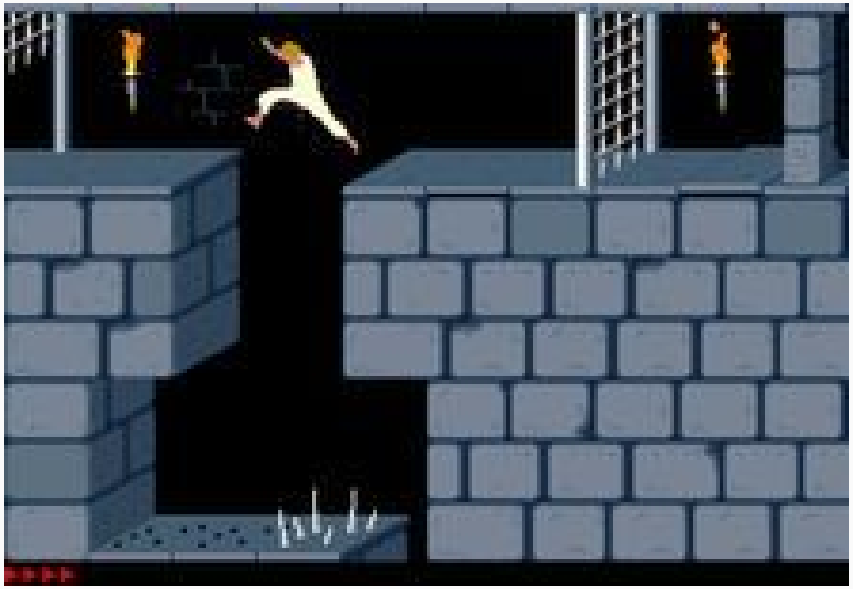
# Custom-Precision Mathematical Library Explorations for Code Profiling and Optimization

David Defour, Pablo de Oliveira Castro, **Matei Iștoan**, Eric Petit

Université de Versailles – Li-PaRAD, Exascale Computing Research Lab, Intel

# The Trick Is to Avoid the Pitfalls, Seize the Opportunities, and Get Back Home by Six O'clock

*(Woody Allen)*



**Floating-Point** numbers are approximations of **real numbers**.

- carefully designed, highly engineered, empirically proved trade-off

**Not** all computations need to **fully utilize** the underlying **hardware**.

- HPC conveniently focusing on using double precision

**Precision** vs. Accuracy.

- accuracy: how close am I to the true mathematical result
- precision: how many significant digits can I compute on this format

# Performance vs. Accuracy When Using the Math Library

General consensus: ↑ more accuracy = ↓ less performance

What happens when accuracy decreases?

- ↓ less energy
- ↓ less latency
- ↑ more throughput*

| Performance/ Max. Error | High Accuracy | Low Accuracy | Enhanced Performance |
|---|---|---|---|
| Expected performance | Default | Better Performance | Best Performance |
| Maximum accuracy error | 1ulp | 4ulp | The lower half of the mantissa may be incorrect |

**Intel MKL Vector Mathematics**

E.g.: Intel **M**ath **K**ernel **L**ibrary (**MKL**) 2020 **V**ector **M**athematics (**VM**)

| | High Accuracy | Low Accuracy | Enhanced Performance |
|---|---|---|---|
| Performance | 9.15 cycles | 6.84 cycles | 5.8 cycles |
| Accuracy | 0.75 ulp | 1.37 ulp | 5590000.0 ulp |

**erfc() in binary64**

| | High Accuracy | Low Accuracy | Enhanced Performance |
|---|---|---|---|
| Performance | 4.47 cycles | 2.96 cycles | 2.76 cycles |
| Accuracy | 0.51 ulp | 2.18 ulp | 26200000.0 ulp |

**exp() in binary64**

# Mathematical Library: Current State of Affairs

**Floating-Point formats** supported in the math library:

? 16-bit (*maybe*)
\+ 32-bit
\+ 64-bit
* 128-bit (*non-standard*)
– multi-precision (*very high cost, limited support*)

**Accuracy of computations** in the math library:

- usually specified in **ulp** (**u**nit in the **l**ast **p**lace)
- **ulp(**$x$**)**is the difference between the two finite floating-point numbers nearest to $x$, even if $x$ is one of them
- **ideally**, accuracy $<$ **0.5ulp**, (correct rounding)
- **acceptable**, accuracy $\leq$ **1ulp**, (faithful rounding)

# Meeting the Application's Needs

**Different format**?

- binary16
- bfloat
- 80-bit floating-point (extended precision)

**Different accuracy**?

- default implementation
  - accuracy *mostly* specified in the documentation
- fast implementation: **-ffast-math**


  - *how accurate*? *same accuracy* for all functions?
  - . . . and more importantly: **what does it break**?
  - advantages: *faster*. but how fast? and is it worth it?

