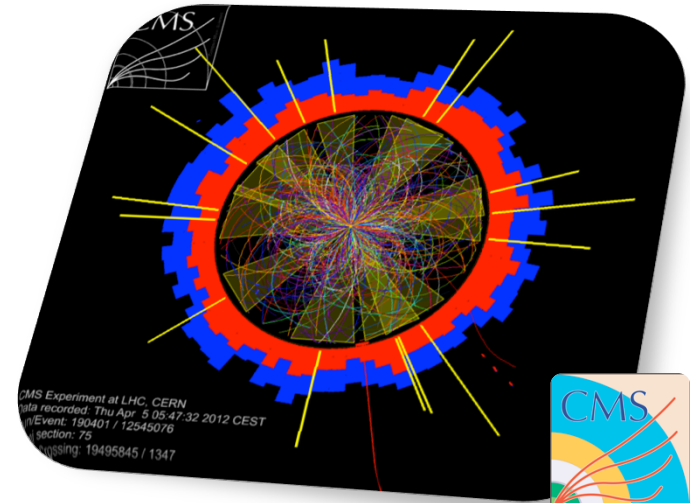
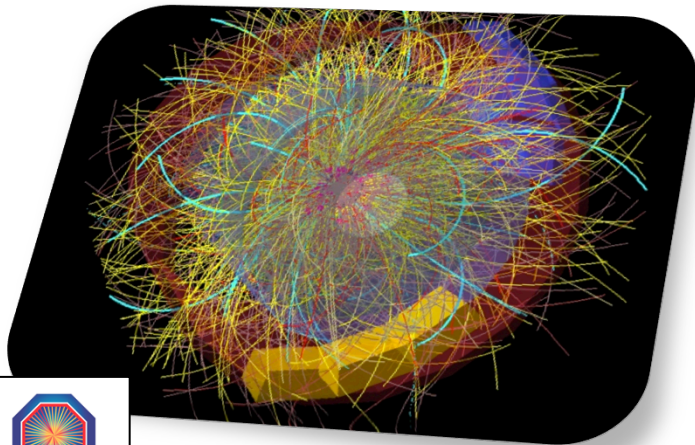


Speeding up HEP experiments' software with a library of fast and autovectorisable mathematical functions



D. Piparo, V. Innocente – PH-SFT

T. Hauth – PH-CMG

CHEP 2013



- Floating point calculations represent a significant portion of the runtime budget of HEP applications
- Sophisticated FORMulas TRANslated from literature into code)
 - **Mathematical functions appear often**
 - Their execution is expensive!

Target Accuracy – Operative Definition

Select the minimal accuracy in the intermediate steps which allows to obtain a correct final result.

Can the concept of target accuracy help us to improve the impact of mathematical functions' evaluation?

Pricetags of Mathematical Functions

Prices reported in percentage of the runtime of a full job

- CMS (TTBar events @ 8 TeV):
 - Reconstruction (~25 PU): **~9%**
 - Simulation: **~12%**
 - Simulation Initialisation: **~30%**
- Alice (pp, Geant4):
 - Simulation: **~15%**

Libm (glibc) implementation of these functions

Software stacks versions: beginning 2013



LHC Sim. Reminder 2012

Billions of Geant4 events

ATLAS: 2.1 (1.8 Fast-sim)

CMS: 4

LHCb: 1.2

ALICE: 1 (~30mins each!)

Libm: the default library

With some exceptions, the default mathematical library used for HEP calculations is **Libm** (glibc implementation)

Running on linux powered machines

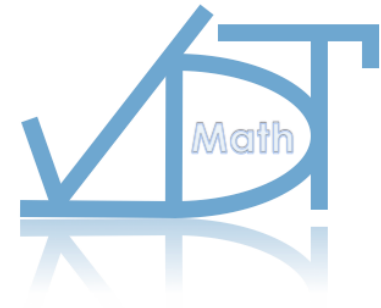
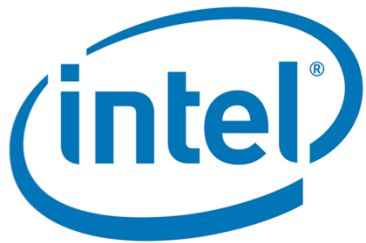


- **A rock-solid reference!**
- **Always focussed on accuracy rather than performance**
- **A single implementation dating 20 years ago**

A Selection of Alternatives

A plethora of different products are available, for example:

- Intel's SVML, IMF, MKL (commercial)
- AMD Libm (free, closed source)
- VDT (Vectorised maTh: free and open source)
 - **Our contribution**



Differences in the implementations but common underlying principle:

Trade off between accuracy and speed of execution



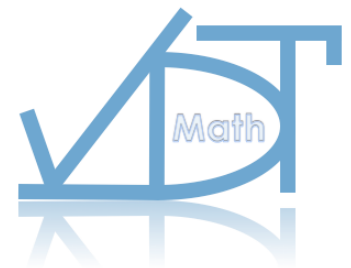
VDT

What is VDT?

- An **open source** math library library, LGPL3 licence
- Single/Double precision of (a)sin, (a)cos, sincos, (a)tan, atan(2), log, exp and 1/sqrt
- **Fast, approximate, inline**
- Symbols names are different from traditional ones: `vdt::fast_<name>`
 - Do not force drop-in replacement, allow full control
- **Autovectorisable** since gcc 4.7
 - **Array signatures** available: calculate on multiple elements conveniently
 - Can be inserted in **autovectorised loops** (inline!)
- Inspired by the good old Cephes (and Quake III videogame)
- **Standard C code only** is used (no intrinsics): **portability guaranteed**
 - ARM, x86, GPGPUs, Xeon Phi, <future microarchitecture>



<https://svnweb.cern.ch/trac/vdt>



Pade' Approximants

- VDT (and Cephes) double precision functions: Pade' Approximants
- Single Precision: polynomials

The “best” **approximation of a function by a rational function** of a given order → Often better approximation than a truncated Taylor series

