Practical Numerical Profiling: Unlocking Low-precision Floating-Point for Scientific Computing

Faveo Hoerold[1,3], Ivan R. Ivanov[2,3], Akash Dhruv[4], William S. Moses, Anshu Dubey[4], Mohamed Wahib[3], Jens Domke[3]

1. ETH Zurich
2. Institute of Science Tokyo
3. RIKEN R-CCS
4. Argonne National Laboratory

Why low-precision floating-point?

AI is changing floating-point

"Artifical intelligence, particularly deep learning has profoundly impacted floating-point computation."

Dongarra et al. https://doi.org/10.48550/arXiv.2411.12090

AI is changing floating-point

"Artifical intelligence, particularly deep learning has profoundly impacted floating-point computation."

"As a result, **reduced-precision** floating-point types, [...] have become **increasingly popular in AI** applications."

Dongarra et al. https://doi.org/10.48550/arXiv.2411.12090

Where did all my FLOPS go?

	H200 SXM[1]	H200 NVL[1]
FP64	34 TFLOPS	30 TFLOPS
FP64 Tensor Core	67 TFLOPS	60 TFLOPS
FP32	67 TFLOPS	60 TFL
TF32 Tensor Core²	989 TFLOPS	835 TFL
BFLOAT16 Tensor Core²	1,979 TFLOPS	1,671 TFL
FP16 Tensor Core²	1,979 TFLOPS	1,671 TFLO
FP8 Tensor Core²	3,958 TFLOPS	3,341 TFLO
INT8 Tensor Core²	3,958 TFLOPS	3,341 TFLOP
GPU Memory	141GB	141GB
GPU Memory Bandwidth	4.8TB/s	4.8TB/s

Hopper

Blackwell

Individual Blackwell Ultra GPU Specifications		
FP4 Tensor Core[1]	20 P PS | 15 PFLOPS	18 PFLOPS | 14 PFLOPS
FP8/FP6 Tensor Core²	PFLOPS	9 PFLOPS
INT8 Tensor Core²	TOPS	307 TOPS
FP16/BF16 Tensor Core²	LOPS	4.5 PLFOPS
TF32 Tensor Core²	2.5 LOPS	2.2 PFLOPS
FP32	80 LOPS	75 TFLOPS
FP64/FP64 Tensor Core	1.3 TFLOPS	1.2 TFLOPS
GPU Memory | Bandwidth	279 GB HBM3E | 8 TB/s	270 GB HBM3E | 7.7 TB/s

AMD is taking a different approach…

Ozaki Schemes:

Nick Malaya
https://www.hpcwire.com/2026/03/13/amd-hints-at-big-fp64-increases-in-mi430x-gpu-as-ozaki-underwelms/

AMD is taking a different approach…

Ozaki Schemes:

✖ Not IEEE-compliant

Nick Malaya
https://www.hpcwire.com/2026/03/13/amd-hints-at-big-fp64-increases-in-mi430x-gpu-as-ozaki-underwelms/

AMD is taking a different approach…

Ozaki Schemes:

✖ Not IEEE-compliant

✖ Deviate from true FP64

Nick Malaya
https://www.hpcwire.com/2026/03/13/amd-hints-at-big-fp64-increases-in-mi430x-gpu-as-ozaki-underwelms/

AMD is taking a different approach…

Ozaki Schemes:

- ✖ Not IEEE-compliant
- ✖ Deviate from true FP64
- ✖ Only DGEMM

Nick Malaya
https://www.hpcwire.com/2026/03/13/amd-hints-at-big-fp64-increases-in-mi430x-gpu-as-ozaki-underwelms/

AMD is taking a different approach…

Ozaki Schemes:

- ✖ Not IEEE-compliant
- ✖ Deviate from true FP64
- ✖ Only DGEMM

MI325X: ~80 TFLOPS

MI430X: ~200 TFLOPS ?

Nick Malaya
https://www.hpcwire.com/2026/03/13/amd-hints-at-big-fp64-increases-in-mi430x-gpu-as-ozaki-underwelms/

Scientific simulations: Still relevant

"[...] physics-based models will remain vital, simply because the physics generalizes in ways that are designed to fill in [...] sparse observations"

Bauer et al. https://doi.org/10.1038/s43017-023-00468-z
Zhang et al. https://doi.org/10.48550/arXiv.2508.15724

How to low-precision floating-point?

How to low-precision floating-point?

Keep it simple!

How to low-precision floating-point?

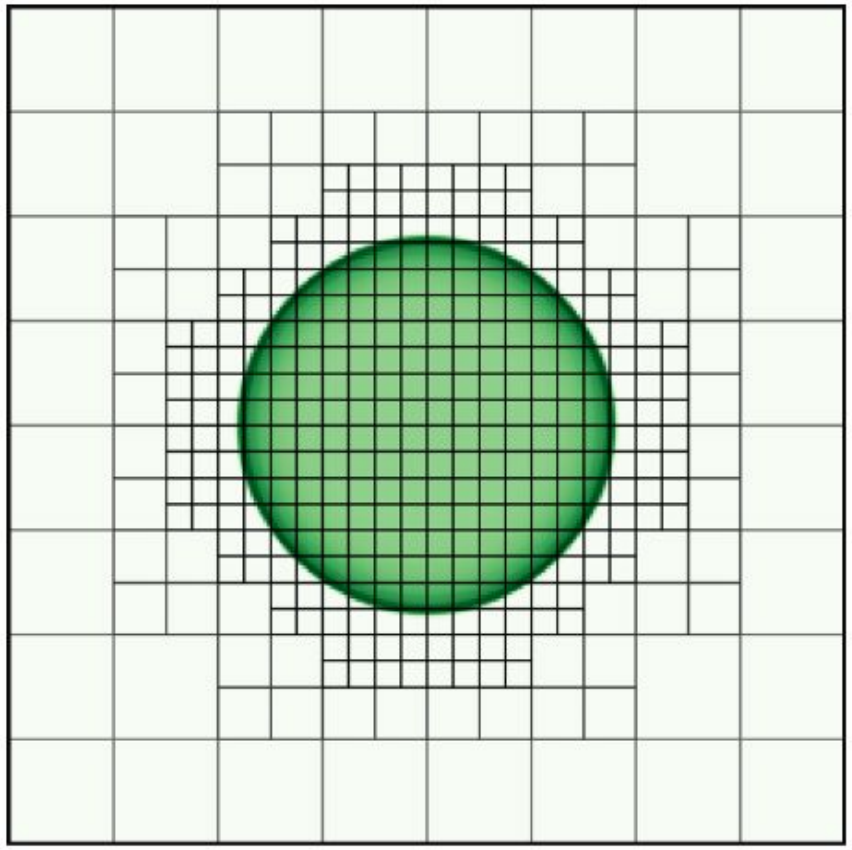
Keep it simple!

Domain scientists know what they are doing!

How to low-precision floating-point?

Keep it simple!

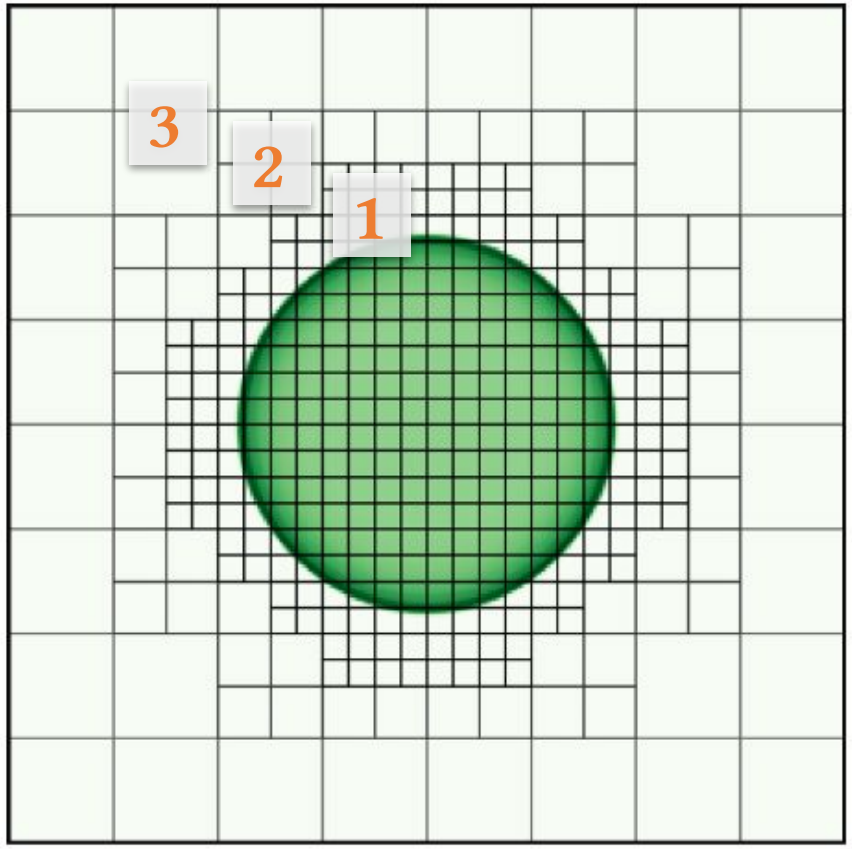
Domain scientists know what they are doing!



How to low-precision floating-point?

Keep it simple!

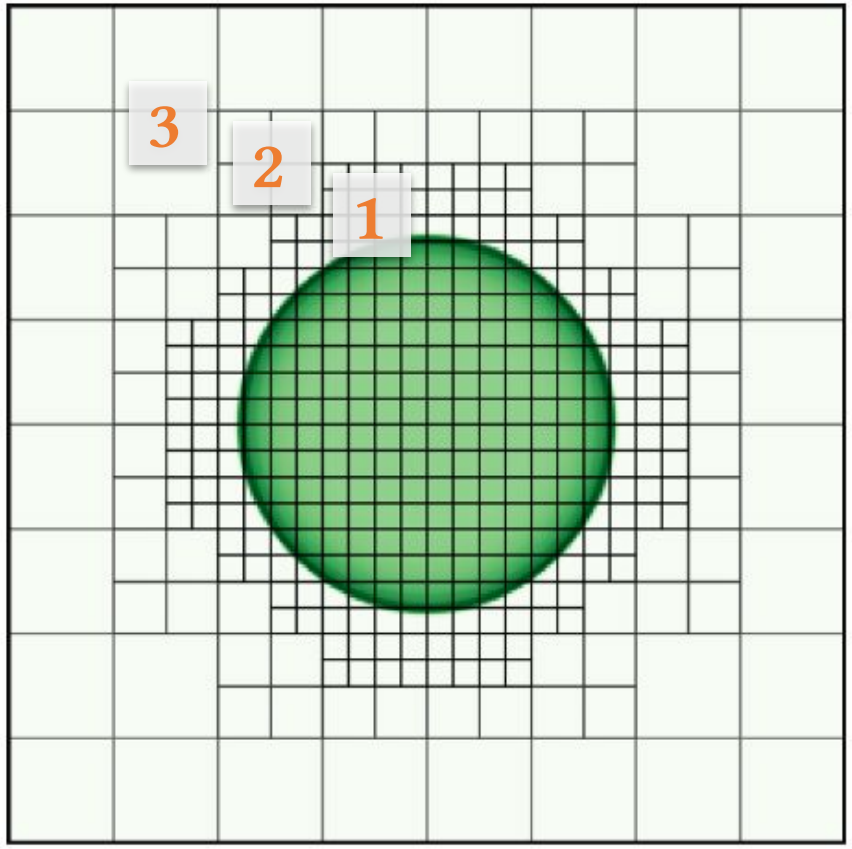
Domain scientists know what they are doing!



How to low-precision floating-point?

Keep it simple!

Domain scientists know what they are doing!



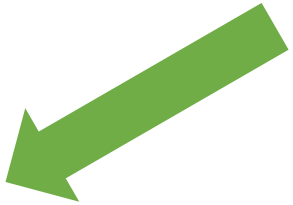
Test their hypothesis.

Empirical studies

```
find . -name *.cpp -exec sed -i 's/double/float/g' {} \;
```

Empirical studies

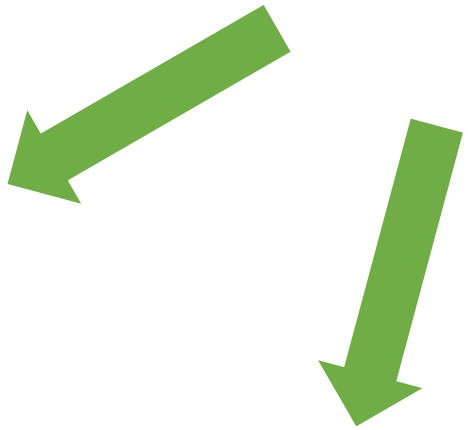
```
find . -name *.cpp -exec sed -i 's/double/float/g' {} \;
```



```
fp32
tf32
bfloat16
fp16
fp8 e5m2
fp8 e4m2
fp4
...
```

Empirical studies

```
find . -name *.cpp -exec sed -i 's/double/float/g' {} \;
```

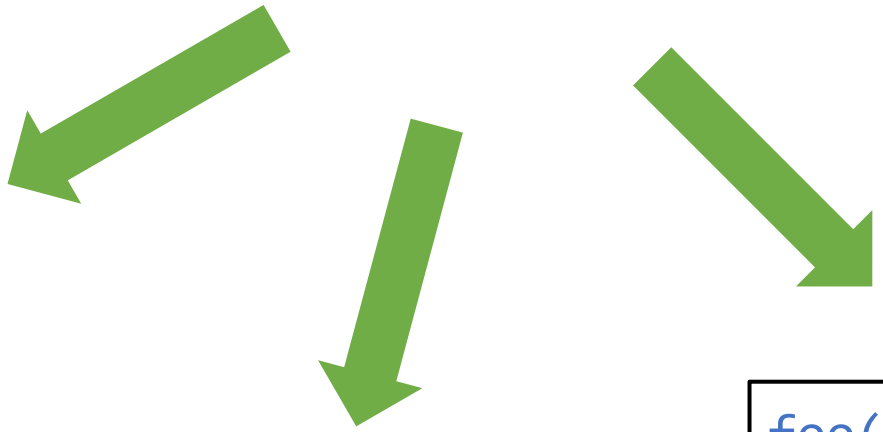


```
fp32
tf32
bfloat16
fp16
fp8 e5m2
fp8 e4m2
fp4
...
```

```
foo()
bar()
baz()
```

Empirical studies

~~`find . -name *.cpp -exec sed -i 's/double/float/g' {} \;`~~



```
fp32
tf32
bfloat16
fp16
fp8 e5m2
fp8 e4m2
fp4
...
```

```
foo()
bar()
baz()
```

```
foo()
if (s >= 3) {
  bar()
} else {
  bar()
}
baz()
```