# (I Can't Get No) Satisfaction

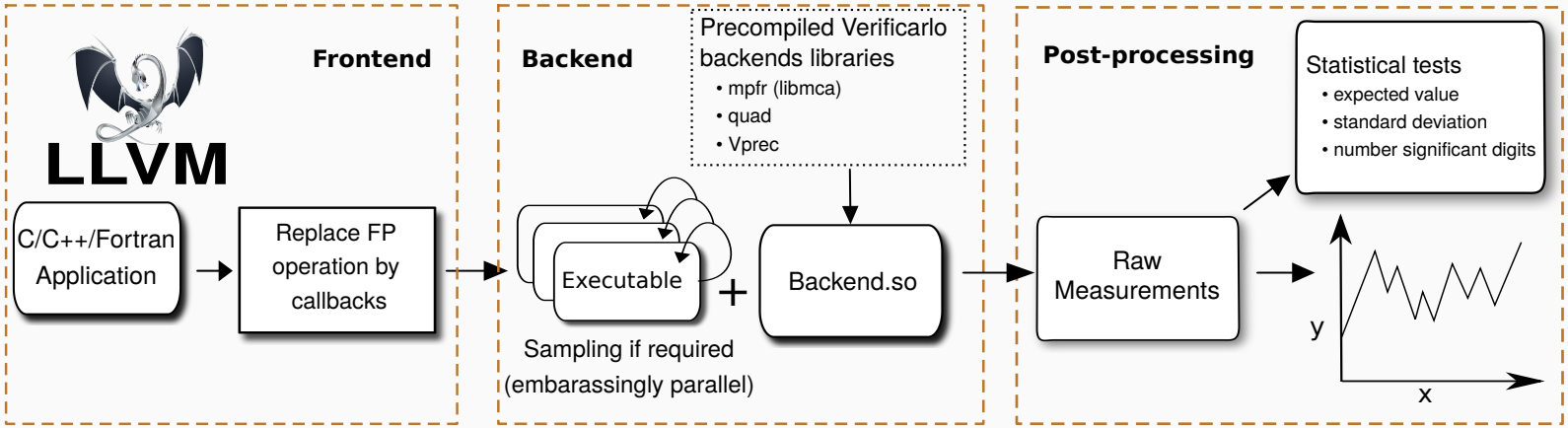

**What if you're still not satisfied?**

# Verificarlo: a Framework for Floating Point Verification, Debugging and Optimization


- **Objectives:**
  - Numerical stability, verification, debugging
  - Numerical portability across HW and SW
  - Performance optimization, mixed precision
- `github.com/verificarlo/verificarlo`
- [DdOCP16, CdOCP+18, SdOCF+18, CPdOC+19]

# Verificarlo: Workflow

- Verificarlo is based on the **LLVM compiler**
- **instrumentation/emulation** occurs **just before code generation**
- Verificarlo analyzes the code after optimization
- 2 main **backends**: MCA and Vprec



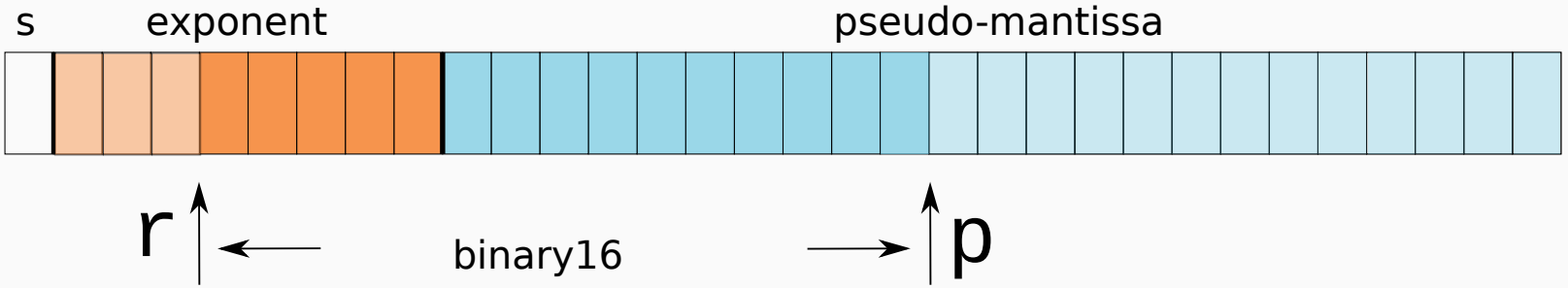
# Verificarlo: VPREC backend for mixed precision exploration

**Emulation** [CPdOC+19]:

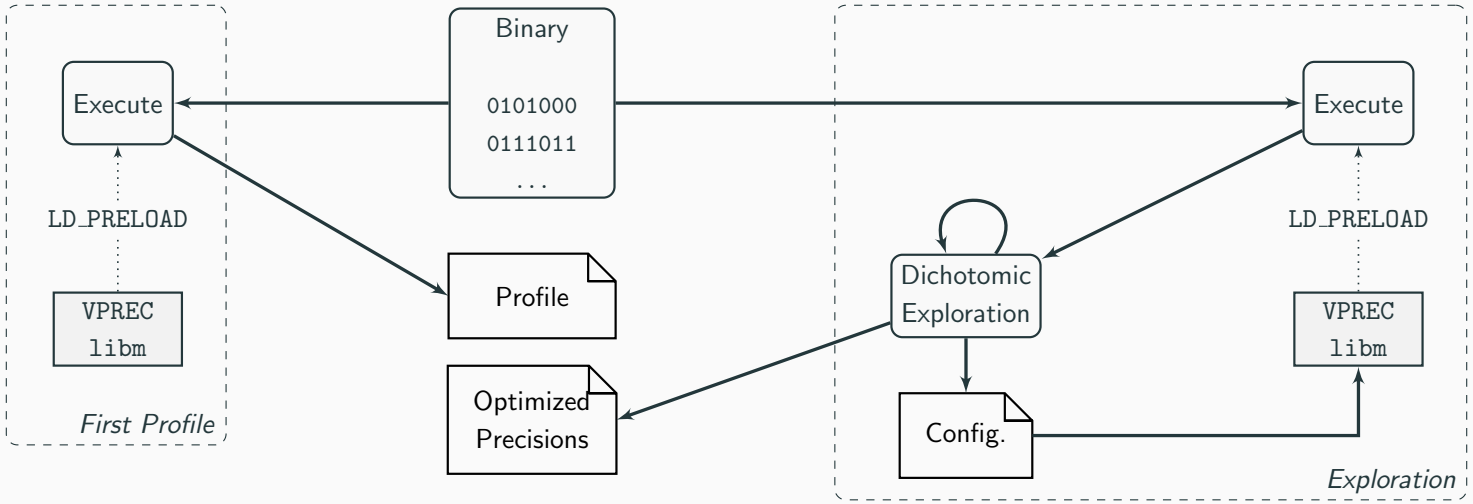
- any range and precision fitting in the original type
- handle **denormals**, **special values**
- implement **faithful rounding**
- only basic instructions: $+$, $-$, $*$, $/$