Padé approximant of $f(x)$ of order $[m/n]$ is the function

$$R(x) = \frac{\sum_{j=0}^m a_j x^j}{1 + \sum_{k=1}^n b_k x^k} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m}{1 + b_1 x + b_2 x^2 + \cdots + b_n x^n}$$

which agrees to the highest possible order to $f(x)$

$$\begin{aligned} f(0) &= R(0) \\ f'(0) &= R'(0) \\ f''(0) &= R''(0) \\ &\vdots \\ f^{(m+n)}(0) &= R^{(m+n)}(0) \end{aligned}$$



Quake II Fast isqrt

- Only exception to previous slide
- Light effects (e.g. reflections): **3-vector normalization**
- Important piece : **“magic constant” for a first rough value**, then improved with Newton’s method iterations

```
/// Sqrt implementation from Quake3
inline float fast_isqrtf_general(float x, const uint32_t ISQRT_ITERATIONS) {

    const float threehalfs = 1.5f;
    const float x2 = x * 0.5f;
    float y = x;
    {
        uint32_t i = details::sp2uint32(y);
        i = 0x5f3759df - ( i >> 1 );
        y = details::uint322sp(i);
    } !!
    for (uint32_t j=0;j<ISQRT_ITERATIONS;++j)
        y *= ( threehalfs - ( x2 * y * y ) );

    return y;
}
```

Adjust accuracy with a parameter (impossible in libm)

Speed: VDT Vs Libm

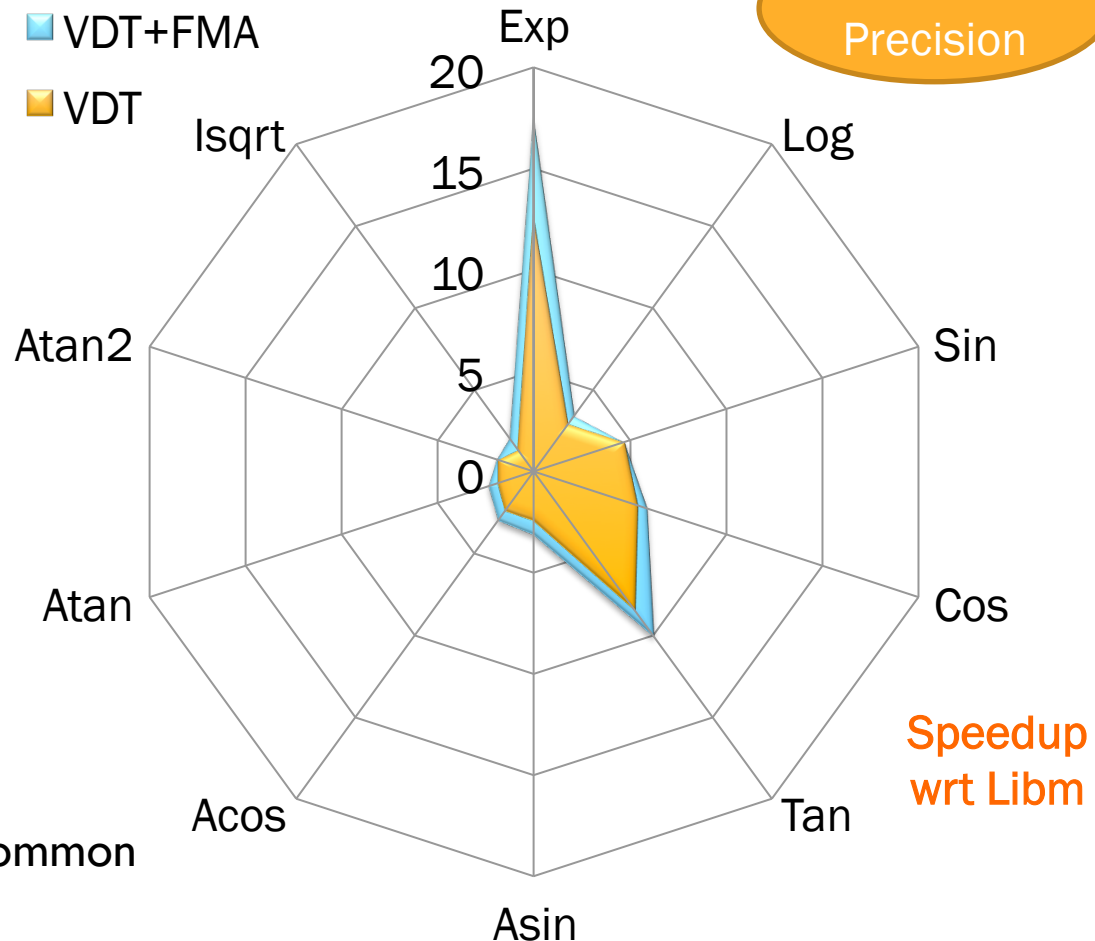
Fnc.	Libm	VDT	VDT-FMA
Exp	102	8	5.8
Log	33.3	11.5	9.8
Sin	77.8	16.5	16.5
Cos	77.6	14.4	13.2
Tan	89.7	10.6	8.9
Asin	21.3	8.9	6.9
Acos	21.6	9.1	7.3
Atan	15.6	8.4	6.7
Atan2	36.4	19.9	18.9
Isqrt	5.7	4.3	2.8

Time in **ns** per value calculated

- Operative input range: [-5k, 5k]
- **Speedup factors of >5** not uncommon
- **Effect of FMA clearly visible**
 - **A waste not to profit from it!**
- Even more to gain when vectorising