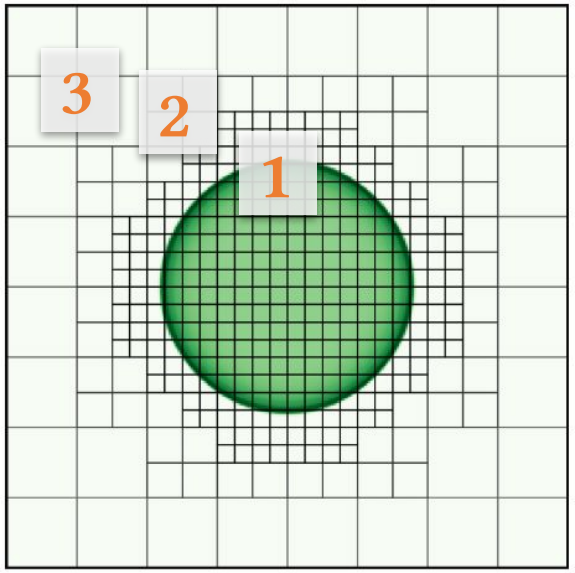
Empirical studies

~~~~
find . -name *.cpp -exec sed -i 's/double/float/g' {} \;
~~~~

```
fp32
tf32
bfloat16
fp16
fp8 e5m2
fp8 e4m2
fp4
...
```

```
foo()
bar()
baz()
```

```
foo()
if (s >= 3) {
  bar()
} else {
  bar()
}
baz()
```



Tool: Raptor

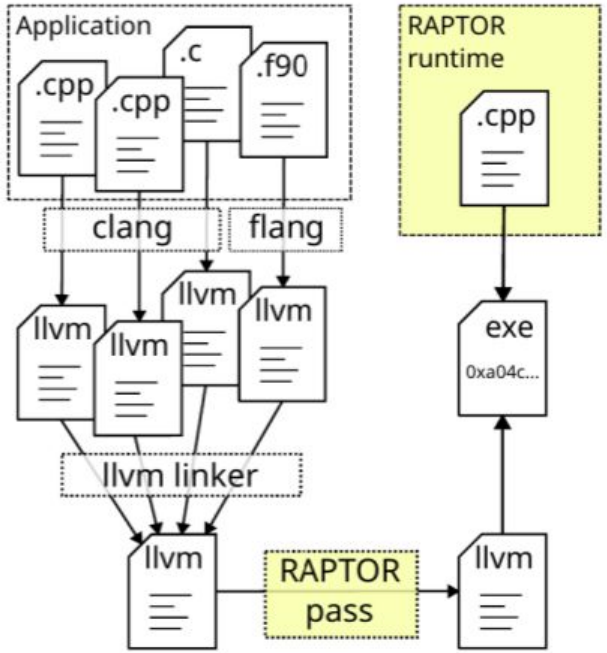
LLVM-based tool for precision experimentation with focus on scientific code

- Wide range of FP formats
- Ease of use
- Control of granularity
- C/C++/Fortran support
- CPU and GPU support

Tool: Raptor

Approach	Category	Feature set								Ref.
		Full app. truncation	Dynamic truncation	Flexible formats	Scoped truncation	Granular truncation	Error tracking	Non-∇ Code	Supported languages[7]	
ADAPT	B	👎	👎	👍	👎	👍	👍	👎	C, C++, Fortran	[22]
CADNA	C	👍	👎	👎	N/A	👍	👍	👍	Ada, C, Fortran	[17]
FPSpy	G	👍	👎	👎	N/A	N/A	👎	👍	Binary	[8]
FPVM	D	👍	👎	👍	👎	👎	👍	👍	Binary	[9]
GPU-FPX	G	👍	👎	N/A	N/A	👎	👍	👍	GPU Binary	[20]
GPUMixer	B	👎	👎	👎	👎	👎	👍	👎	CUDA	[18]
Jost et al.	C, E, F	👍	👎	👍	N/A	👍	👎	👍	C	[16]
Gu et al.	E	👎	👎	👎	👎	👍	👎	👍	C, C++	[14]
NEAT	E	👍	👍	👍	👎	👍	👎	👍	Binary	[3]
Precimonious	A	👎	N/A	👎	👍	👎	👎	👍	C	[25]
Puppeteer	D	👍	👍	N/A	👎	👍	👎	👎	C, C++	[24]
RAPTOR	B, C, E	👍	👍	👍	👍	👍	👍	👍	C, C++, Fortran	

RAPTOR: Architecture Overview



Two main components:

- LLVM pass
- Runtime

RAPTOR: Architecture Overview

Op-Mode: LLVM IR Transformation

fp64 to fp32 (available in hardware)

```
fp64 foo(fp64 %a, fp64 %b, fp64 %c):
  %d = fadd %a, %b : fp64
  call @print(%d : fp64)
  %e = fmul %c, %d : fp64
  return %e : fp64
```

```
fp64 foo_32(fp64 %a, fp64 %b, fp64 %c):
  %a32 = ftrunc %a to fp32
  %b32 = ftrunc %b to fp32
  %d32 = fadd %a32, %b32 : fp32
  %d = fexpand %d32 to fp64
  call @print(%d : fp64)
  %d32 = ftrunc %d to fp32
  %c32 = ftrunc %c to fp32
  %e32 = fmul %c32, %d32 : fp32
  %e = fexpand %e32 to fp64
  return %e : fp64
```

RAPTOR: Architecture Overview

Op-Mode: LLVM IR Transformation

Op-Mode: LLVM IR Transformation & Runtime

fp64 to fp(3, 6) (unavailable in hardware)

```
fp64 foo(fp64 %a, fp64 %b, fp64 %c):
  %d = fadd %a, %b : fp64
  call @print(%d : fp64)
  %e = fmul %c, %d : fp64
  return %e : fp64
```

```
fp64 foo_3_6(fp64 %a, fp64 %b, fp64 %c):
  %d = call @__raptor_op_fadd(%a, %b, 3, 6) : fp64
  call @print(%d : fp64)
  %e = call @__raptor_op_fmul(%c, %d, 3, 6) : fp64
  return %e : fp64
```

RAPTOR Runtime

```
double __raptor_op_fadd(double a, double b,
                        int e, int m) {
  mpfr_t ma, mb, mc;
  mpfr_set_exponent(e);
  mpfr_init2(ma, m); mpfr_init2(mb, m); mpfr_init2(mb, m);

  mpfr_set(ma, a); mpfr_set(ma, b);

  mpfr_add(mc, ma, mb);

  double c = mpfr_get_f(mc);

  mpfr_clear(ma);  mpfr_clear(mb); mpfr_clear(mc);
  return c;
}
```

MPFR:
Library for multiple-precision floating-point computation. Can be used to simulate floating-point computation with arbitrary exponent and mantissa size.

Note: Unavailable on GPUs

17

RAPTOR: Architecture Overview

Op-Mode: LLVM IR Transformation

Op-Mode: LLVM IR Transformation & Runtime

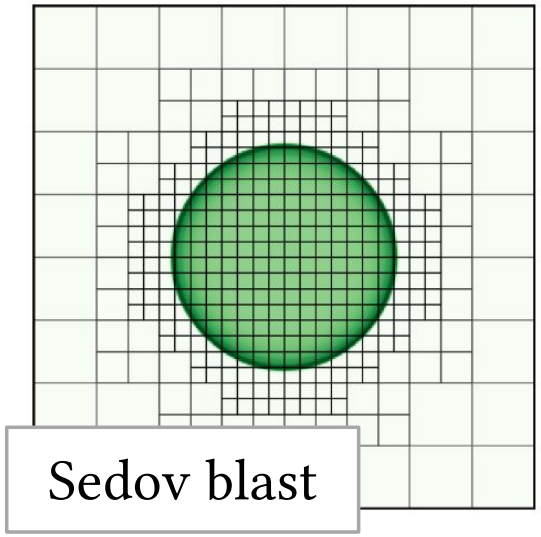
Op-Mode: LLVM IR Transformation

Automatically truncates all transitively called functions

```
fp64 foo(fp64 %a, fp64 %b):
  %e = call @bar(%a, %b, %c)
  return %e : fp64

fp64 bar(fp64 %a, fp64 %b):
  ...
```

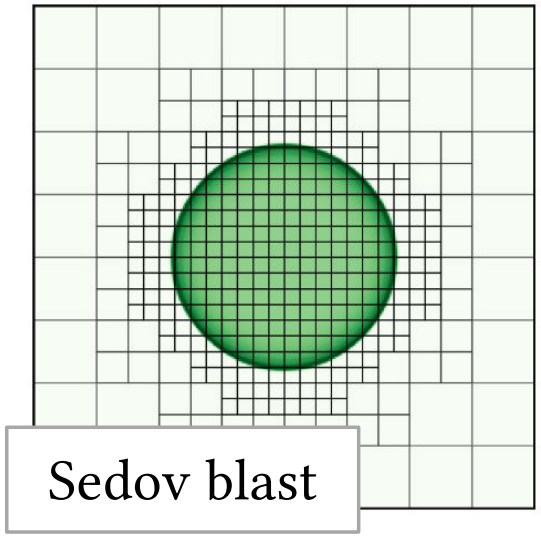
```
fp64 foo_32(fp64 %a, fp64 %b):
  %e = call @bar_32(%a, %b, %c)
  return %e : fp64

fp64 bar_32(fp64 %a, fp64 %b):
  ...
```