**Exploration:**

- **heuristic algorithm** to **explore lower precision implementation** of an algorithm over time
- complementary to other approaches like *delta debug* and *automatic differentiation*

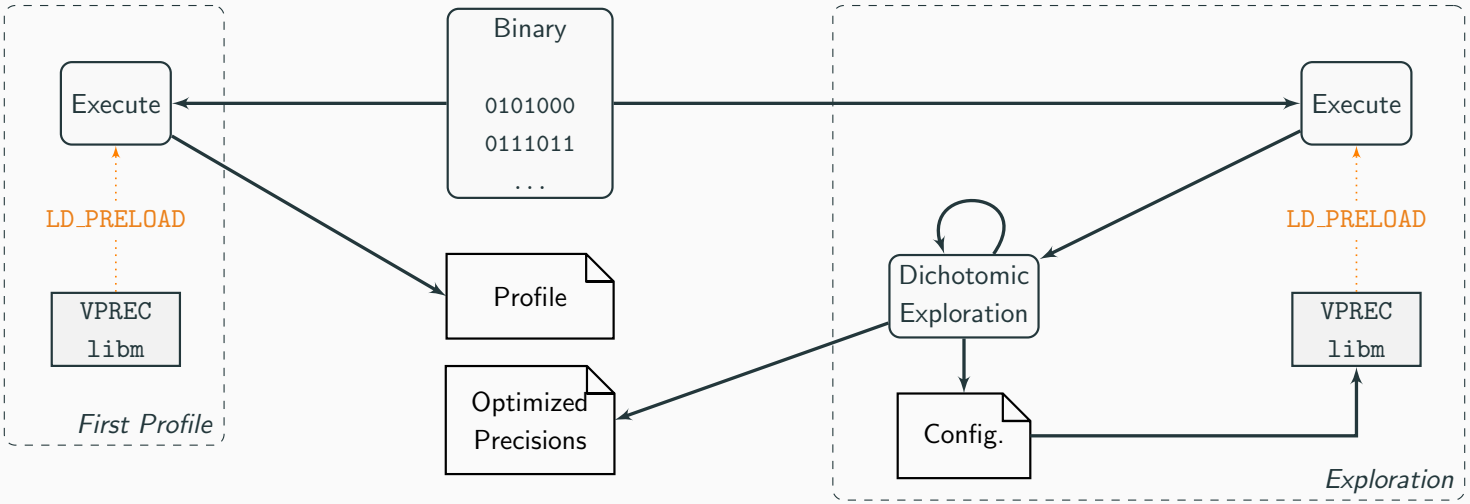


# VPREC-libm: Simulate a Variable-Precision libm




- simulate a custom-precision math library (`libm`)
- library computations done using `binary128` (`libquadmath`)
  - emulate operations that fit on `binary32` and `binary64`
  - **precise-enough**
  - output accuracy $\leq$ **1ulp**
- **VPREC features:**
  - inputs rounded to user-defined format
  - outputs rounded to user-defined format
  - or both

# VPREC-libm: Intercept Calls to libm



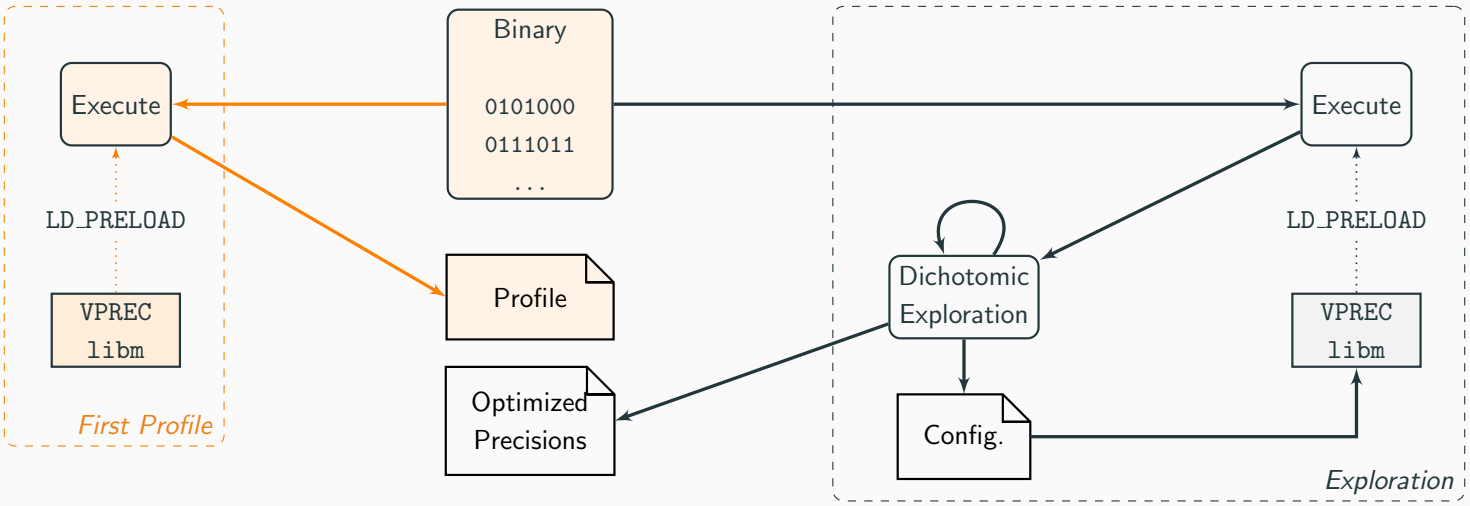

Through **library preloading** using LD_PRELOAD

- load *custom libraries* before the *system libraries*, overriding them
- allows library functions to be **intercepted** and **replaced**