■ VDT+FMA

■ VDT



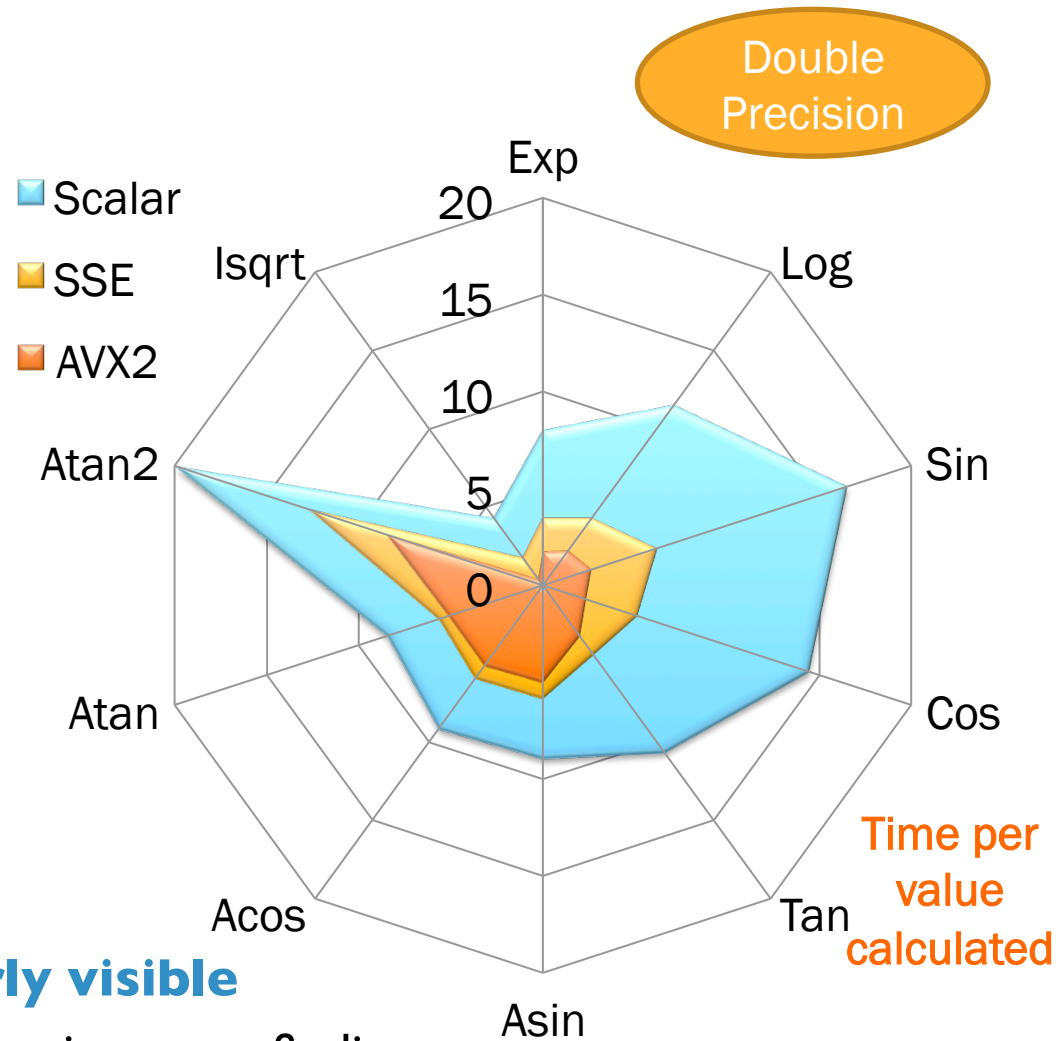
Testbed:

SLC6-GCC48, i7-4770K at 3.50GHz Haswell
glibc 2.12-1.107.el6_4.4 and VDT v0.3.6

Speed: VDT Vectorisation

Fnc.	Scalar	SSE	AVX2
Exp	8	3.5	1.7
Log	11.5	4.3	2.2
Sin	16.5	6.2	2.6
Cos	14.4	5.1	2.3
Tan	10.6	4.4	3.2
Asin	8.9	5.8	5
Acos	9.1	5.9	5.1
Atan	8.4	5.6	5.1
Atan2	19.9	12.7	8.4
Isqrt	4.3	1.8	0.4

Time in **ns** per value calculated

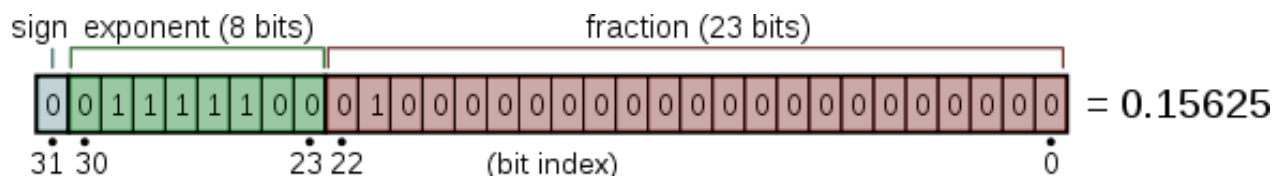


- **Effect of vectorisation clearly visible**
- CMS uses such vectorised signatures in vertex finding

Accuracy: An Example

- Accuracy was measured comparing the results of **Libm and VDT bit by bit with the same input**
- **Differences quoted in terms of most significant different bit**
- In the end they are just 32 (64) bits which are properly interpreted!