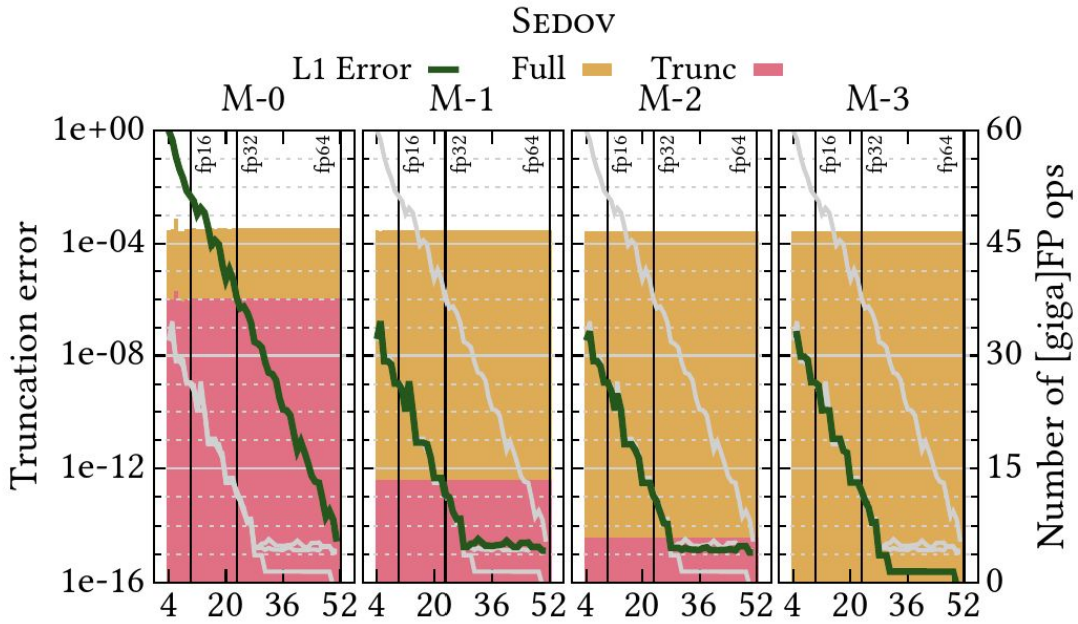
Sedov Blast and Sod Shock Analysis



Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

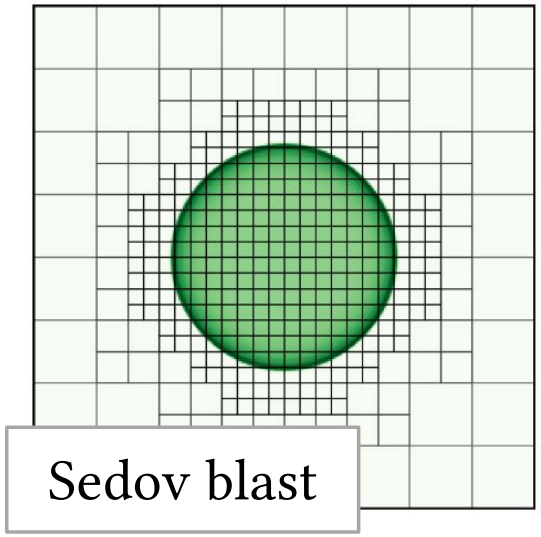
Sedov Blast and Sod Shock Analysis

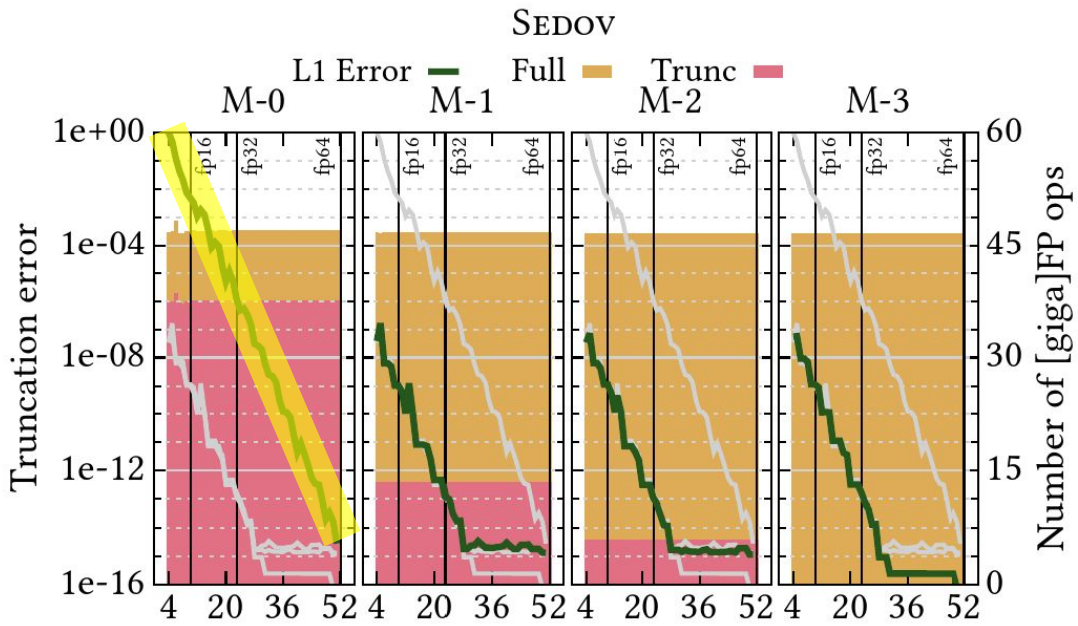




Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

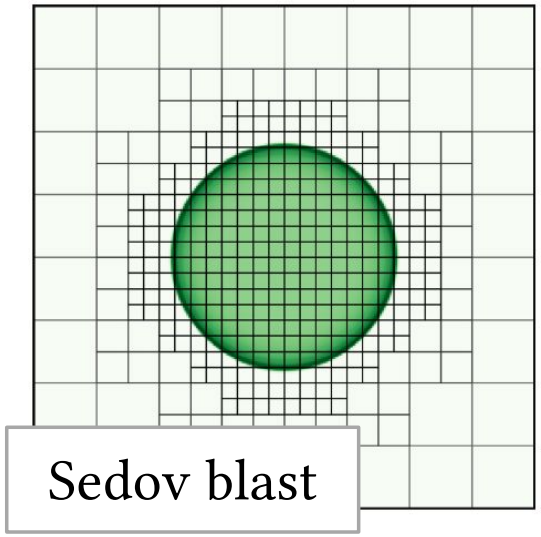
Sedov Blast and Sod Shock Analysis

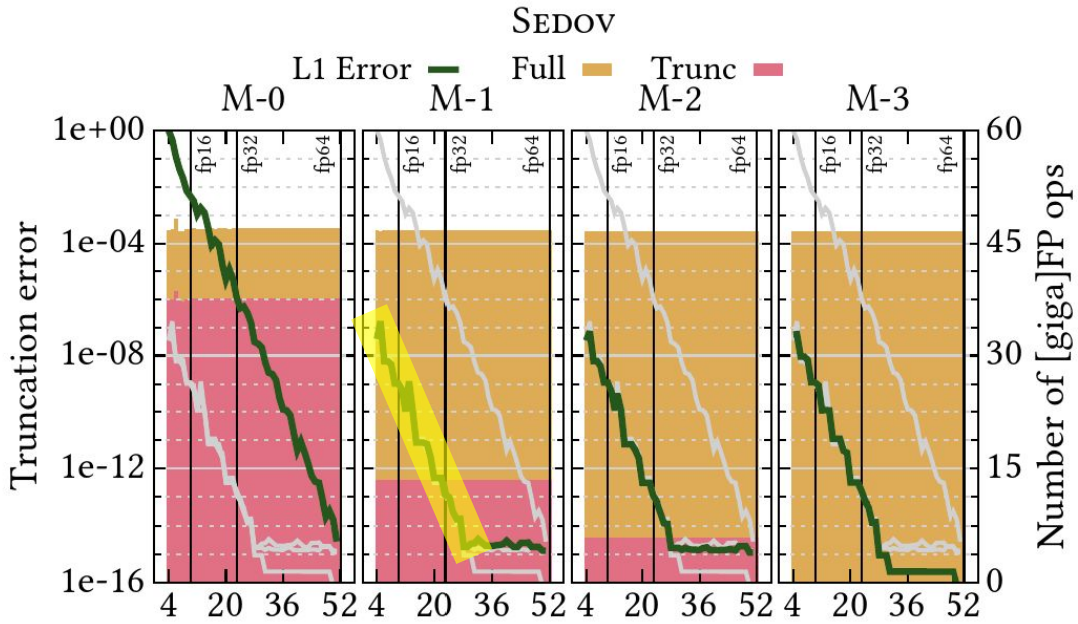




Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

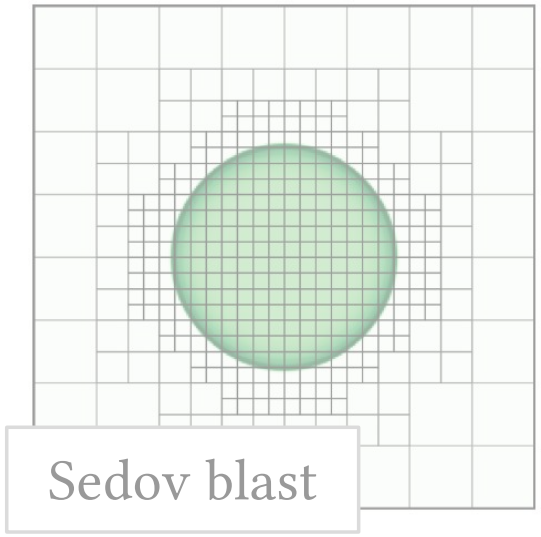
Sedov Blast and Sod Shock Analysis



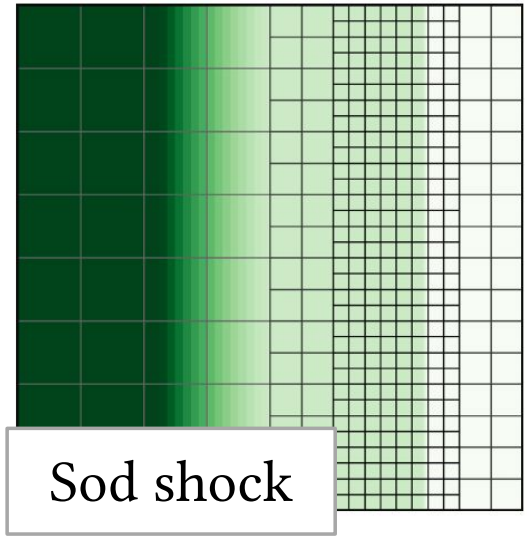


Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

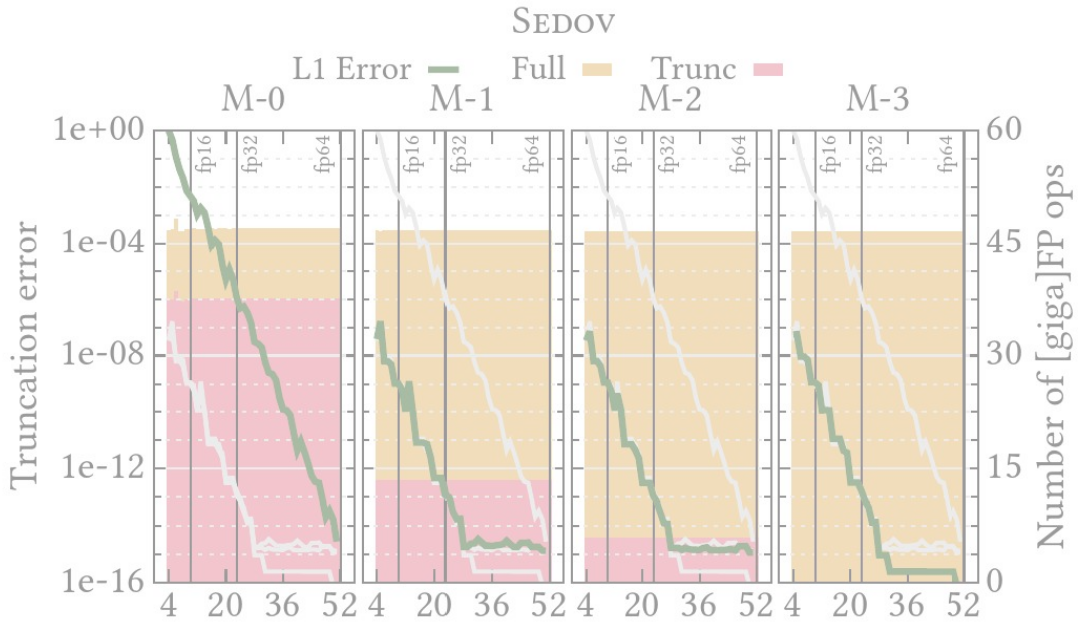
Sedov Blast and Sod Shock Analysis



Sedov blast

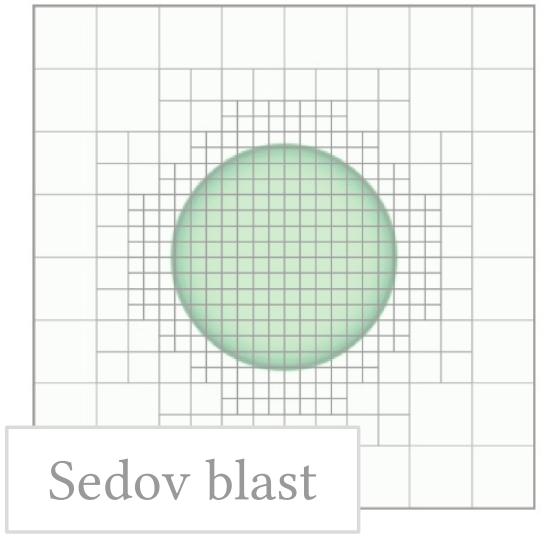


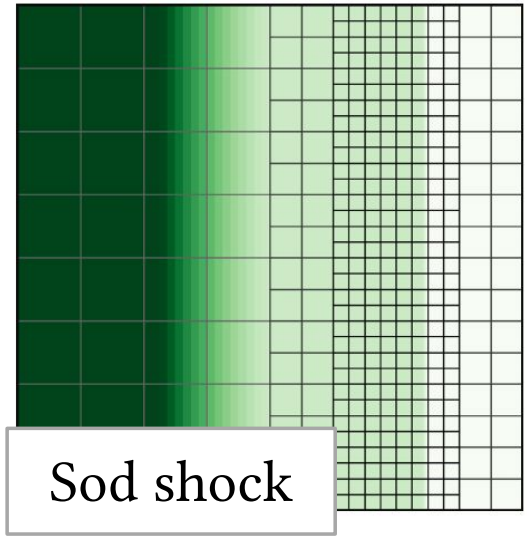
Sod shock

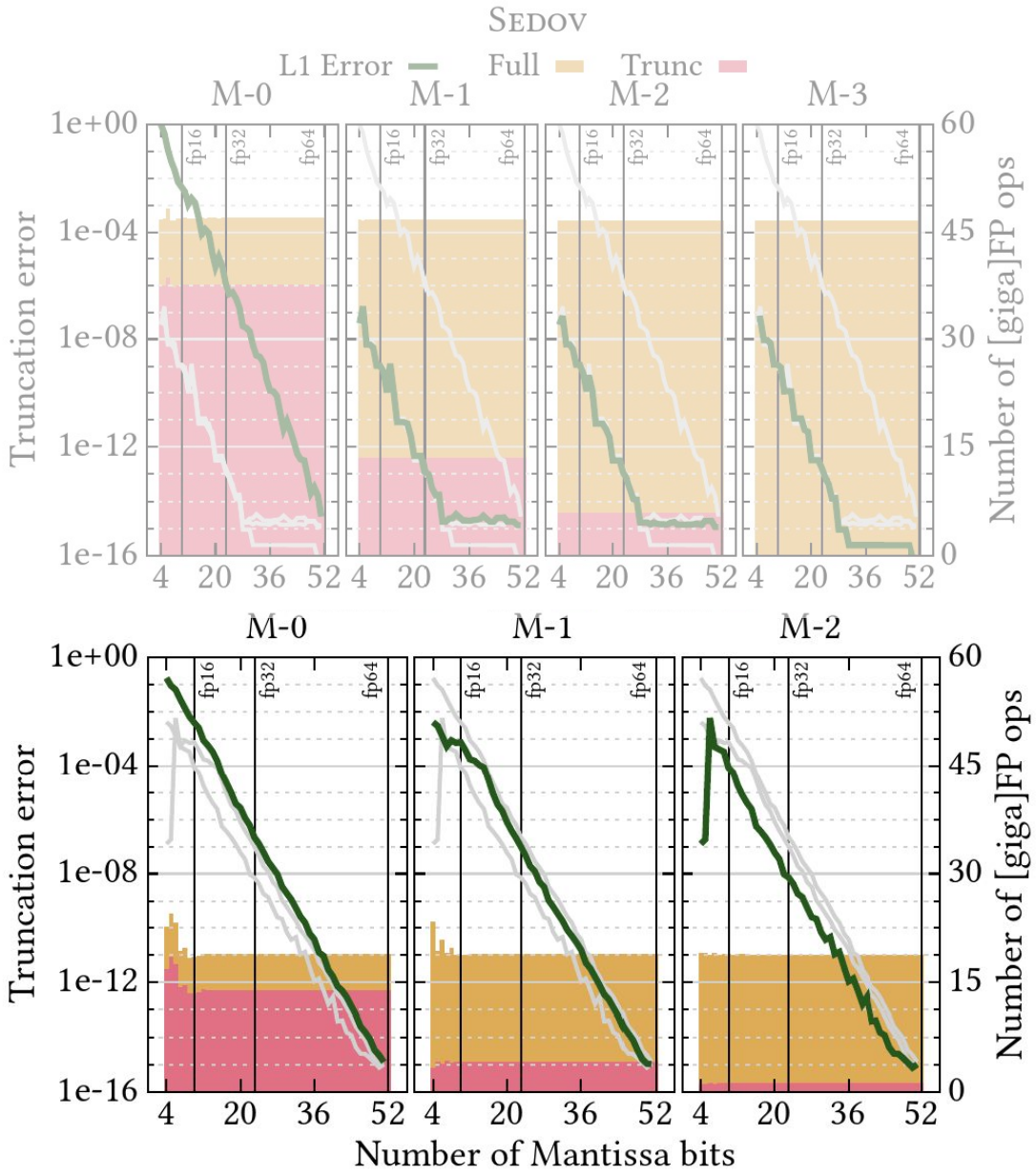


Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Sedov Blast and Sod Shock Analysis

