	73410.8041792	4134
1123015	0.000219112293296	520.133257527	4115
763666	7.15255737305e-07	0.999999701977	4090
475896	0.000800461976442	861.412920114	4098
397701	106.018962316	48499.2142766	4144

Figure 2. The list of stack traces that led to the calculation of a logarithm. The row relative to stack 4090 signals a stack trace which was identified 763666 times and for which the input value of the logarithm was between 7e-07 and about one. Cases like this one could benefit from a polynomial approximation.

CptnHook Report Stack 4090

```
cmsRun() [0x40b789]
G4VProcess::ResetNumberOfInteractionLengthLeft()
G4VDiscreteProcess::PostStepGetPhysicalInteractionLength(G4Track const&, double, G4ForceCondition*)
G4SteppingManager::DefinePhysicalStepLength()
G4SteppingManager::Stepping()
G4TrackingManager::ProcessOneTrack(G4Track*)
G4EventManager::DoProcessing(G4Event*)
RunManager::produce(edm::Event&, edm::EventSetup const&)
OscarProducer::produce(edm::Event&, edm::EventSetup const&)
edm::one::EDProducerBase::doEvent(edm::EventPrincipal&, edm::EventSetup const&, edm::ActivityRegistry*
```

Figure 3. The names of the individual stack frames which led to the call of the logarithm. With this information it's possible to understand in which context the replacement with a polynomial expansion could be put in place.

The profile can also be converted to ROOT format, respecting the original column-wise storage of the data adopting ROOT tree data structures. Selections can be then applied on individual argument values or stacktraces and ROOT data visualisation routines leveraged. See for example figure 4.

3.3. A Real Life Example

A production workflow of the CMS [9] experiment was analysed with CptnHook: generation and Geant4 [10] based simulation of proton collisions at a centre of mass of 13 TeV originating top quark pairs. On a regular server machine, the profiler worked correctly demonstrating the ability to deal with applications of the order of millions of code lines like the CMS software stack. A detailed analysis of the results obtained is beyond the scope of this paper, but a preliminary assessment could point out to potential portion of the simulation code where the usage of polynomial expansions might be profitable, notably in the initialisation phase.

4. Conclusions and Future Work

This work is the first step toward a free and open source tool which is able to generate the optimal function for each context allowing to study the input values of mathematical functions resolving the different stacktraces which led to their invocation.

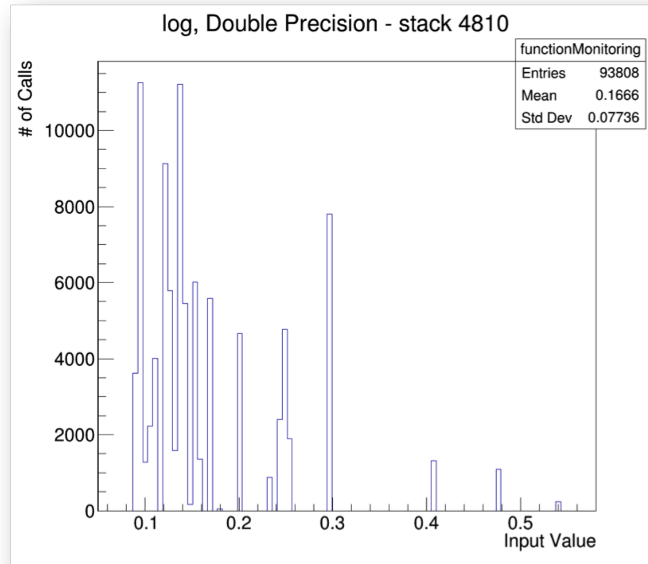

Figure 4. Simulation of proton collisions in the CMS detector. A ROOT histogram showing the input values of the log function for a particular stacktrace, hashed by the number 4810.

The hooking mechanism of CptnHook imposes a reasonable overhead on the profiled application and does not require recompilations nor code instrumentations. The profile written on disk during a measurement is in a compressed binary format and can be transformed either in a browsable web report summarising the principal informations about the calls carried out to mathematical functions or in a ROOT file for in-depth analysis based on non trivial cuts

Some of the main milestones of the future development program are a plug-in mechanism to add functions to be tracked, a richer look and feel of the default web reports e.g. visualising more statistics and control plots and the possibility to reduce the precision of the saved input values to further reduce the size of the reports on disk.

References

[1] Metalibm `http://www.metalibm.org`

[2] The GNU C library `https://www.gnu.org/software/libc`

[3] D Piparo et al.; 2014 J. Phys.: Conf. Ser. 513 052027, "Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions"

[4] A. Lasheen et al.; 2015 Proceedings of HB2014 East-Lansing, MI, USA, "Synchrotron Frequency Shift as a Probe of the CERN SPS Reactive Impedance"

[5] AMD LibM `http://developer.amd.com/tools-and-sdks/cpu-development/libm`

[6] Intel's MKL `http://software.intel.com/en-us/intel-mkl`

[7] S Chevillard et al.; 2010 Mathematical Software - ICMS "Sollya: an environment for the development of numerical codes."

[8] R. Brun et al. 1997 ROOT - An Object Oriented Data Analysis Framework *Nucl. Inst. Meth. In Phys.* **A 389** pp 81-86

[9] The CMS Experiment `http://cms.web.cern.ch`

[10] S Agostinelli et al.; 2003 Nucl. Instrum. Meth. A506 "GEANT4: A Simulation toolkit"