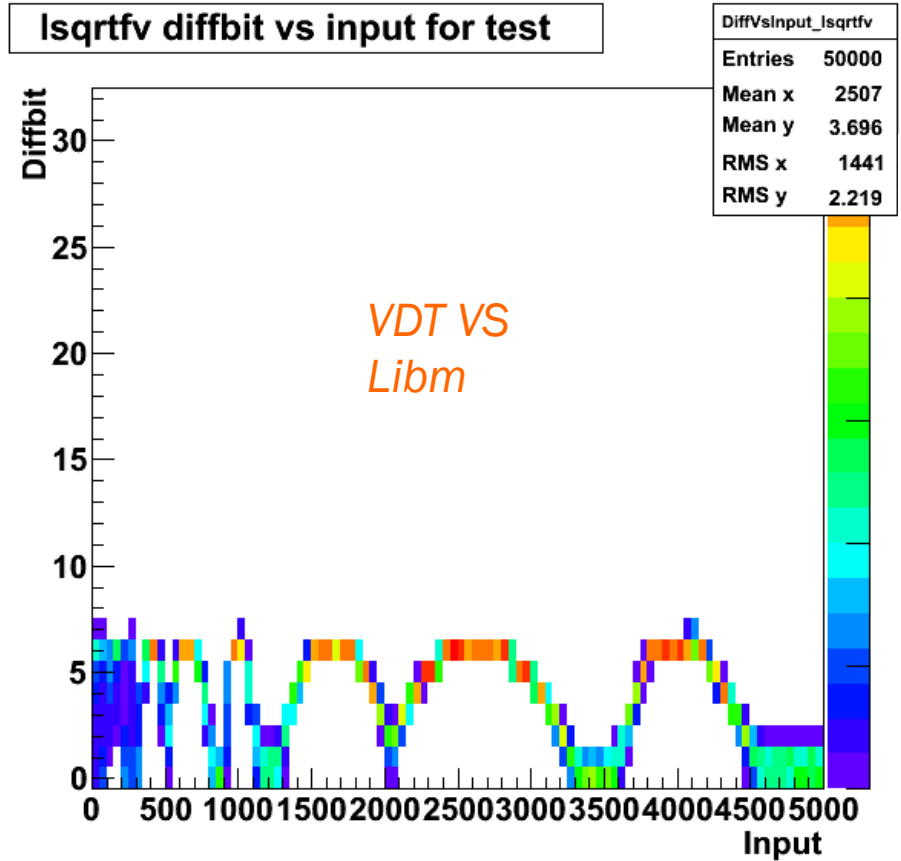
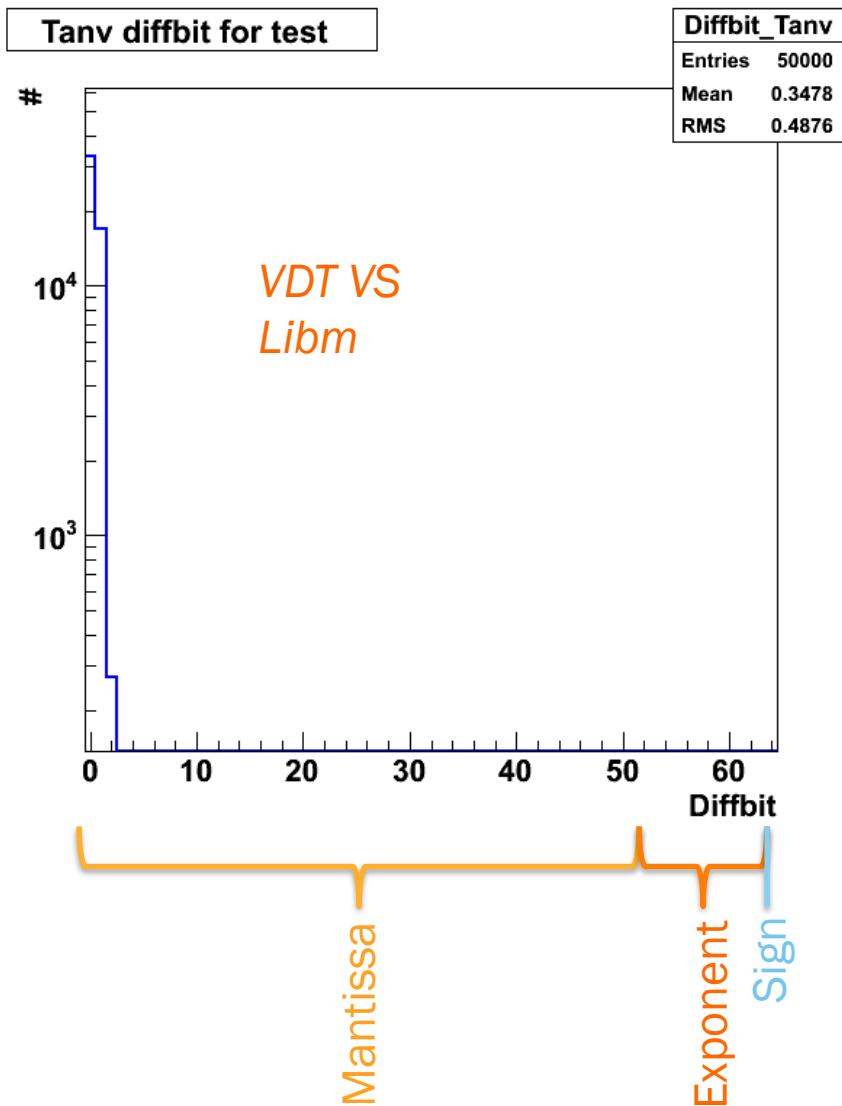
A single precision floating point number



Double Precision	MAX VDT	AVG VDT
Exp	2	0.14
Log	2	0.42
Sin	2	0.25
Cos	2	0.25
Tan	2	0.35
Asin	2	0.32
Acos	8	0.39
Atan	1	0.33
Atan2	2	0.27
Isqrt	2	0.45

Approximate results, but ok for a wide range of applications

Accuracy: A Visualisation



Well known behaviour of the
“Quake III” inverse square root



Examples from the LHC Experiments

Examples from the experiments

Methodology:

- Replace calls to Libm functions with the VDT ones: *LD_PRELOAD*
- **No hotspots but an overall replacement**

Caveats:

- **Not the best way to proceed**: no case by case control of accuracy... **But the least intrusive technique!**
- Code performance improvements cited: **conservative – no inlining with preload!**
- Physics performance: **conservative - maximum variation obtainable (all calls replaced!)**