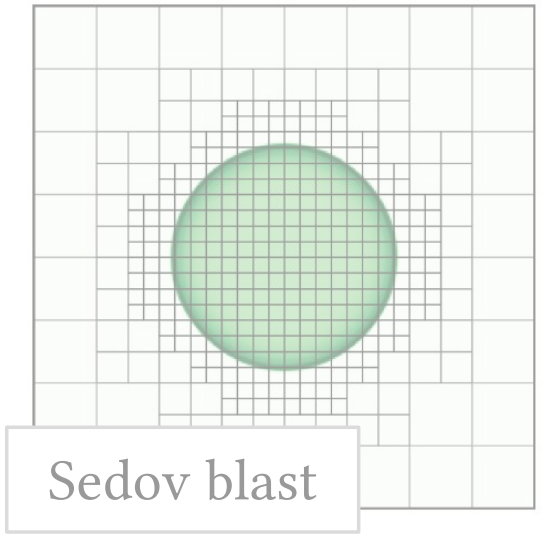


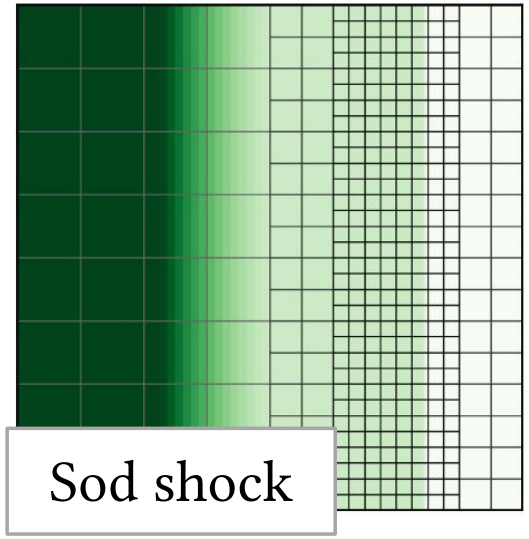


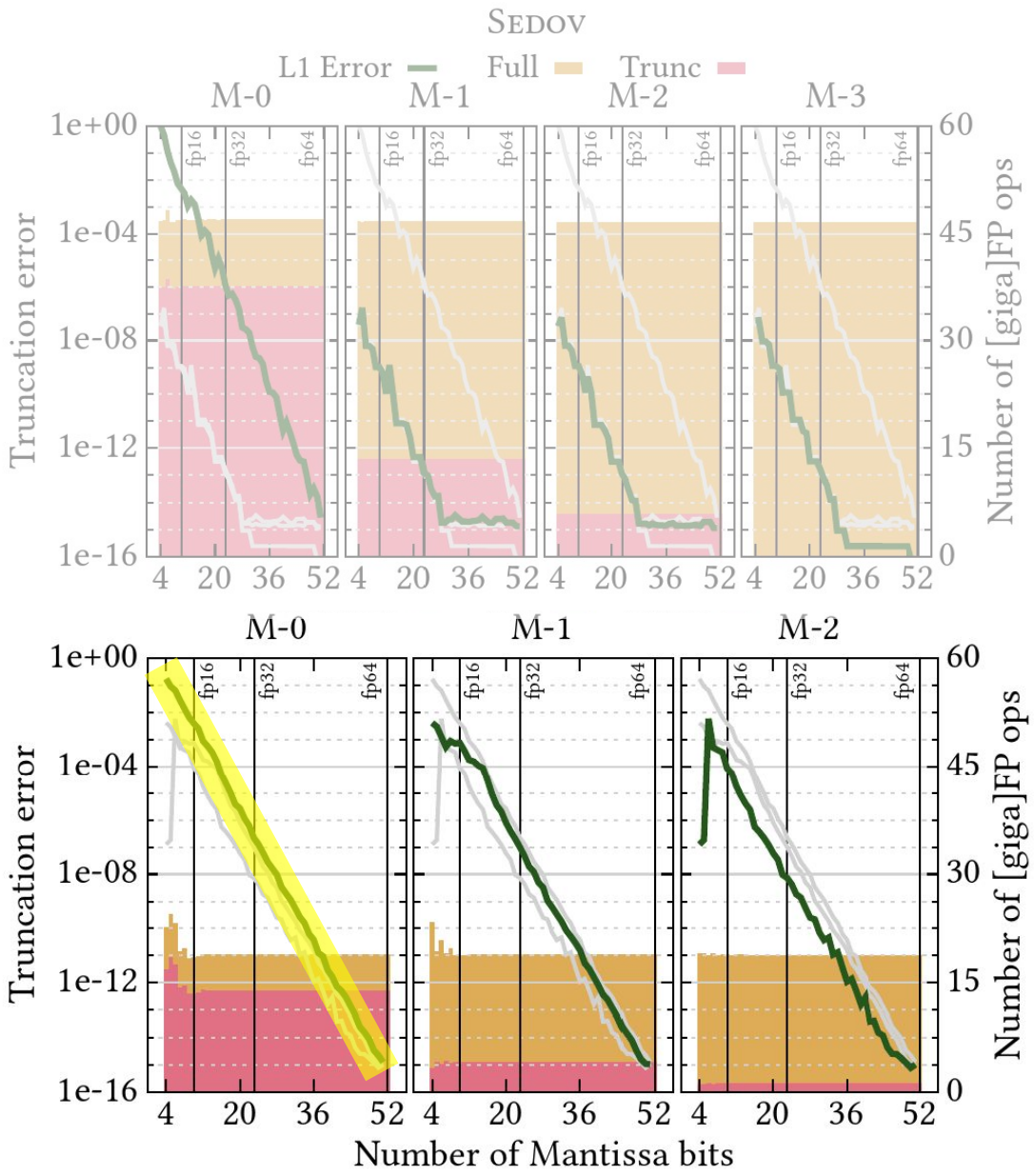


Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Sedov Blast and Sod Shock Analysis

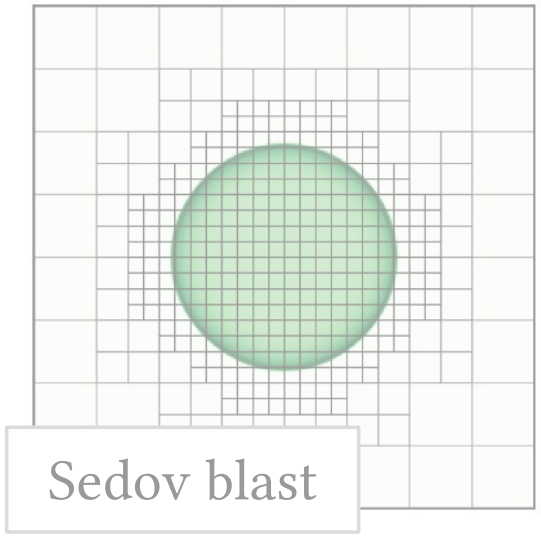


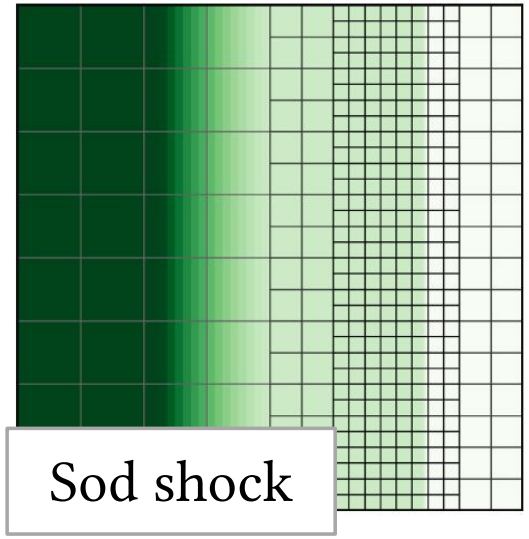


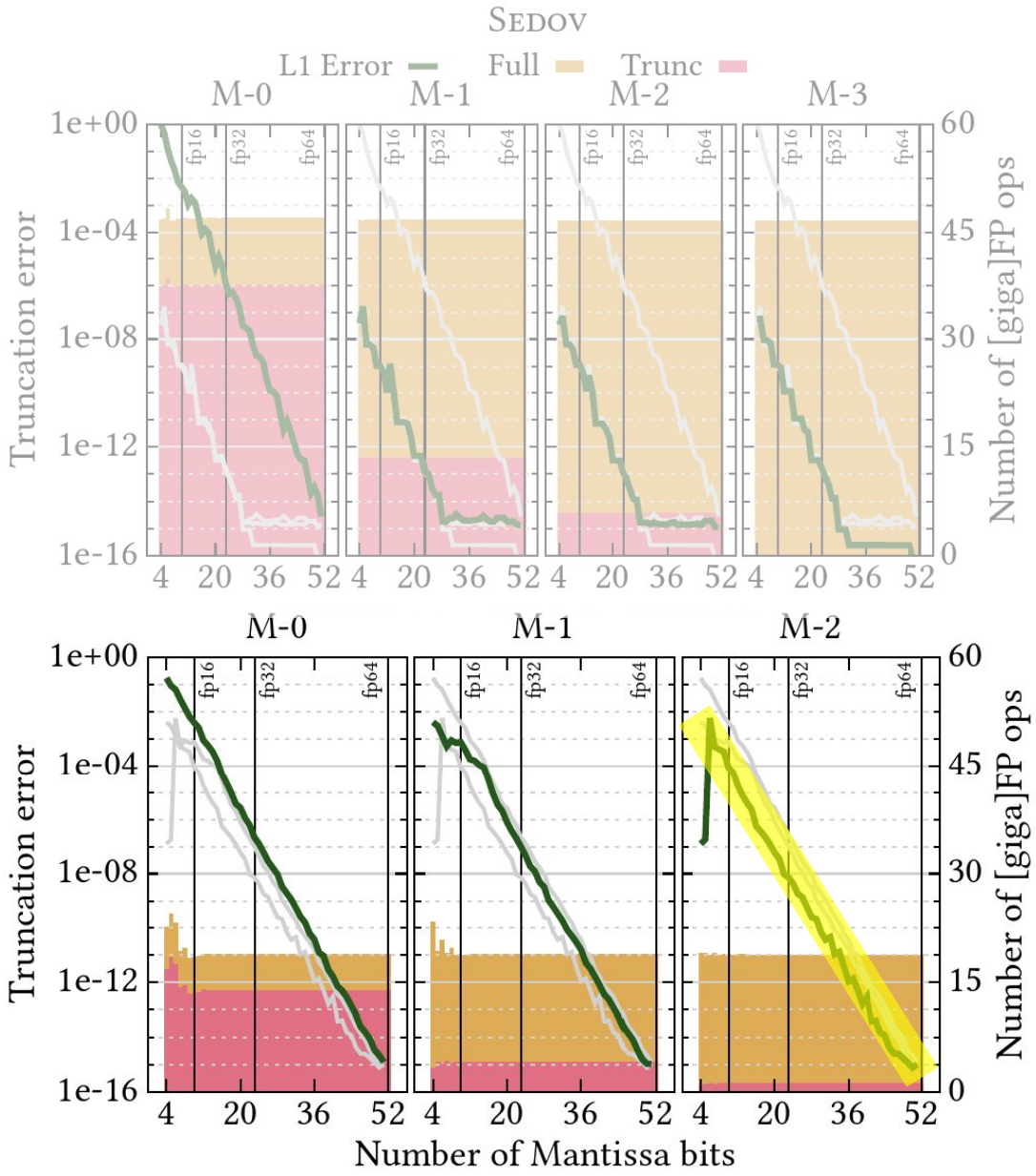


Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Sedov Blast and Sod Shock Analysis







Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

This was all just op-mode. We can do more!

RAPTOR: Architecture Overview

Op-Mode: LLVM IR Transformation

fp64 to fp32 (available in hardware)

```
fp64 foo(fp64 %a, fp64 %b, fp64 %c):
  %d = fadd %a, %b : fp64
  call @print(%d : fp64)
  %e = fmul %c, %d : fp64
  return %e : fp64
```

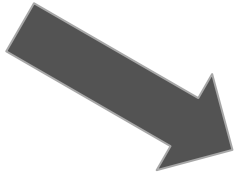
```
fp64 foo_32(fp64 %a, fp64 %b, fp64 %c):
  %a32 = ftrunc %a to fp32
  %b32 = ftrunc %b to fp32
  %d32 = fadd %a32, %b32 : fp32
  %d = fexpand %d32 to fp64
  call @print(%d : fp64)
  %d32 = ftrunc %d to fp32
  %c32 = ftrunc %c to fp32
  %e32 = fmul %c32, %d32 : fp32
  %e = fexpand %e32 to fp64
  return %e : fp64
```

Memorization mode

Store arbitrary information
alongside every value

Memorization mode

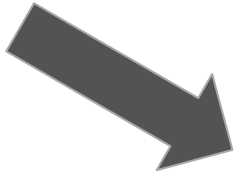
Store arbitrary information alongside every value



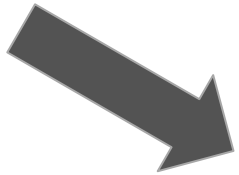
Diagnose operations individually

Memorization mode

Store arbitrary information alongside every value



Diagnose operations individually



Trace and pinpoint problems

Memorization mode

What if I had used FP64 up to here?

```
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 ...
```

→ Truncate to FP16

Memorization mode

What if I had used FP64 up to here?

```
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 ...
```

→ Truncate to FP16

Did FP16 drift?

Memorization mode

What if I had used FP64 up to here?

```
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 ...
```

→ Truncate to FP16

Did FP16 drift?

Where?

Memorization mode

What if I had used FP64 up to here?

```
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 * a5
13 a7 = a6 - a3
14 ...
```

This is an (over-)simplification.

Did FP16 drift?

Where?

Nonetheless, mem-mode is very powerful

→ It will help us learn how to use low-precision FP.

Further Ideas

Automatic diagnosis using bisection

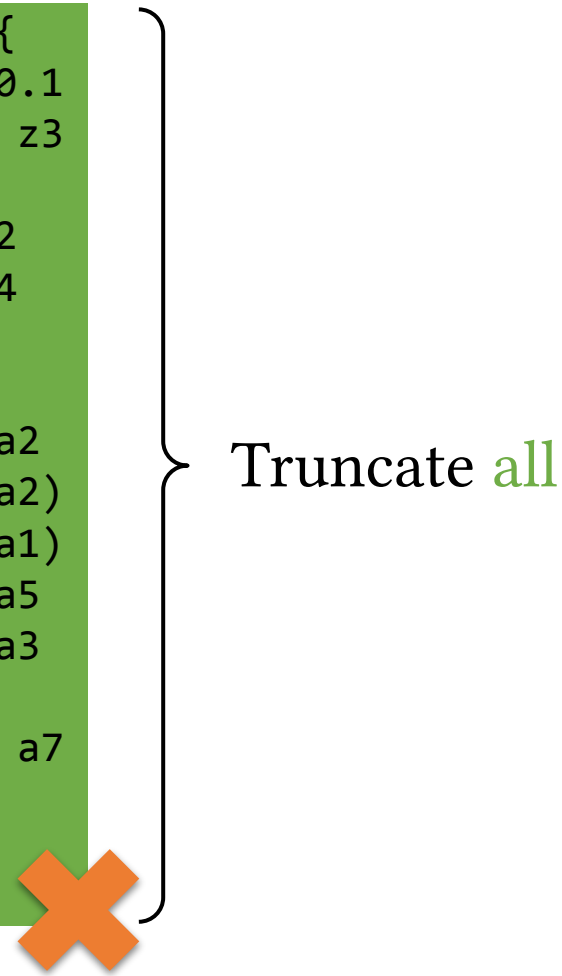
```
 1 for (...) {
 2   z = z + 0.1
 3   z4 = z / z3
 4 }
 5 a = z1 * z2
 6 b = z3 * z4
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 if (...) {
15   x = z2 / a7
16 } else {
17   ...
18 }
```

Truncate all

Further Ideas

Automatic diagnosis using bisection

```
 1 for (...) {
 2   z = z + 0.1
 3   z4 = z / z3
 4 }
 5 a = z1 * z2
 6 b = z3 * z4
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 if (...) {
15   x = z2 / a7
16 } else {
17   ...
18 }
```