**Advantages:**

- program behavior can be **modified non-invasively**
- **no** need to **recompile**

# VPREC-libm: Run Target Code




**VPREC-libm** creates a **profile of calls** to the `libm`:

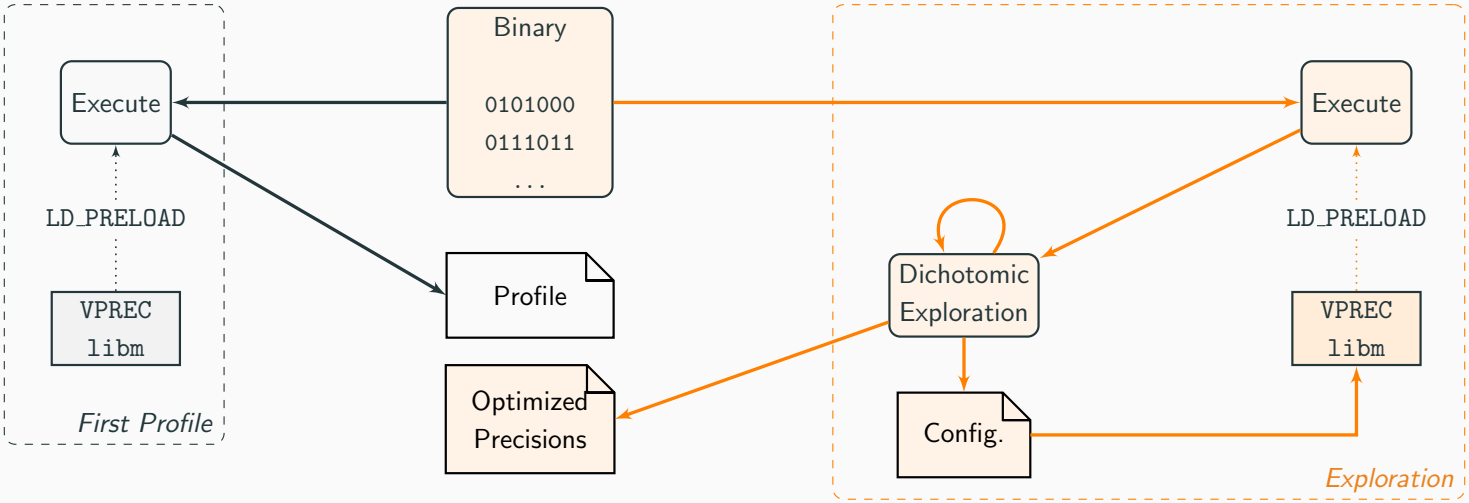
- what **functions** are **called** → what functions are needed in `libm`
- **frequency** of the calls → which functions to optimize from `libm`
- **domain** of the inputs → what formats to use
  and what optimizations* can be done

**VPREC-libm** creates a **configuration for future calls** to the `libm`.

# VPREC-libm: Code Profile Example

```
Function Type       Function ID       Nb Calls       Min Arg       Max Arg

ACOS      _start+0x2a/__libc_start_main+0xe7/main+0xd61/_Z6rv2coePdS_dRdS0_S0+0x392/acos      634
-1.065937e-01      1.000000e+00
FABS      _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x79e2/fabs
183      7.200000e+02      9.360000e+03
COS      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x7b41/cos
7      9.513252e-01      4.666078e+00
SIN      _start+0x2a/__libc_start_main+0xe7/main+0xd61/_Z6rv2coePdS_dRdS0_S0+0xa29/_Z8newtonnuddRdS_+0x1b2/sin
634      -2.601703e-04      -3.138574e+00
COS      _start+0x2a/__libc_start_main+0xe7/main+0x4d3/_Z10twoline2rvPcS_ccc13gravconsttypeRdS1_S1_R8elsetrec+0xf5f/
_Z8sgp4init13gravconsttypecidddddddddR8elsetrec+0x28c6/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x7ee1/cos
5      2.703247e+00      -7.929611e+00
SQRT      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x163b/sqrt      32
1.022487e+00      3.788501e+01
FLOOR      _start+0x2a/__libc_start_main+0xe7/main+0xbf7/_Z7invjdaydRiS_S_S_S_Rd+0x11a/floor      6
2.600000e+01      2.600000e+01
COS      _start+0x2a/__libc_start_main+0xe7/main+0x4d3/_Z10twoline2rvPcS_ccc13gravconsttypeRdS1_S1_R8elsetrec+0xf5f/
_Z8sgp4init13gravconsttypecidddddddddR8elsetrec+0x1fbe/+0x3d4e/cos      24      4.950053e-01      6.198556e+00
FABS      _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x1167/fabs      487
8.902677e-01      2.000000e+00
FMOD      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xaf1/fmod      66
4.972443e-03      6.283185e+00
SIN      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x13fa/sin      133
3.992762e-04      5.749288e+00
COS      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x1421/cos      133
3.992762e-04      5.749288e+00
SIN      _start+0x2a/__libc_start_main+0xe7/main+0x53a/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x7e06/sin
5      -3.706113e+00      -1.394880e+01
COS      _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x7e54/cos
140      5.036263e-01      6.136305e+00
```

# VPREC-libm: Optimize the Configuration




**VPREC-libm explores** possible configurations to find an **optimal** one:

- **gradually reduce** the **precision** of `VPREC-libm` functions
  - from most performance impactful, to least impactful
- **automate** as much as possible
  - **scripted** process
  - floating-point **range** and **precision computed** automatically