The Alice Case



Pb+Pb @ \sqrt{s} = 2.76 ATeV

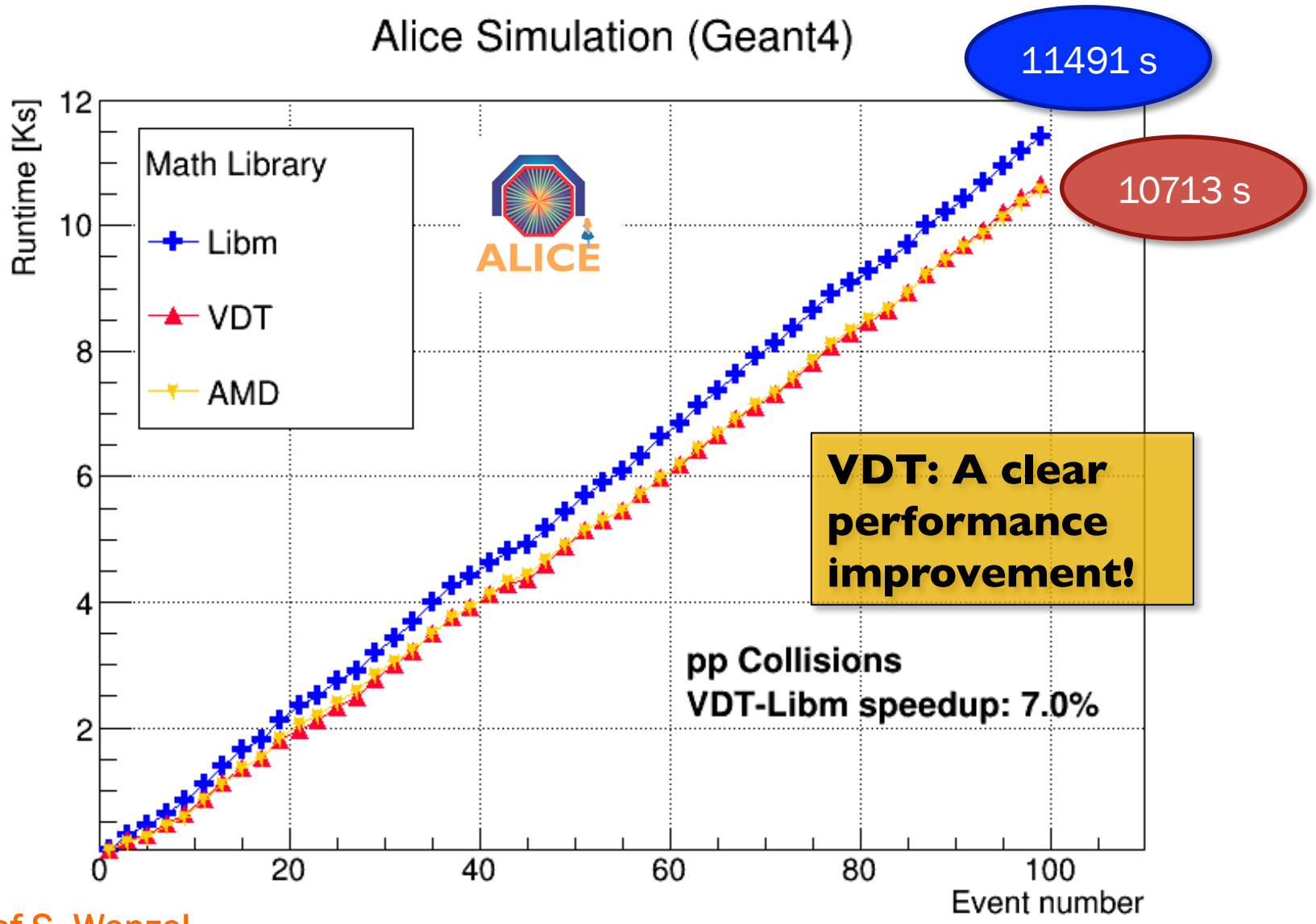
2010-11-08 11:30:46

Fill : 1482

Run : 137124

Event : 0x00000000D3BBE693

Alice Simulation: Switching to VDT



Courtesy of S. Wenzel

Alice Sim: Preliminary Validation

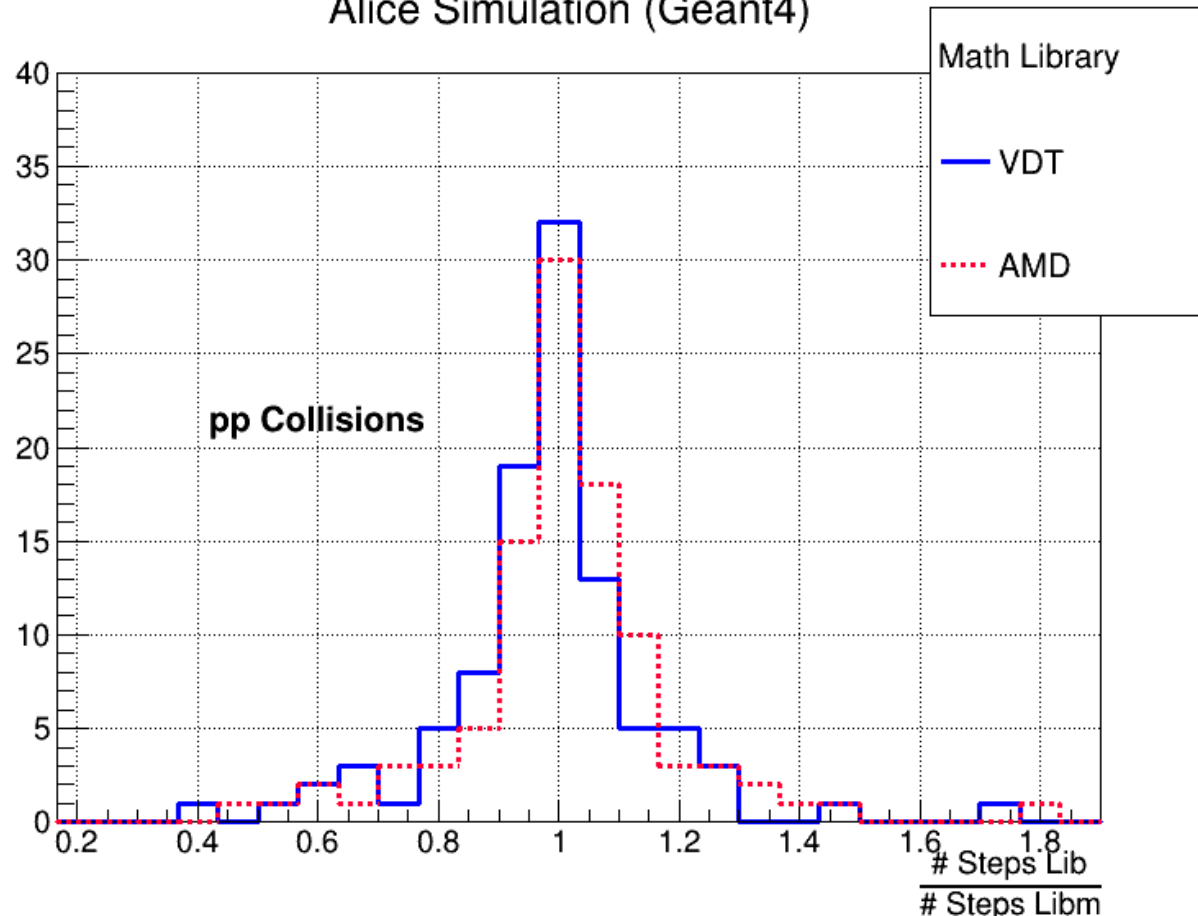
Output of simulation validated in terms of number “simulation steps”