Truncate all

Further Ideas

Automatic diagnosis using bisection

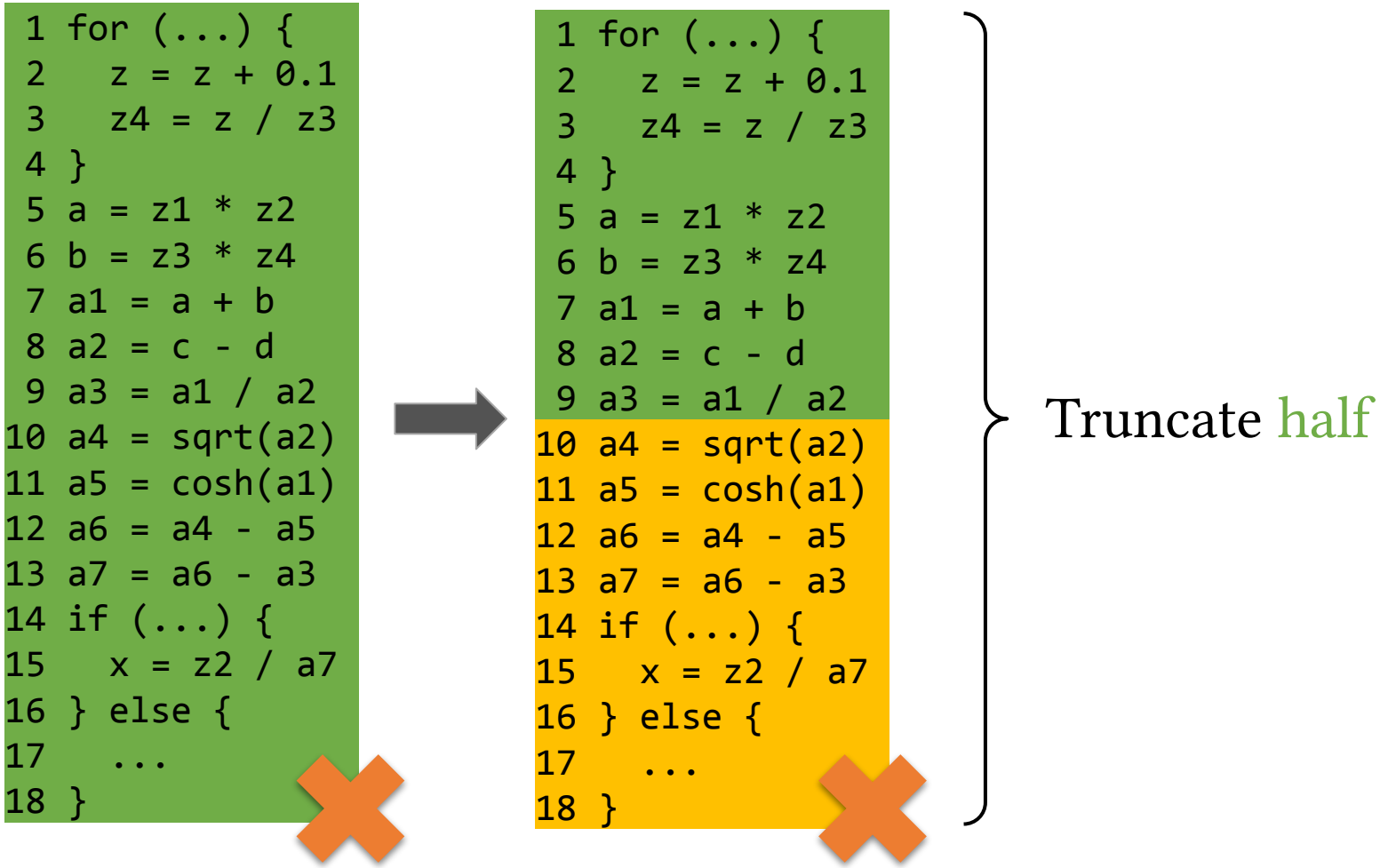
```
 1 for (...) {
 2   z = z + 0.1
 3   z4 = z / z3
 4 }
 5 a = z1 * z2
 6 b = z3 * z4
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 if (...) {
15   x = z2 / a7
16 } else {
17   ...
18 }
```

```
 1 for (...) {
 2   z = z + 0.1
 3   z4 = z / z3
 4 }
 5 a = z1 * z2
 6 b = z3 * z4
 7 a1 = a + b
 8 a2 = c - d
 9 a3 = a1 / a2
10 a4 = sqrt(a2)
11 a5 = cosh(a1)
12 a6 = a4 - a5
13 a7 = a6 - a3
14 if (...) {
15   x = z2 / a7
16 } else {
17   ...
18 }
```