# VPREC-libm: Optimal Configuration Example

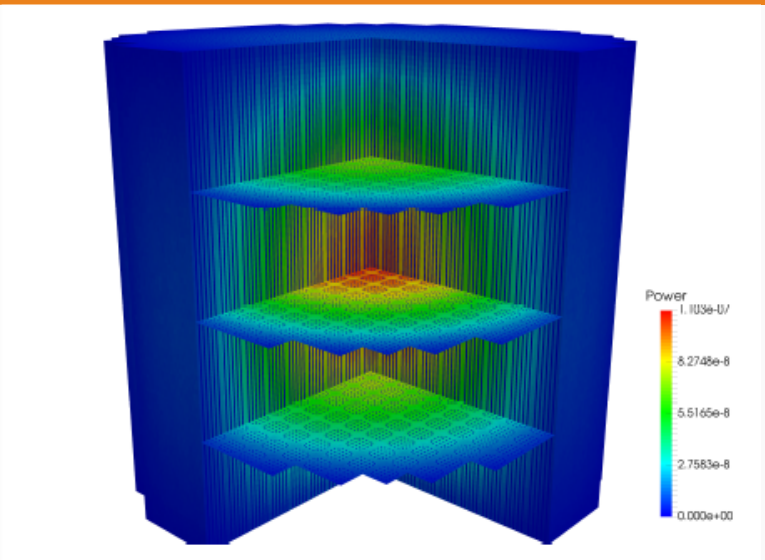
```
Function Type       Function ID       Nb Calls       FP Range       FP Precision

FABS _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x13bc/fabs 3187 7 2
COS _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x1421/cos 2549 1 50
SIN _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x13fa/sin 2549 1 50
FABS _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x14cc/fabs 2549 1 0
POW _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x956/pow 1280 4 49
POW _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x9a9/pow 1280 4 29
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xb90/fmod 1276 3 52
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xb57/fmod 1276 3 49
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xaf1/fmod 1276 3 52
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xb24/fmod 1276 3 52
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x136b/fmod 1276 3 51
ATAN2 _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x177e/atan2 1274 2 50
ATAN2 _start+0x2a/__libc_start_main+0xe7/main+0xd61/_Z6rv2coePdS_dRdS0_S0_S0+0xa29/_Z8newtonnuddRdS_+0x184/atan2
1268 2 31
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xd61/_Z6rv2coePdS_dRdS0_S0_S0+0xa29/_Z8newtonnuddRdS_+0x444/fmod 1268
1 0
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xd61/_Z6rv2coePdS_dRdS0_S0_S0+0xa29/_Z8newtonnuddRdS_+0x3e9/fmod 1268
2 33
FMOD _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0x904/+0x7895/fmod 974 3
49
SQRT _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xc5/
_Z12getgravconst13gravconsttypeRdS0_S0_S0+0x12a/sqrt 640 5 45
SIN _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xbb7/sin 638 1 30
COS _start+0x2a/__libc_start_main+0xe7/main+0xa09/_Z4sgp413gravconsttypeR8elsetrecdPdS2_+0xbde/cos 638 3 33
```

# VPREC-libm + MKL VML: Optimal Configuration Search

**Intel MKL VML:**

- math library optimized for Intel architectures
- vectorized ... but not only
- data formats:
  - `binary32`
  - `binary64`
- accuracy modes:
  - *High-Accuracy* $\longrightarrow$ most accurate (1ulp)
  - *Low-Accuracy* $\longrightarrow$ less accurate (4ulp)
  - *Enhanced Performance* $\longrightarrow$ highest performance
- encoded as (in the following):
  - **0** for `binary32` Enhanced-Performance (**least precise**)
  - ...
  - **5** for `binary64` High-Accuracy (**most precise**)