- **Similar simulation steps’ number: indication of compatible results**
- Number of steps for the AMD Libm and VDT cases compared to Libm:
 - **Nicely distributed around 1**

Some validation work needed for final sign-off, But results already very encouraging!

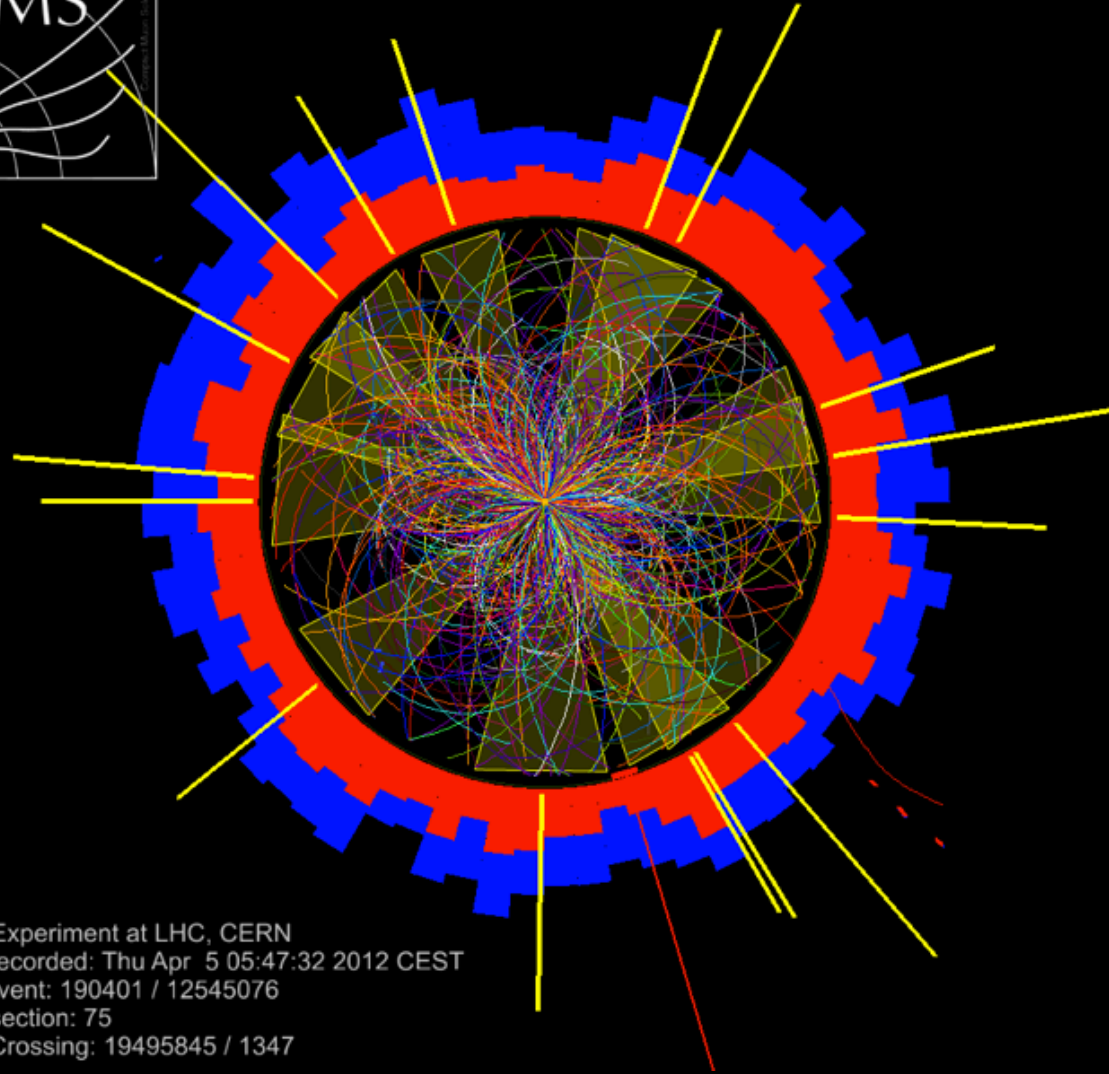


Alice Simulation (Geant4)