Truncate half

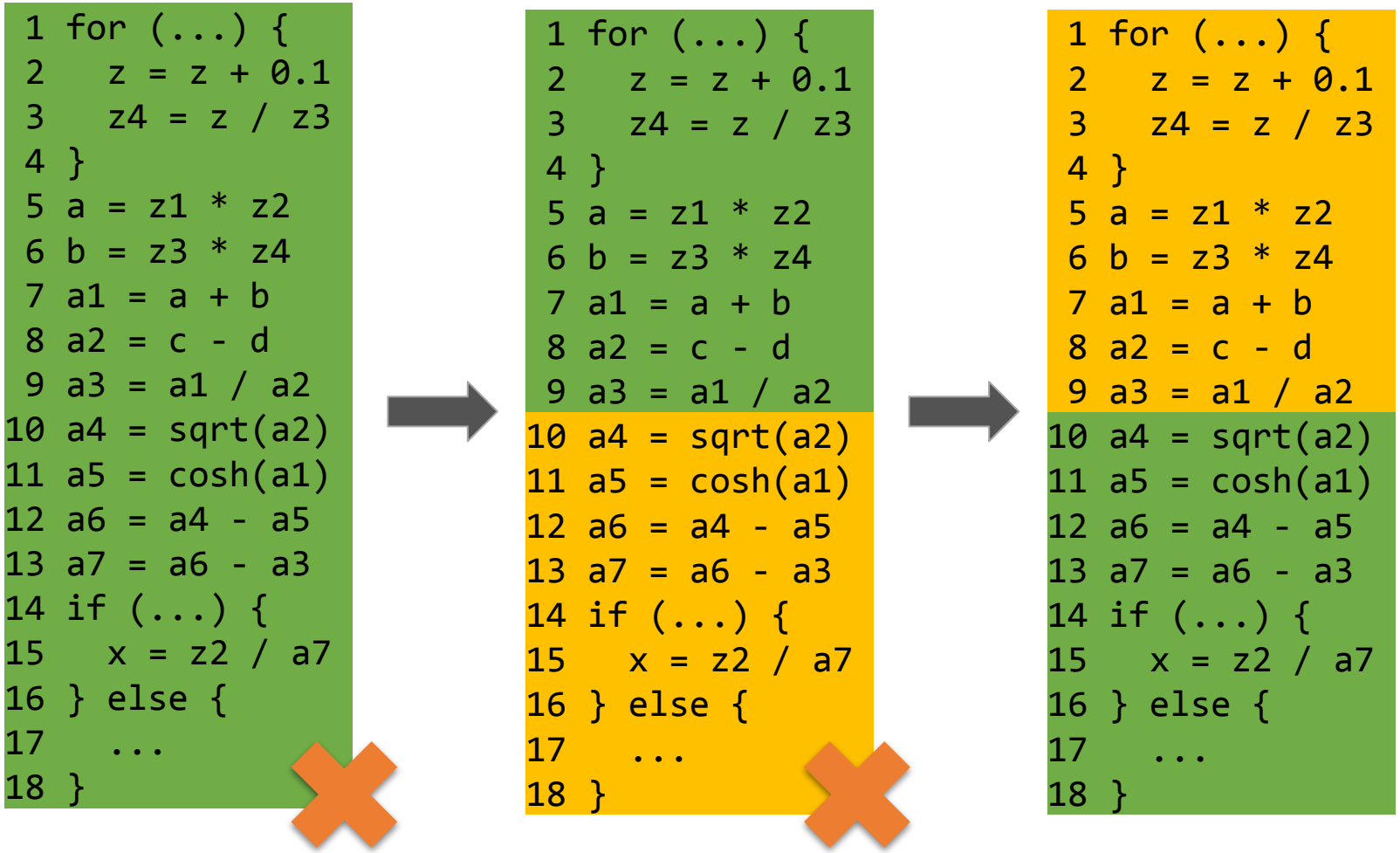
Further Ideas

Automatic diagnosis using bisection



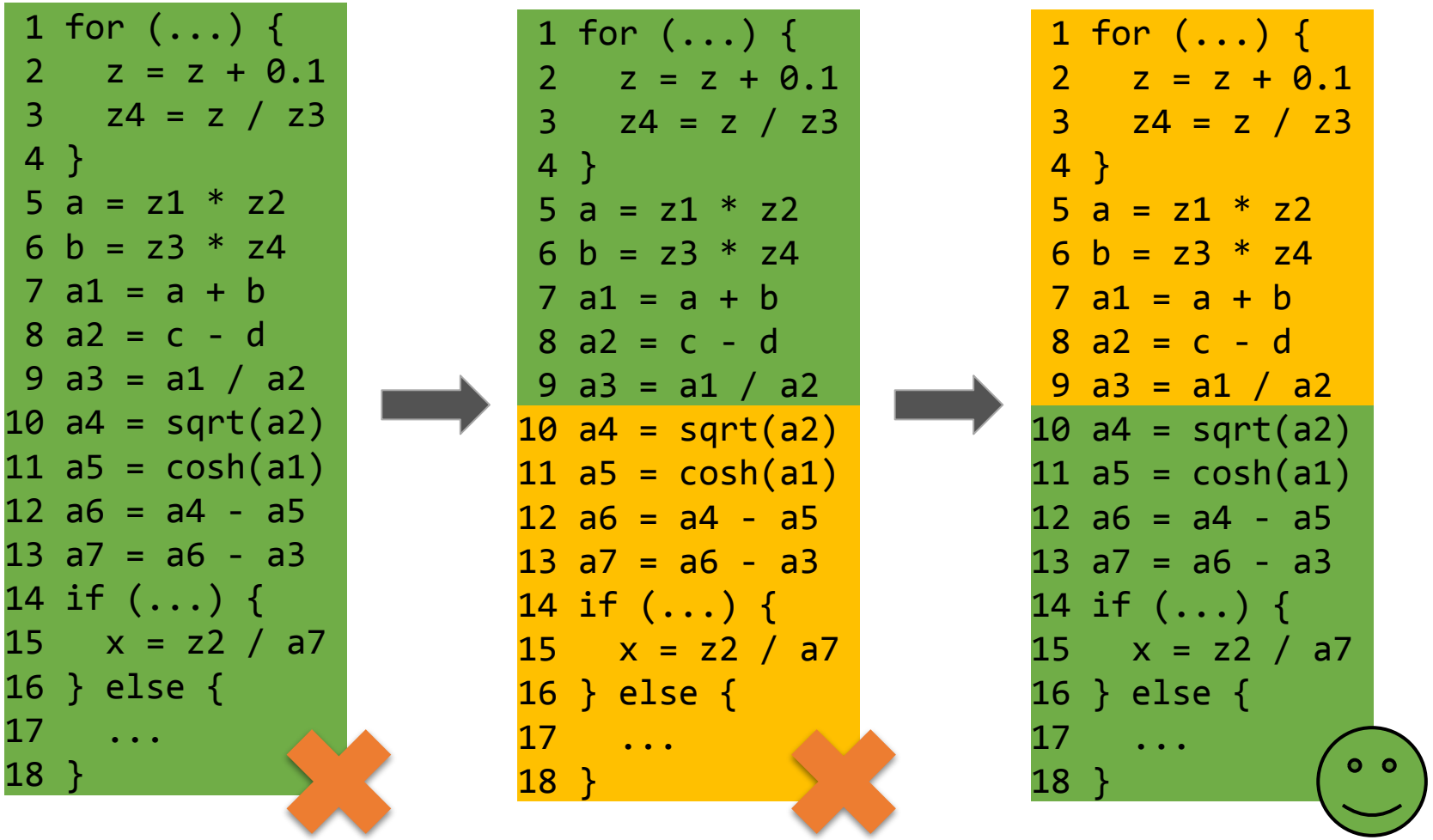
Further Ideas

Automatic diagnosis using bisection



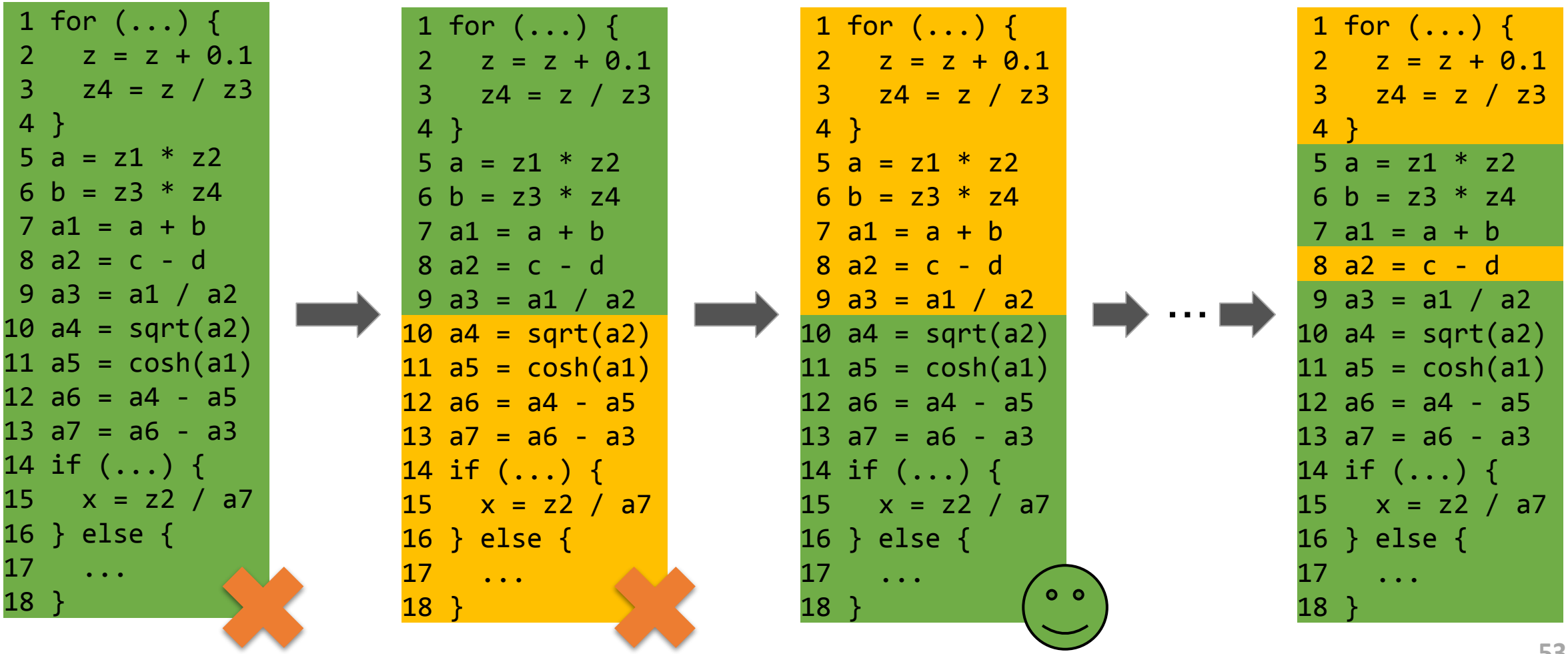
Further Ideas

Automatic diagnosis using bisection



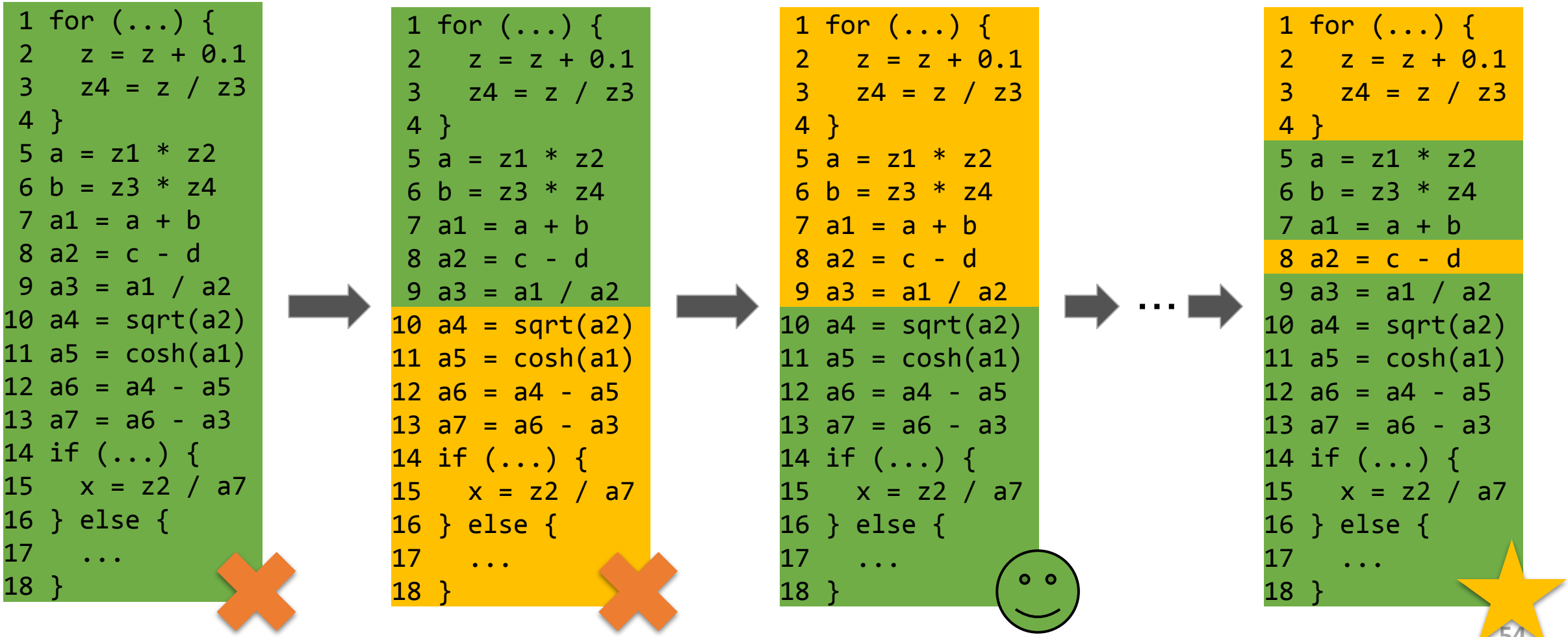
Further Ideas

Automatic diagnosis using bisection



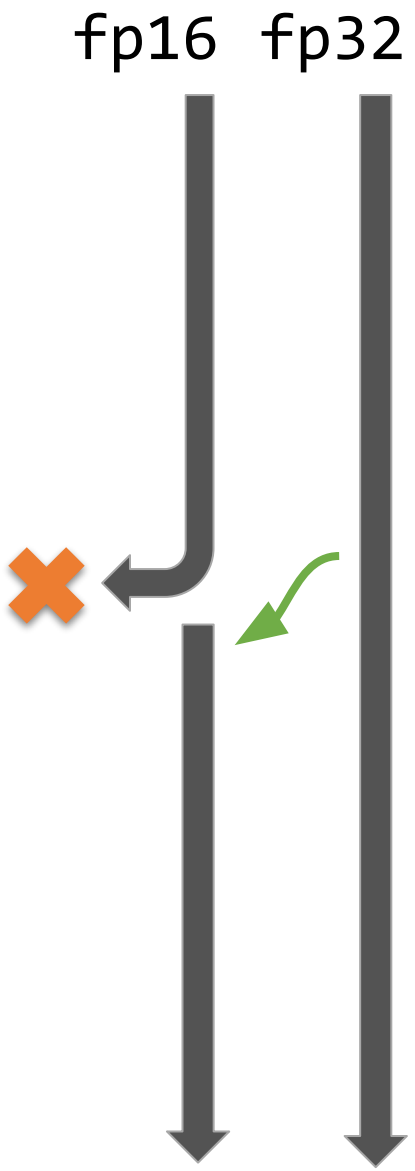
Further Ideas

Automatic diagnosis using bisection



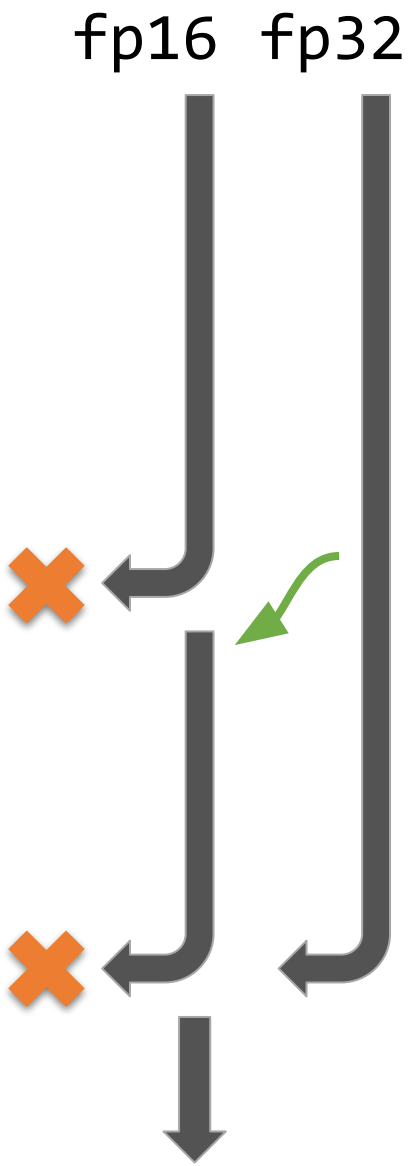
Further Ideas

Simultaneous precision streams



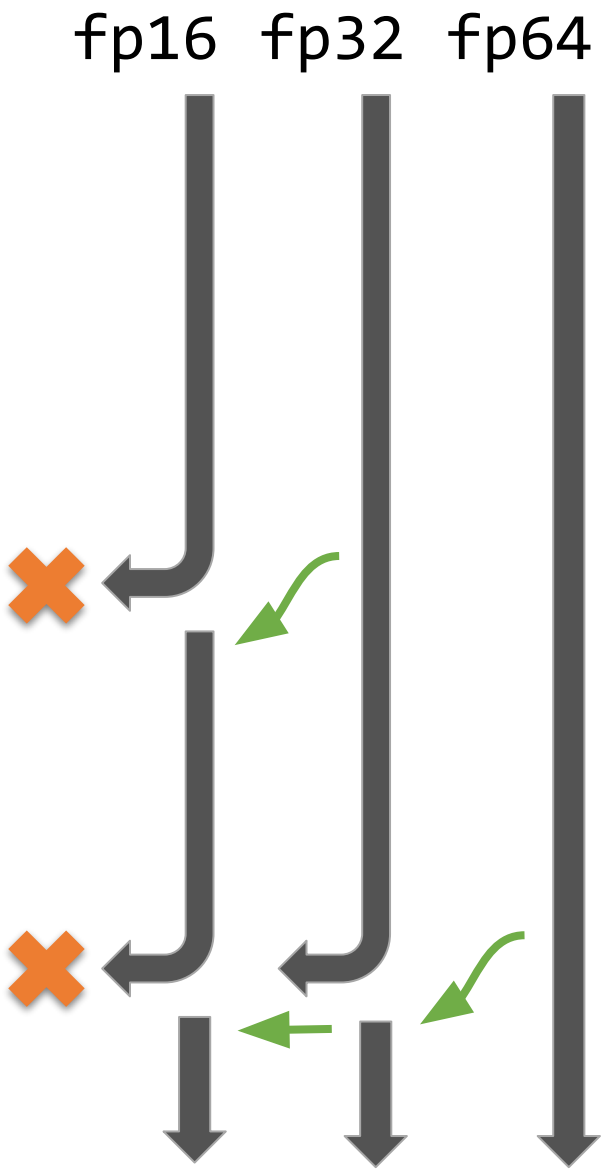
Further Ideas

Simultaneous precision streams



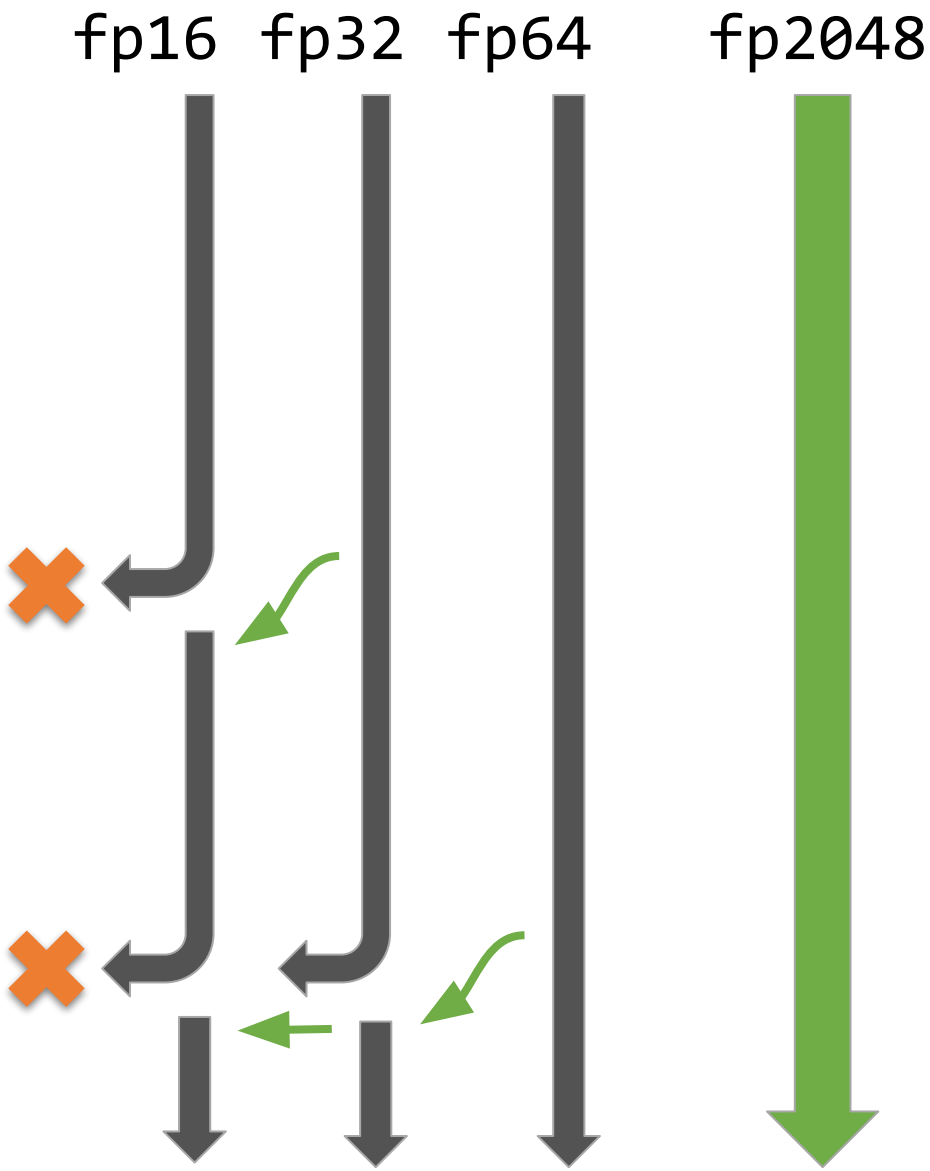
Further Ideas

Simultaneous precision streams



Further Ideas

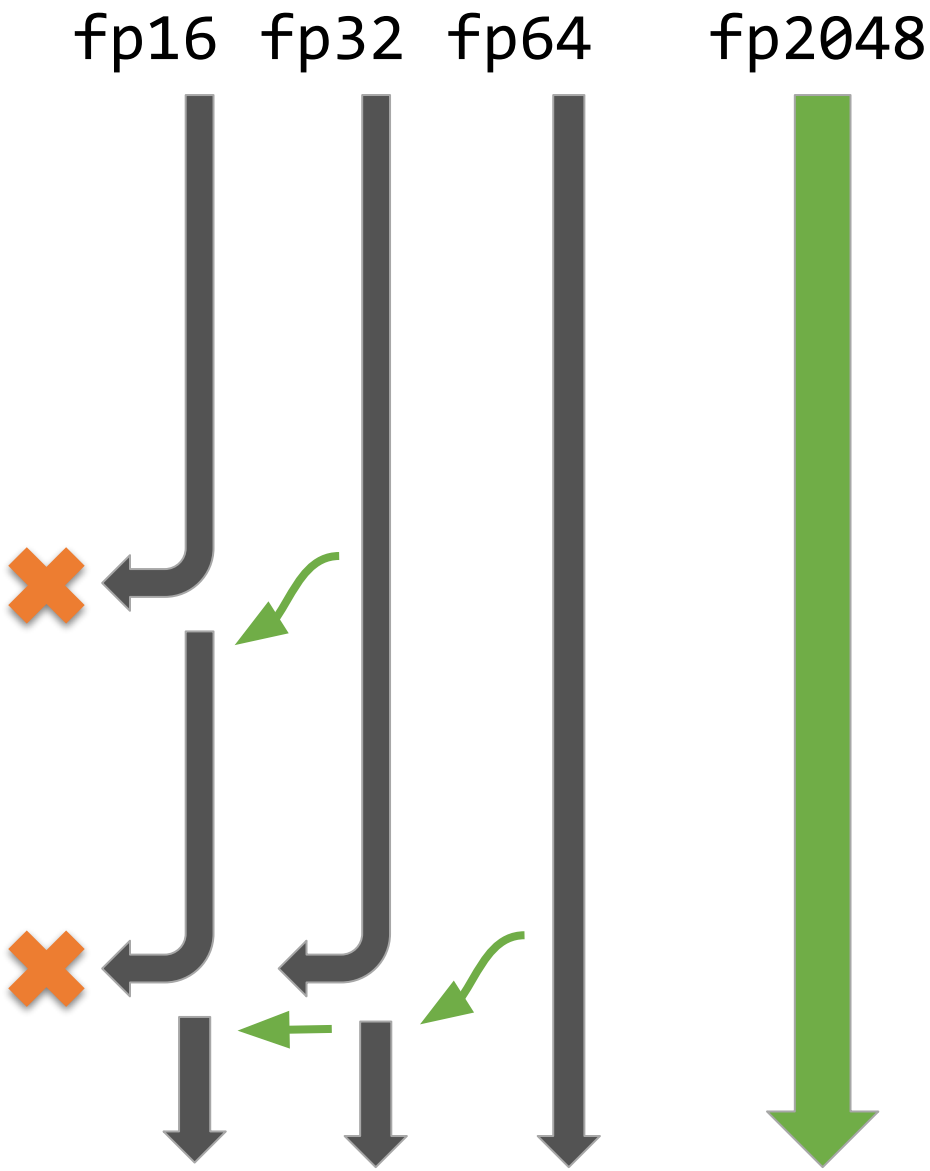
Simultaneous precision streams



Further Ideas

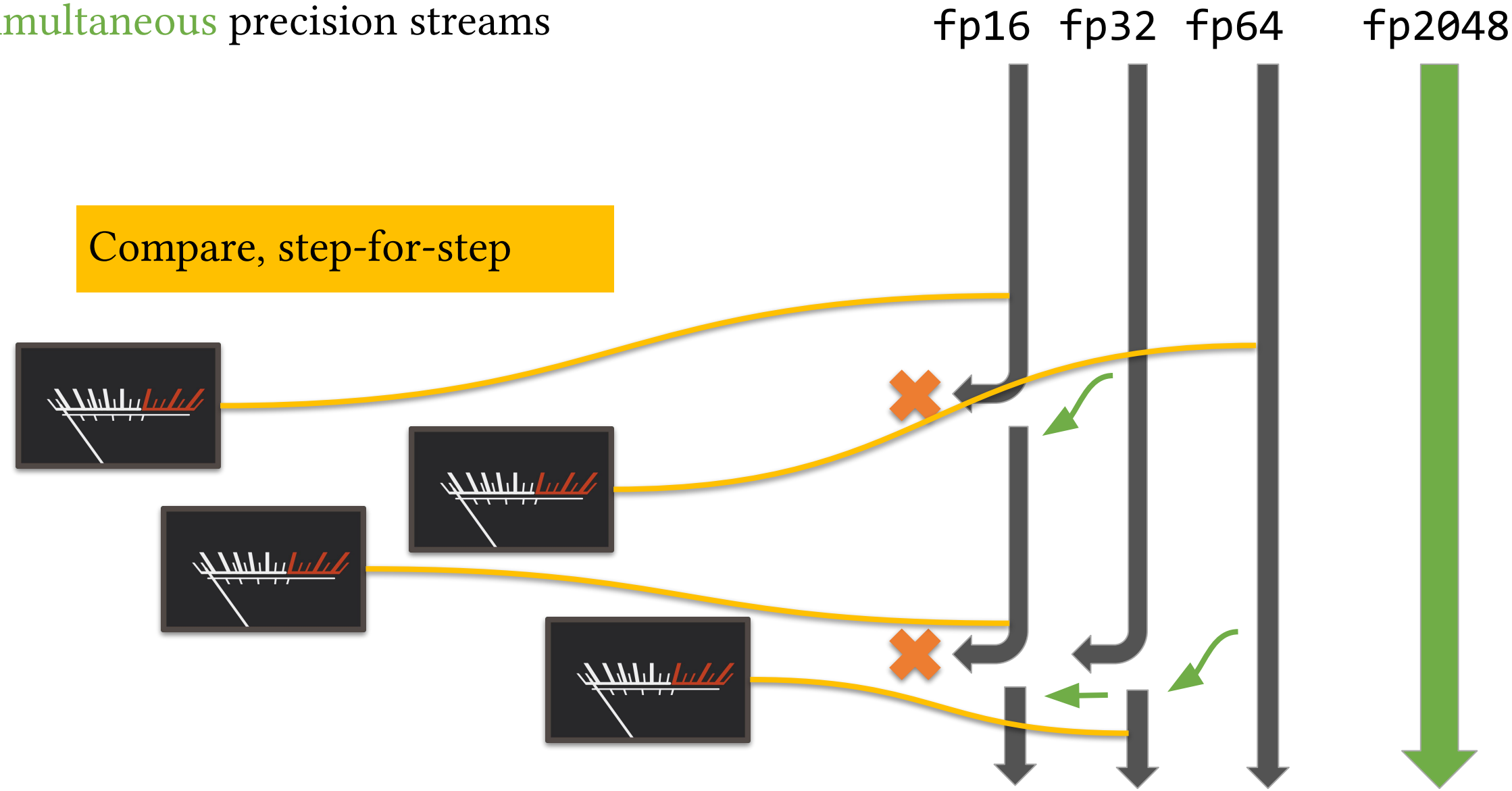
Simultaneous precision streams

Compare, step-for-step



Further Ideas

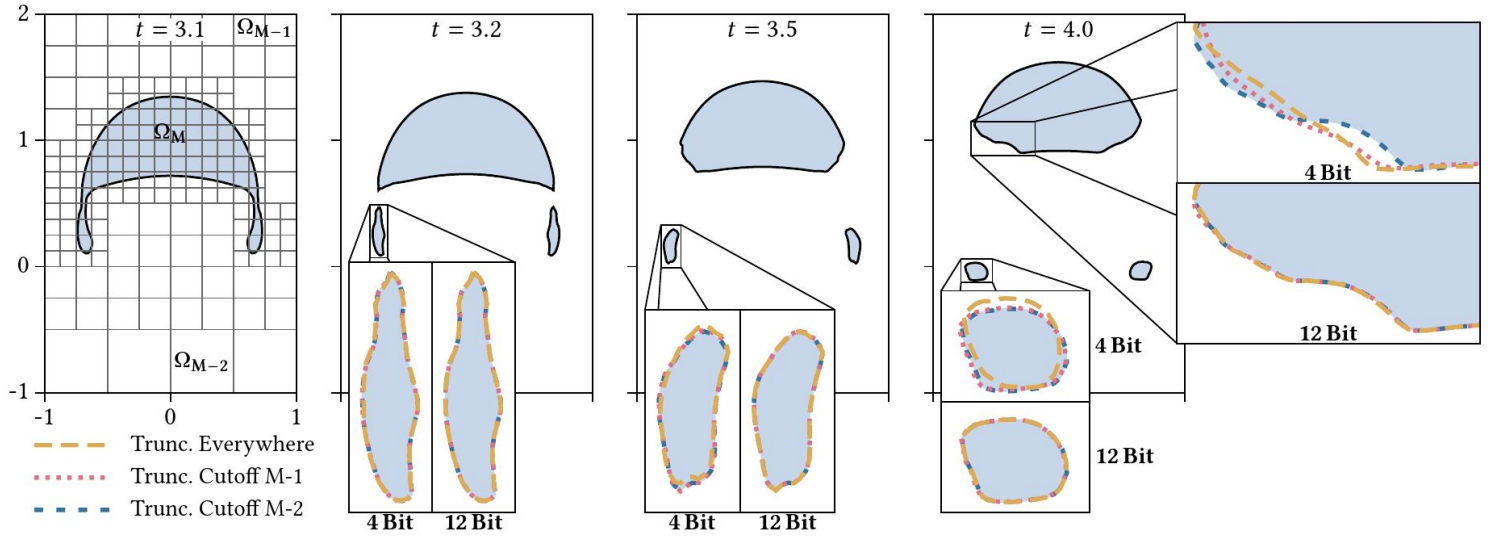
Simultaneous precision streams



Conclusion

RAPTOR - tool for numerical profiling and experimentation

- Experiment with precisions on CPU and GPU
- Scientific applications written with fp64 can tolerate lower precisions (sometimes!)



RAPTOR

github.com/RIKEN-RCCS/RAPTOR

Extras

RAPTOR's Capabilities

- Replacing existing fp operations with a specified one
  - IEEE standard formats
  - Arbitrary exponent & mantissa
- Scoping of transformation
  - Function
  - File
  - Program
- Gathering information about fp operations
  - Number
  - Errors
  - Memory

Scope	op-mode	mem-mode
Function	Fully Auto.	Semi Auto.
File	Fully Auto.	N/A
Program	Fully Auto.	N/A

Features	op-mode	mem-mode
Profile	Local	Extensive
Precision Increase	N/A	Supported

Features & Limitations

- Support for GPU (only op-mode to hardware types)
- Support for OpenMP (no reductions in mem-mode)
- Compatible with MPI (with reduction caveats in mem-mode)

Check out the paper for details!

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 32);

c = f(a, b);
...
```

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 32);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 16);

c = f(a, b);
...
```

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 32);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 16);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 1,
  /* to_exponent */ 3,
  /* to_mantissa */ 20);

c = f(a, b);
...
```

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 32);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 16);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 1,
  /* to_exponent */ 3,
  /* to_mantissa */ 20);

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 1,
  /* to_exponent */ 4,
  /* to_mantissa */ 3);

c = f(a, b);