# PATMOS – Prototype Monte Carlo Neutron Transport Code

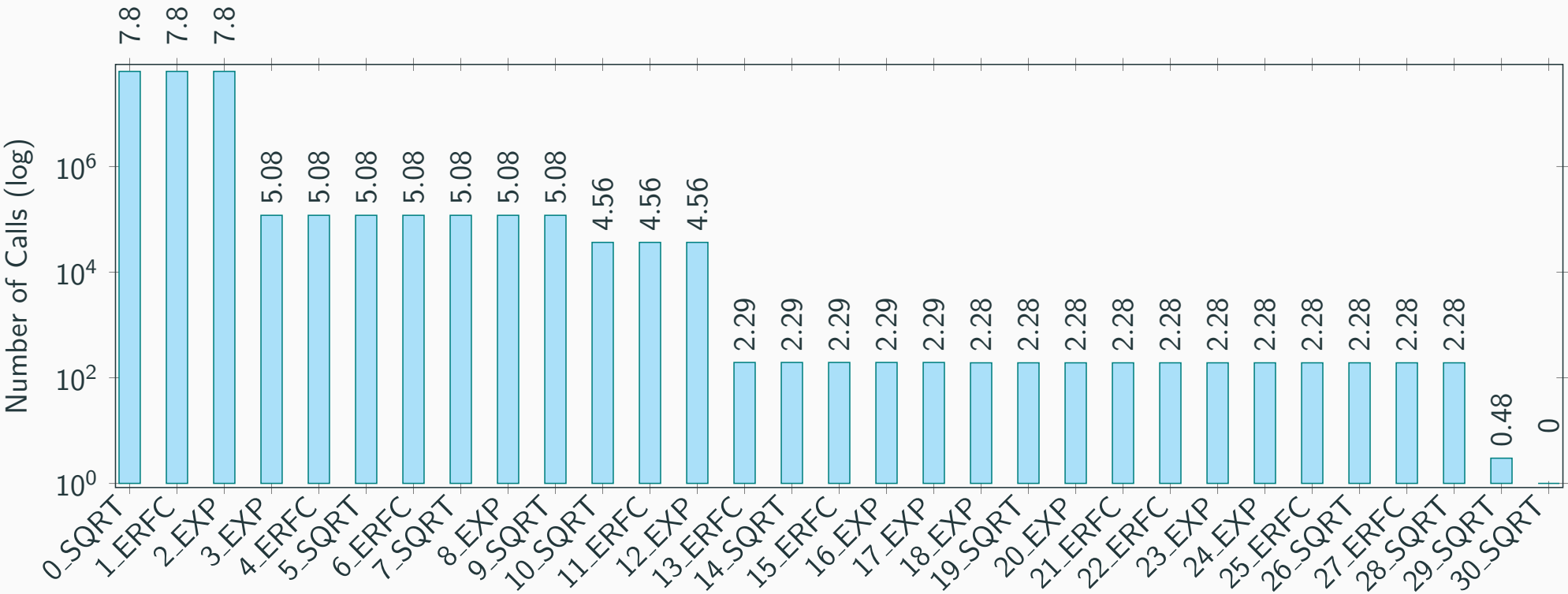



- perform pin-by-pin full core depletion calculations for large nuclear power reactors
- **simulate the life of a particle**, from its creation to its destruction
- a succession of collisions and movements
- materials made of **different nuclei**
- **different probabilities** of a particle **to interact with the nuclei**
- **objectives:**
  - **collisions:** measure the number of collisions
  - **trace:** measure the distance traveled in a given volume
- collaboration with Davide Mancusi and Emeric Brun of CEA



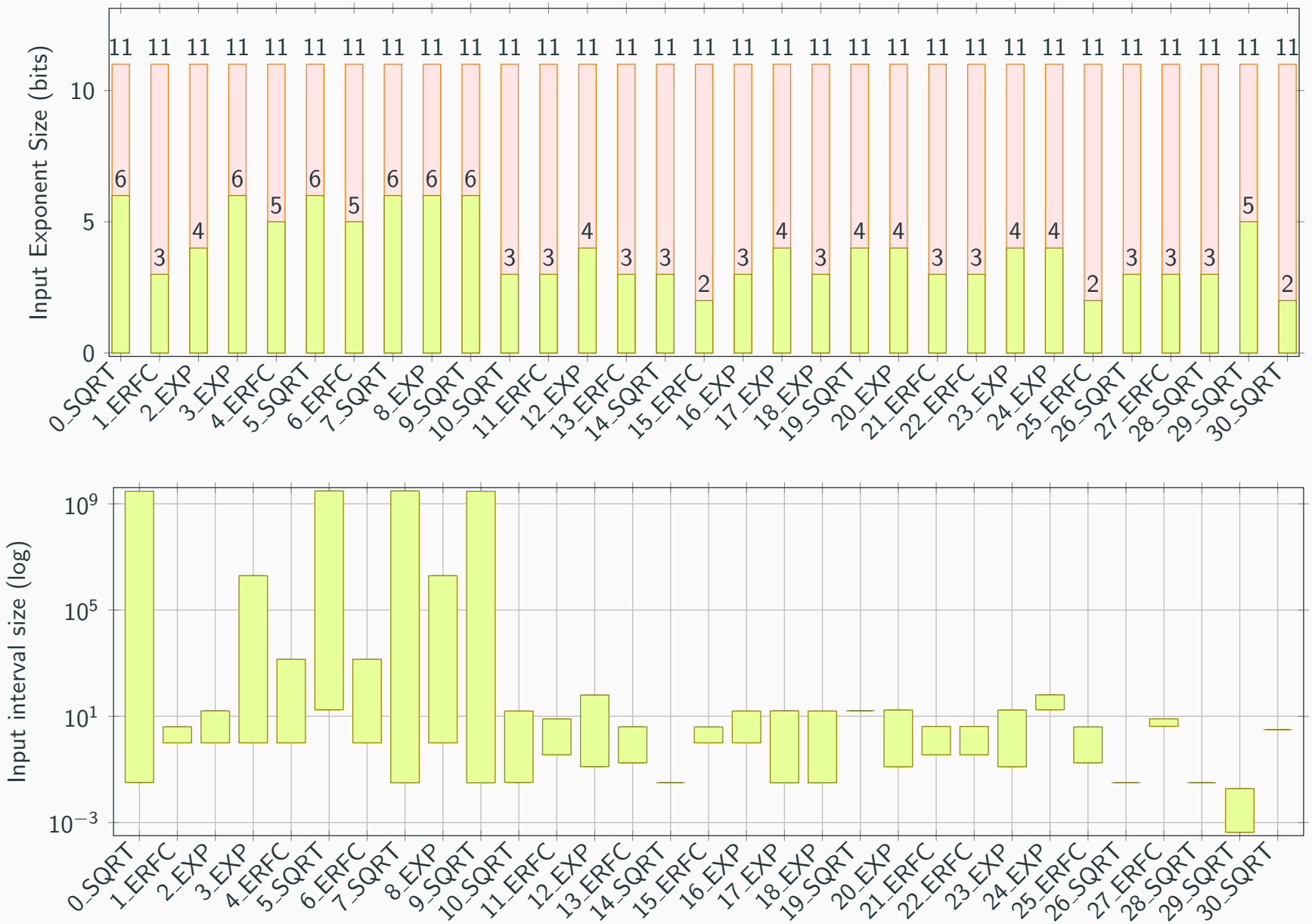
*Image Credit: CEA, https://www.cea.fr/energies/tripoli-4/Pages/Patmos/patmos.aspx