Courtesy of S. Wenzel

The CMS Case



CMS Experiment at LHC, CERN
Data recorded: Thu Apr 5 05:47:32 2012 CEST
Run/Event: 190401 / 12545076
Lumi section: 75
Orbit/Crossing: 19495845 / 1347

CMS Simulation: Switching to VDT

Top-antitop events @ 8 TeV: Inclusive channel, **test numerous code paths**

Two workflows benchmarked:

- **50 events: mimic a long running job, e.g. Grid (~5ks)**
- **1 event: de facto measure startup overhead**

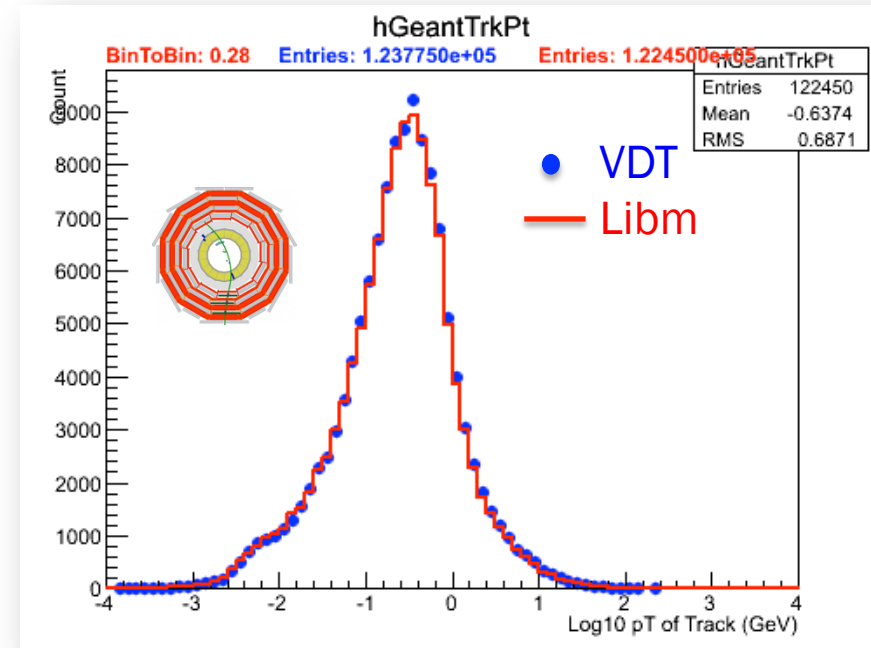
Result:

- **6.6% speedup achieved - 50 Events**
- **25% speedup achieved - 1 Event**

Validation:

- **Good compatibility** of results
 - Use standard DQM histograms
- Changes expected and found
 - Different accuracy of the functions!
 - Experts' validation must sign-off!