...
```

RAPTOR Usage (Op-Mode)

```
double foo(double a, double b);
```

```
...
c = foo(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
  /* function */ f
  /* from_type */
  /* sim(1) or har
  /* to_type */ 32

c = f(a, b);
...
```

```
...
auto f = __raptor_trunc_func_op(
```

You can also specify on the command line:

```
clang ... --raptor-truncate-all="ieee(64)_sim(5,21)"
clang ... --raptor-truncate-all="ieee(64)_ieee(16)"
```

For file-scope transformation

```
..
au
  /* sim(1) or hardware(0) */ 0,
  /* to_type */ 16);

c = f(a, b);
...
```

```
r_trunc_func_op(
  foo,
/ 64,
  /* sim(1) or hardware(0) */ 1,
  /* to_exponent */ 4,
  /* to_mantissa */ 3);

c = f(a, b);
...
```

RAPTOR Usage (Mem-Mode)

```
double foo(double a, double b);
```

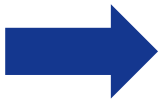
```
...
c = foo(a, b);
...
```

```
...

auto f = __raptor_trunc_func_mem(
  /* function */ foo,
  /* from_type */ 64,
  /* sim(1) or hardware(0) */ 1,
  /* to_exponent */ 3,
  /* to_mantissa */ 20);

c =
  raptor_post_convert(
    f(
      raptor_pre_convert(a),
      raptor_pre_convert(b)
    )
  );

...
```

Mem-Mode: LLVM IR Transformation & Runtime

fp64 to fp(3, 6) (unavailable in hardware)

```
fp64 foo(fp64 %a, fp64 %b, fp64 %c):
  %d = fadd %a, %b : fp64
  call @print(%d : fp64)
  %e = fmul %c, %d : fp64
  return %e : fp64
```

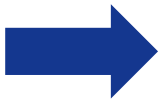


```
fp64 foo_3_6(fp64 %a, fp64 %b, fp64 %c):
  %d = call @__raptor_mem_fadd(%a, %b, 3, 6) : fp64
  call @print(%d : fp64)
  %e = call @__raptor_mem_fmul(%c, %d, 3, 6) : fp64
  return %e : fp64
```

Mem-Mode: LLVM IR Transformation & Runtime

fp64 to fp(3, 6) (unavailable in hardware)

```
fp64 foo(fp64 %a, fp64 %b, fp64 %c):
  %d = fadd %a, %b : fp64
  call @print(%d : fp64)
  %e = fmul %c, %d : fp64
  return %e : fp64
```



```
fp64 foo_3_6(fp64 %a, fp64 %b, fp64 %c):
  %d = call @__raptor_mem_fadd(%a, %b, 3, 6) : fp64
  call @print(%d : fp64)
  %e = call @__raptor_mem_fmul(%c, %d, 3, 6) : fp64
  return %e : fp64
```

RAPTOR Runtime

```
double __raptor_mem_fadd(double a, double b,
                         int e, int m) {
  mpfr_t *ma = get_raptor_val(a);
  mpfr_t *mb = get_raptor_val(b);

  pair<mpfr_t *, double> mc, c = get_new_raptor_val(e, m);

  mpfr_add(*mc, *ma, *mb);

  return c;
}

mpfr_t *get_raptor_val(double a) {
  return raptor_vals[(int)a];
}
```

Mem-Mode RAPTOR

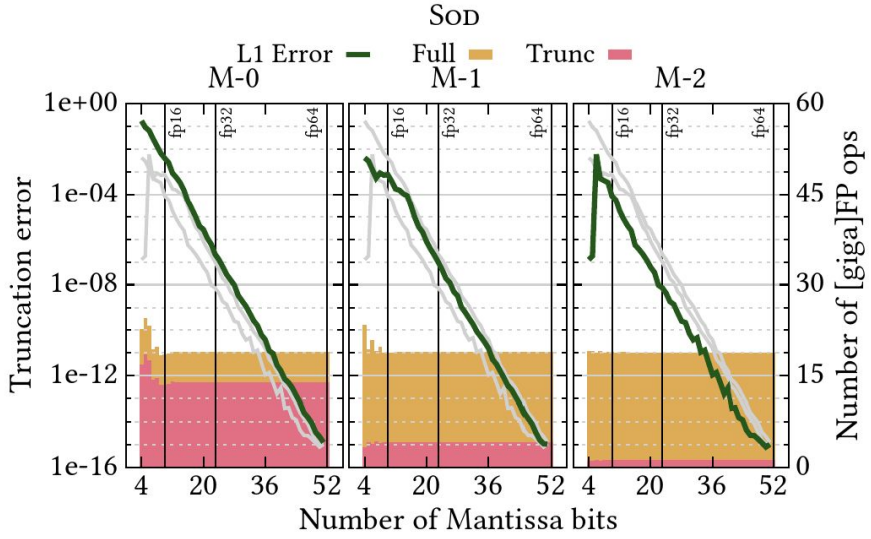
RAPTOR Runtime

```
double __raptor_mem_fadd(double a, double b,
                         int e, int m) {
  mpfr_t *ma = get_raptor_val(a);
  mpfr_t *mb = get_raptor_val(b);

  pair<mpfr_t *, double> mc, c =
    get_new_raptor_val(e, m);

  mpfr_add(*mc, *ma, *mb);

  return c;
}

mpfr_t *get_raptor_val(double a) {
  return raptor_vals[(int)a];
}
```