# PATMOS: `libm` Call-Site Path Frequency



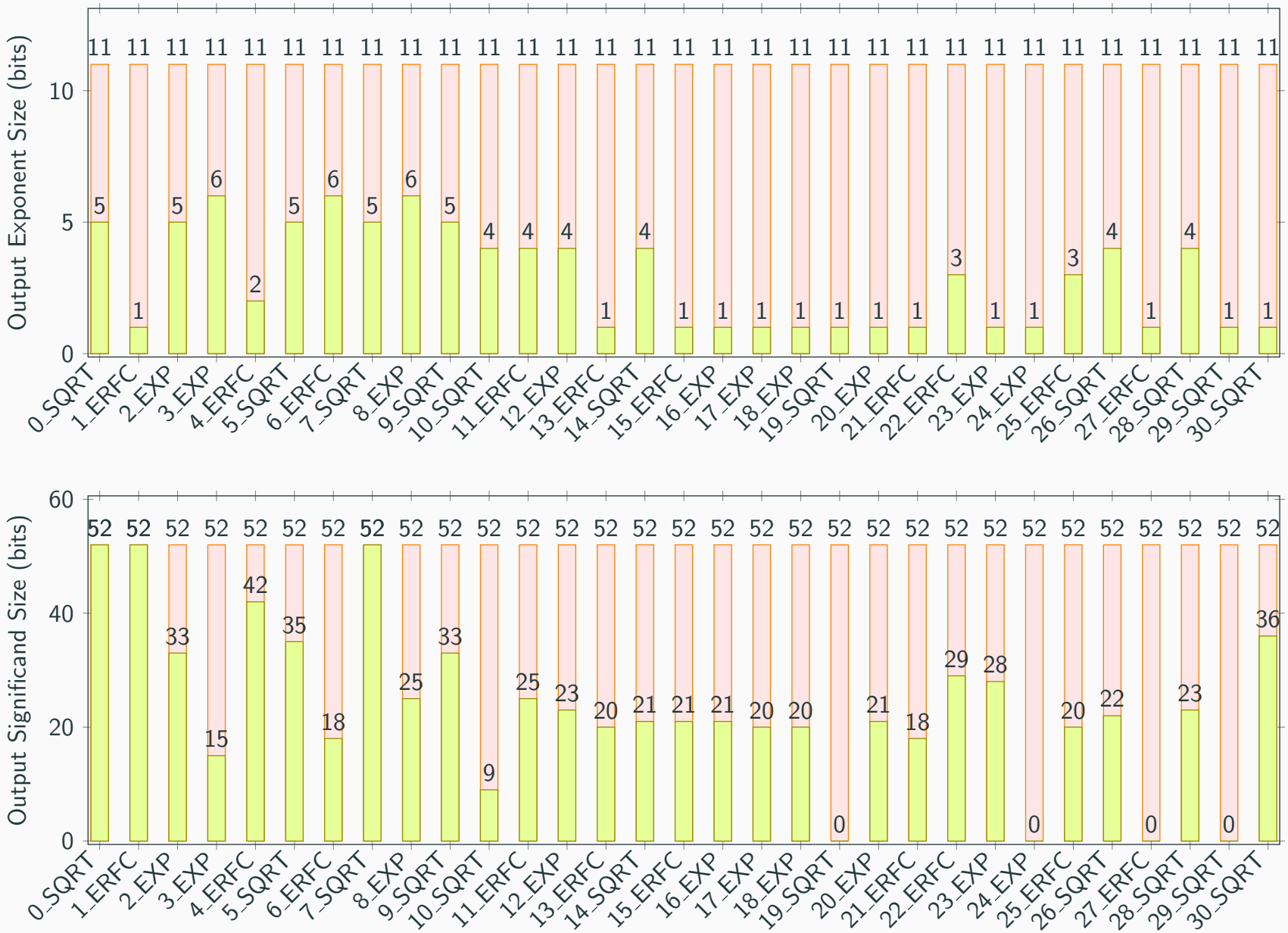

Call-site path frequency for PATMOS kernel

# PATMOS: Input Data Ranges



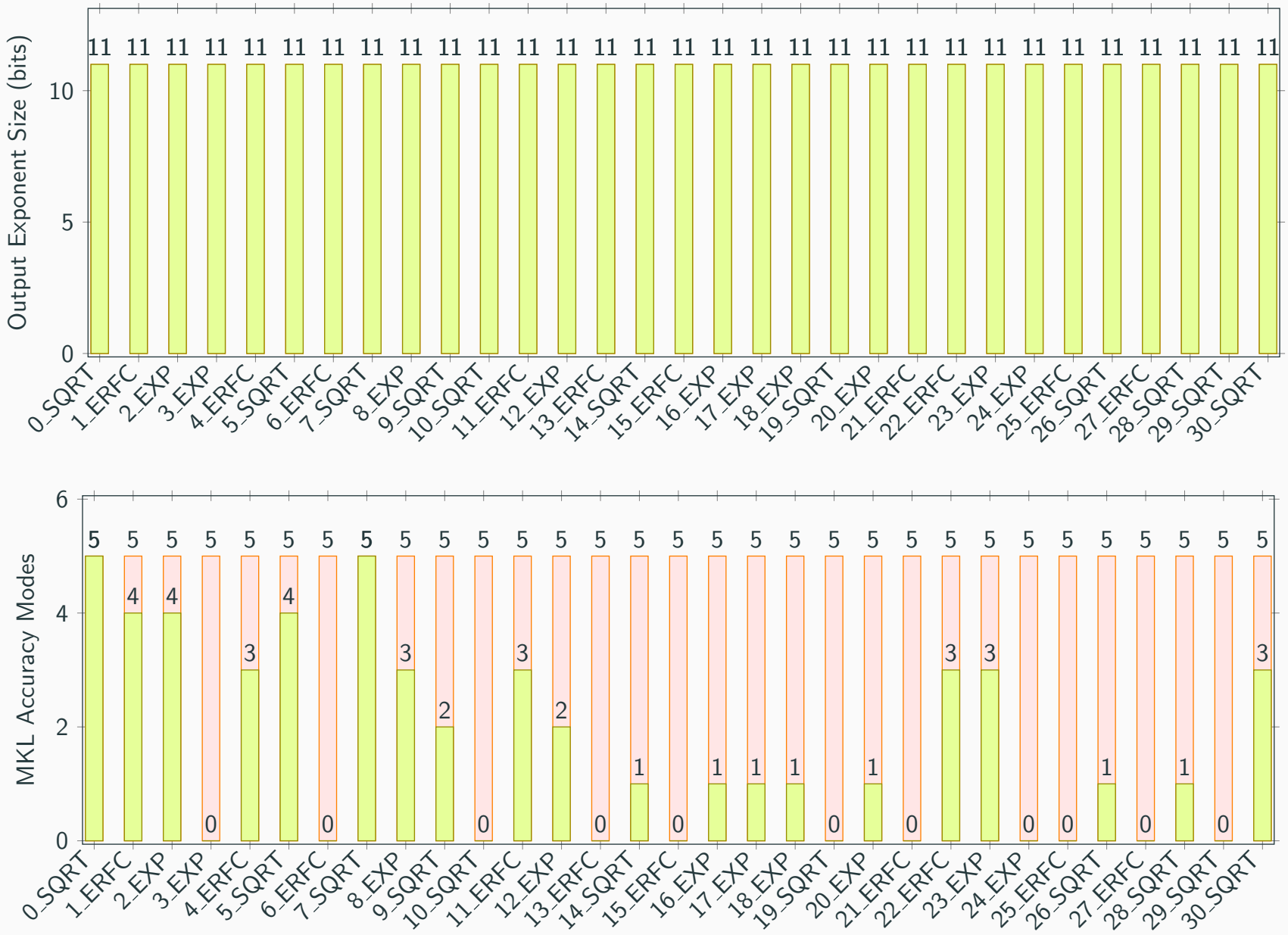

Input data ranges and computed exponent size for PATMOS kernel

# PATMOS: Output Data Formats




Output data formats for PATMOS kernel, accurate to $10^{-6}$, relative error

# PATMOS: MKL VML Accuracy modes




MKL accuracy modes for PATMOS kernel, accurate to $10^{-6}$, relative

# Measuring the Execution Time for PATMOS compute kernel

**Reference executions:**

- `gcc` and `libm`
  - execution time: **2.63424s**
- `gcc`/`icc` and `VML` *(all math functions in HA)*
  - scalar calls (unvectorized)
  - execution time: **6.7570s**
  - maximum absolute relative error wrt. `gcc+libm`: **3.648e-7**
  - ... **but more precise than** `gcc+libm`
- `icc` and `imf` *(default settings)*
  - execution time: **8.8268s**
  - maximum absolute relative error wrt. `gcc+libm`: **1.073e-5**
  - ... **but much more precise than** `gcc+libm`
- `icc` and `SVML` *(all math functions in HA)*
  - execution time: **0.80044s**
  - maximum absolute relative error wrt. `gcc+libm`: **1.072e-5**
  - ... **but much more precise than** `gcc+libm`

# Performance Improvements Using VPREC-libm

**Replace calls to the `libm` with calls to the `MKL VML`:**

- automated process
  - precision determined from precision exploration
  - automated modification of the source files

**Results using `gcc`/`icc` and `VML`** *(scalar calls, unvectorized; error wrt. all HA mode)*:

| VML mode | execution time (s) | abs. max. rel. error |
|---|---|---|
| all HA | 6.7570 | – |
| all LA | 6.3989 | 2.949e-6 |
| all EP | 6.1758 | 8.201e-5 |
| **custom VPREC-libm** | 6.5161 | 8.662e-7 |

**Results using `icc` and `SVML`** *(error wrt. high mode)*:

| SVML mode | execution time (s) | abs. max. rel. error |