Performance improvement with compatible results

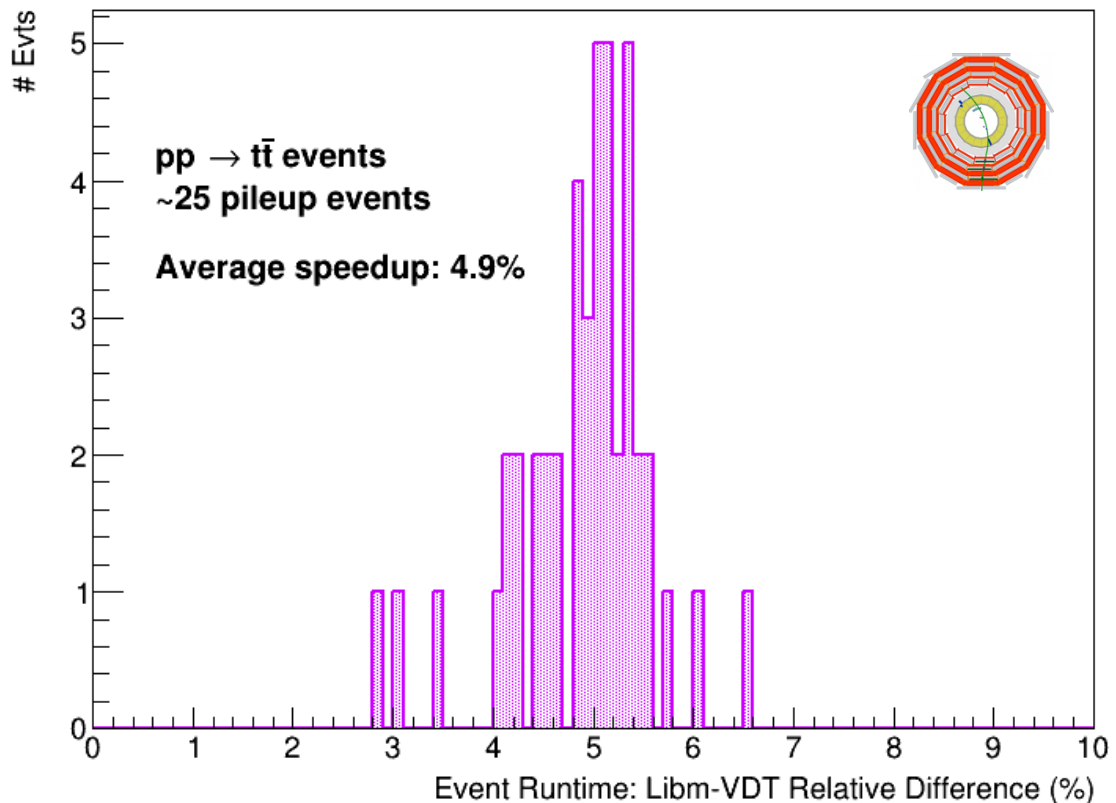


CMS Reco: Switching to VDT

H,A \rightarrow $\tau\tau$ \rightarrow two jets + X, 60 fb⁻¹

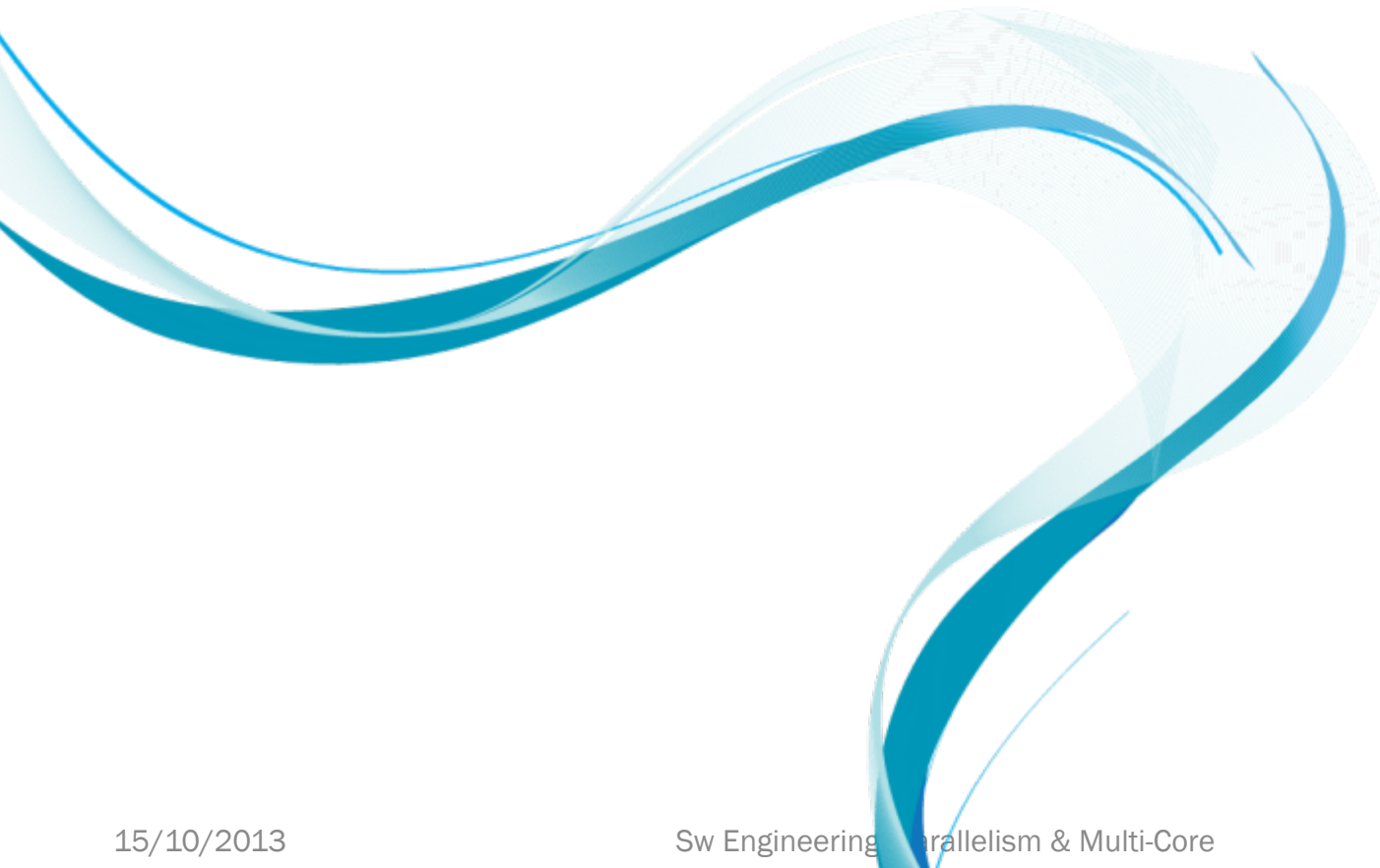
- Reconstruct Monte Carlo top-antitop events + ~25 pileup collisions
- **4.9 % speedup in event loop (w/o initialisation)**
 - CMS uses VDT and other approximations, no improvements achieved there
- **Good agreement of physics performance**
 - 120k plots compared, marginal differences

VDT-Libm Speedup: CMS Reconstruction



Interesting opportunities for fast mathematical functions even with reduced accuracy!

Is This the End?



- Integration of **VDT in ROOT** ongoing: interplay with Stats/Math, ...
- Improve numerical accuracy (e.g. recalculation of some constants)
 - Add more approximate but faster versions?
- Add more functions: hyperbolic, statistical distributions, ...

Beyond VDT

The ultimate mathematical library:

- **A high performance “metalibm”**
- Given a range and an accuracy: automatically obtain functions’ implementation

High quality polynomial approximations:

- Credible alternative to several of the FORmulas TRANslated from literature
- Replacing full formulas: faster and gives better control than replacing just the math functions in it - **tools are available: Sollya, Maxima, ...**

- Math functions: substantial portion of HEP applications' runtime
- **Libm: great reference, probably too expensive:** no optimisation for newer architectures
- Different alternatives to libm available
 - **We developed VDT:** optimised, vectorisable, portable and open source
 - Speedup **factors of ~10** not rare
 - Approximate, but up to a very few bits
 - Compiler alone responsible to generate machine instructions for target platform: **scalable on future and present architectures**
- Potential gains attractive in typical LHC workflows (5-7%)
- Physics outputs: encouraging signs of **good compatibility**



VDT Array and Scalar Signatures

VDT provides “array” and “scalar” signatures

Scalar signature: `T(T)`

- Example: `double y = vdt::fast_exp(x);`

Array signature: `void(const unsigned int,T*,T*)`

- Example: `vdt::fast_expv(11, input_array, output_array);`

Array signatures trivially autogenerated: script steered by CMake

```
void fast_expv(const unsigned int n, float* in, float* out{
    for (unsigned int i=0;i<n;++i)
        out[i] = fast_exp(in[i]);}
```

All the difficulties dropped on the compiler

Similar generator script is in place to autogenerated signatures to allow library preload if requested.