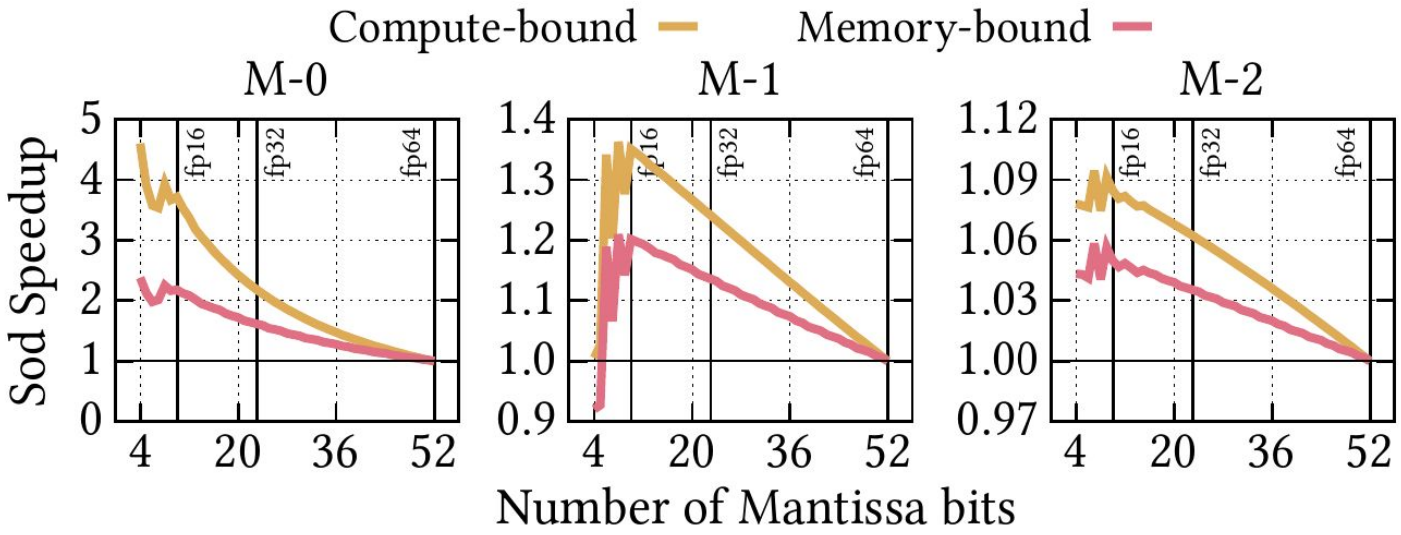
RAPTOR Runtime

```
struct __raptor_fp {
  mpfr_t m;
  double d;
}

double __raptor_mem_fadd(double a, double b,
                         int e, int m) {
  mpfr_t *ma = get_raptor_val(a);
  mpfr_t *mb = get_raptor_val(b);

  pair<mpfr_t *, double> mc, c = get_new_raptor_val(e, m);

  mpfr_add(mc->m, ma->n, mb->m);
  mc->d = ma->d + mb->d;

  check_drift(mc->d, mc->m);

  return c;
}

mpfr_t *get_raptor_val(double a) {
  return raptor_vals[(int)a];
}
```

RAPTOR Showcase: Hardware Co-Design



FP Type	GFLOP/s	Area (kGE)	Perf. density (normalized)
fp64 (11, 52)	3.17	53	1.00
fp32 (8, 23)	6.33	40	2.65
fp16 (5, 10)	12.67	29	7.30
fp8 (5, 2)	25.33	23	18.41



More power means more FLOPS, right?

1000W TBP

HPC PEAK THEORETICAL PERFORMANCE (TFLOPS)	
FP64 vector	81.7
FP32 vector	163.4
FP64 matrix	163.4
FP32 matrix	163.4

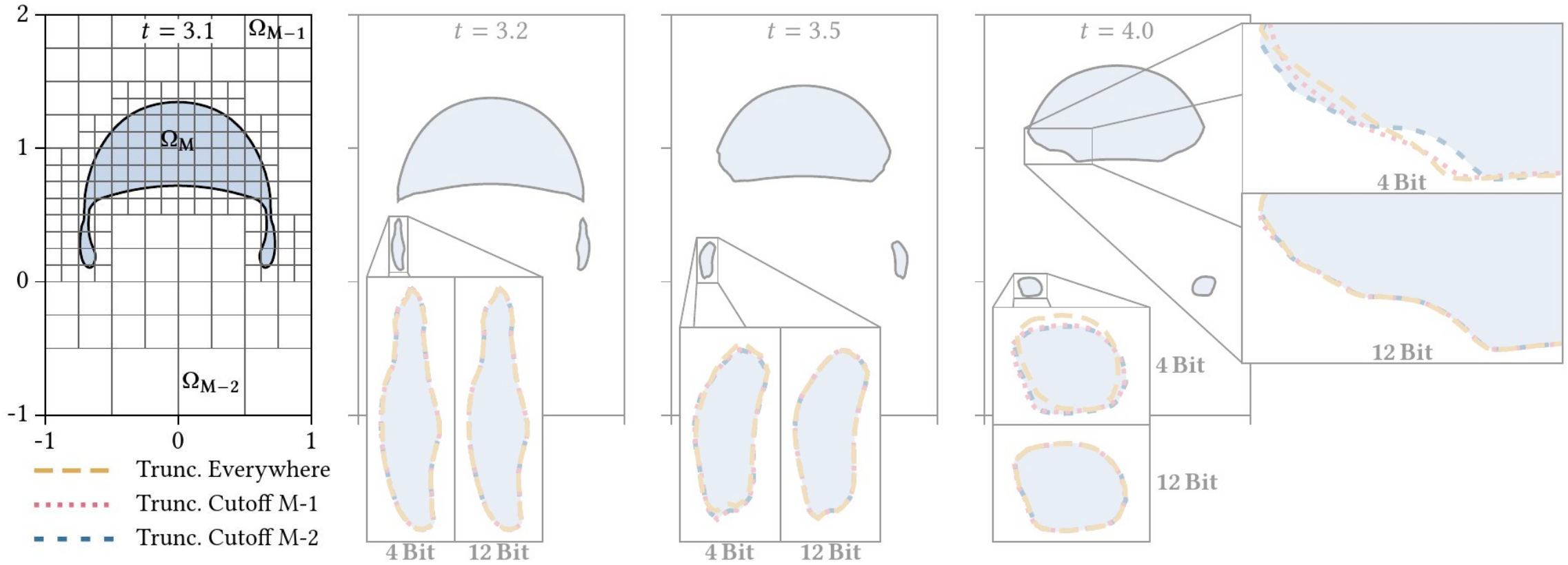
MI325X

1400W TBP

HPC PEAK THEORETICAL PERFORMANCE	
FP64 VECTOR (TFLOPS)	78.6
FP32 VECTOR (TFLOPS)	157.3
FP64 MATRIX (TFLOPS)	78.6
FP32 MATRIX (TFLOPS)	157.3

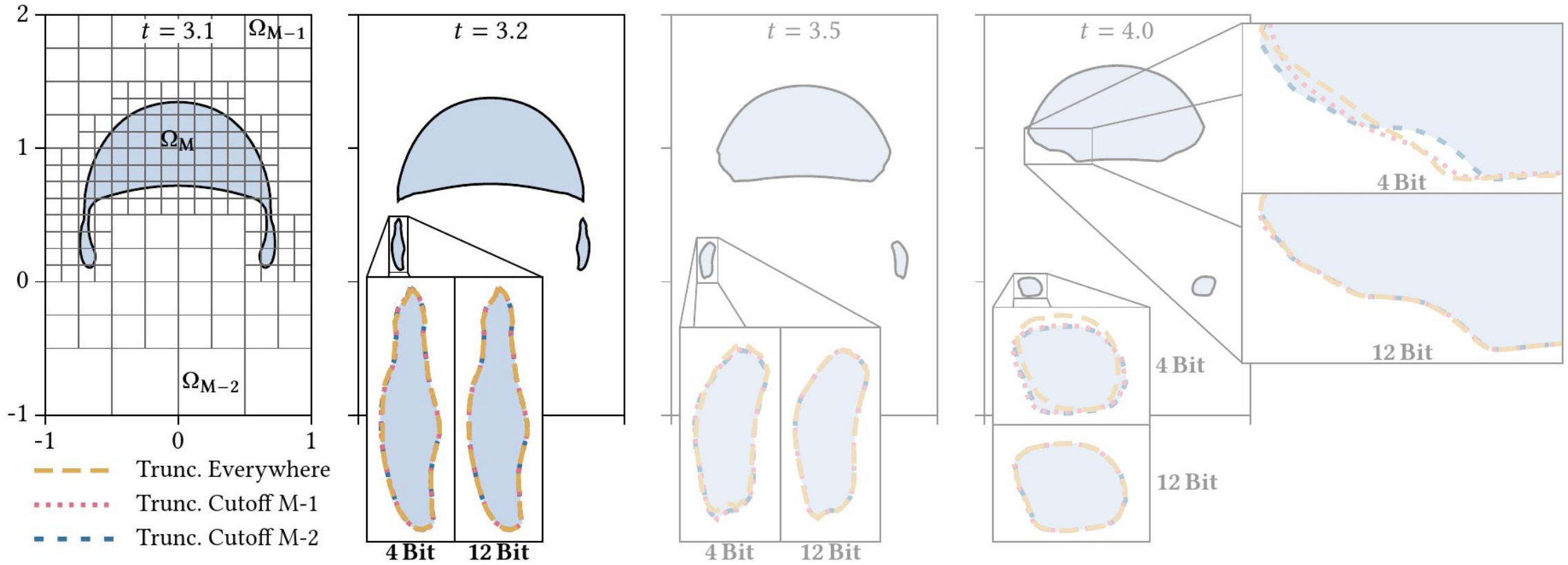
MI355X

Rising Bubble Analysis



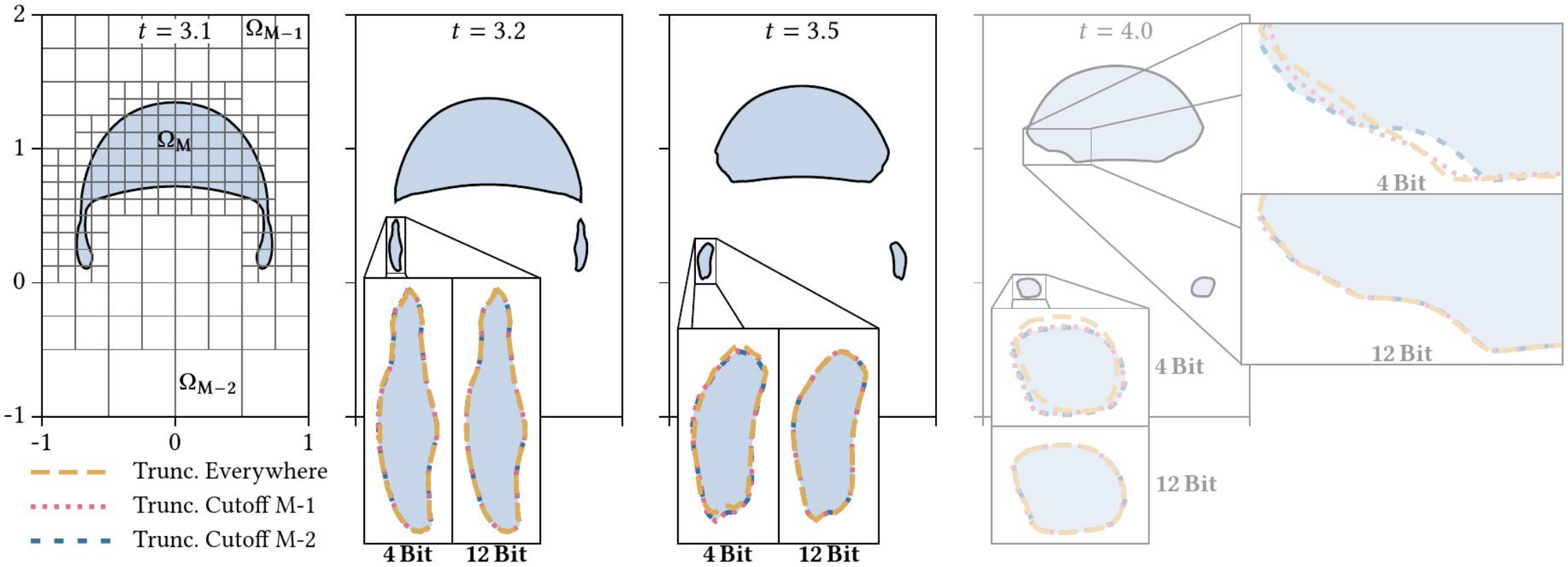
Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Rising Bubble Analysis



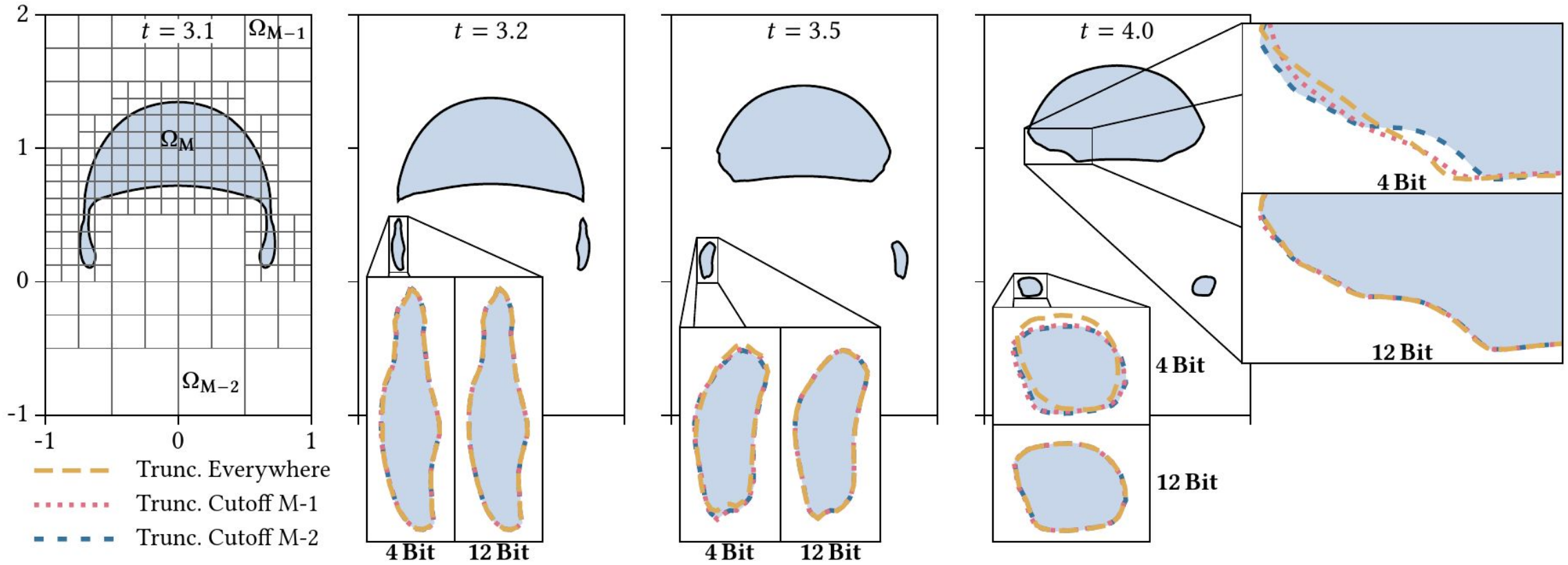
Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Rising Bubble Analysis



Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810

Rising Bubble Analysis



Hoerold, Ivanov et al. https://doi.org/10.1145/3712285.3759810