|---|---|---|
| high (HA) | 0.80044 | – |
| medium (LA) | 0.68405*s* | 2.234e-7 |
| low (EP) | 0.60944 | 8.196e-5 |

# Future Work

Future work:

- generation of optimized versions of the math functions used in the PATMOS kernel
- generate one approximation for the function computed by the PATMOS kernel
  - polynomial approximation
  - code conversion from floating-point to fixed-point
- explore `VML` and `SVML` options

**Goal:**

- **generate a custom libm**, to satisfy fine-grain user requirements
  - enable faster execution
- **profile-guided optimizations**
  - speed-up branching inside compiler
    - speculatively reduce the function domain and/or accuracy

    $\Rightarrow$ limit degree of polynomial approximaton

    $\Rightarrow$ inline final implementation

  $\Rightarrow$ allows for faster execution

# Bibliography i



Yohan Chatelain, Pablo de Oliveira Castro, Eric Petit, David Defour, Jordan Bieder, and Marc Torrent, *VeriTracer: Context-enriched tracer for floating-point arithmetic analysis*, 25th IEEE Symposium on Computer Arithmetic, ARITH 2018, Amherst, MA, USA. June 25th-27th, 2018, IEEE, 2018, pp. 65–72.


Yohan Chatelain, Eric Petit, Pablo de Oliveira Castro, Ghislain Lartigue, and David Defour, *Automatic exploration of reduced floating-point representations in iterative methods*, Euro-Par 2019 Parallel Processing - 25th International Conference, Lecture Notes in Computer Science, Springer, 2019.

# Bibliography ii



Christophe Denis, Pablo de Oliveira Castro, and Eric Petit, *Verificarlo: checking floating point accuracy through monte carlo arithmetic*, Computer Arithmetic (ARITH), 23nd Symposium on, IEEE, 2016, pp. 55–62.


Devan Sohier, Pablo de Oliveira Castro, François Févotte, Bruno Lathuilière, Eric Petit, and Olivier Jamond, *Confidence Intervals for Stochastic Arithmetic*, preprint, July 2018.