Automatic exploration of reduced floating-point representations in iterative methods

Yohan Chatelain[1,4] Eric Petit[5,4] Pablo de Oliveira Castro[1,4]
Ghislain Lartigue[3] David Defour[2,4]

[1]Université de Versailles Saint-Quentin-en-Yvelines, Li-PaRAD
[2]Université de Perpignan Via Domitia, LAMPS
[3]Normandie Université, CORIA – CNRS
[4]Exascale Computing Research, ECR
[5]Intel Corporation

Euro-Par 2019

Context

- Approximate computing use lower precision to reduce **time**, **communications' volume** and **energy** consumption
- Deep learning is driving the research in approximate computing
- Emergence of **new FP formats** that provide great speedup when combined with **custom hardware** (Google TPU [Jouppi et al., 2017], Intel's Nervanna Neural Network [Rao, 2018], IBM's SoC [Fleischer et al., 2018])

Objective: How can we harness these benefits for HPC?

State of the art

Reducing precision is a three-dimensional problem:

- **Spatial**: Finding code parts that can be rewritten into lower precision [Rubio-González et al., 2013, Lam et al., 2013, Graillat et al., 2016, Rubio-González et al., 2016]
- **Temporal**: Finding time lapses where working precision can be lowered. Works [Anzt et al., 2017, Haidar et al., 2018] based on *Iterative refinement* [Wilkinson, 1994]
- **Precision**: Finding the precision to use. Most of works focus on existing IEEE-754 formats (mixed-precision) but new formats exist (bfloat16 [LLC, 2019], DLfloat [dlf,])

Missing: a fine-grain exploration at bitwise level

Contributions

VPREC

- Simulating custom FP formats with correct rounding
- Easily testing new formats on HPC codes

Methodology to find lower precisions

- Automatic exploration on iterative scheme
- Gives lowest precision at bit level for each iteration
- Validation of the robustness to round-off errors

Validation on an industrial code (YALES2)

- Scale mixed-precision version on CRIANN cluster
- Communication's volume reduction and speedup

Example: Newton-Raphson method

k	x_{k+1}	s_k^{10}	s_k^2
0	0.0690266447076745	0.11	0.37
1	0.1230846130203958	0.21	0.70
2	0.1985746566605835	0.43	1.43
3	0.2732703639721015	0.84	2.79
4	0.3119369815109966	1.79	5.95
5	0.3181822938100336	3.40	11.3
6	0.3183098350392471	6.79	22.6
7	0.3183098861837825	13.6	45.2
8	0.3183098861837907	15.6	51.8
9	0.3183098861837907	15.6	51.8

```
double newton(double x0) {
  double x_k;
  double x_k1=x0;
  double b=PI;
  do {
    x_k  = x_k1;
    x_k1 = x_k*(2-b*x_k);
  } while (fabs((x_k1-x_k)/x_k)
                  >= 1e-15);
  return x_k1;
}
```

Table: (*left*) Convergence speed of Newton-Raphson for the computation of the inverse of π using the IEEE-754 `binary64` format (*right*). Highlighted digits in red are non significant

In the first iterations, most digits are incorrect, hinting that a low precision for the evaluation of f is enough

Question: How to find the minimal precision required to converge?

VPREC on the Newton-Raphson method

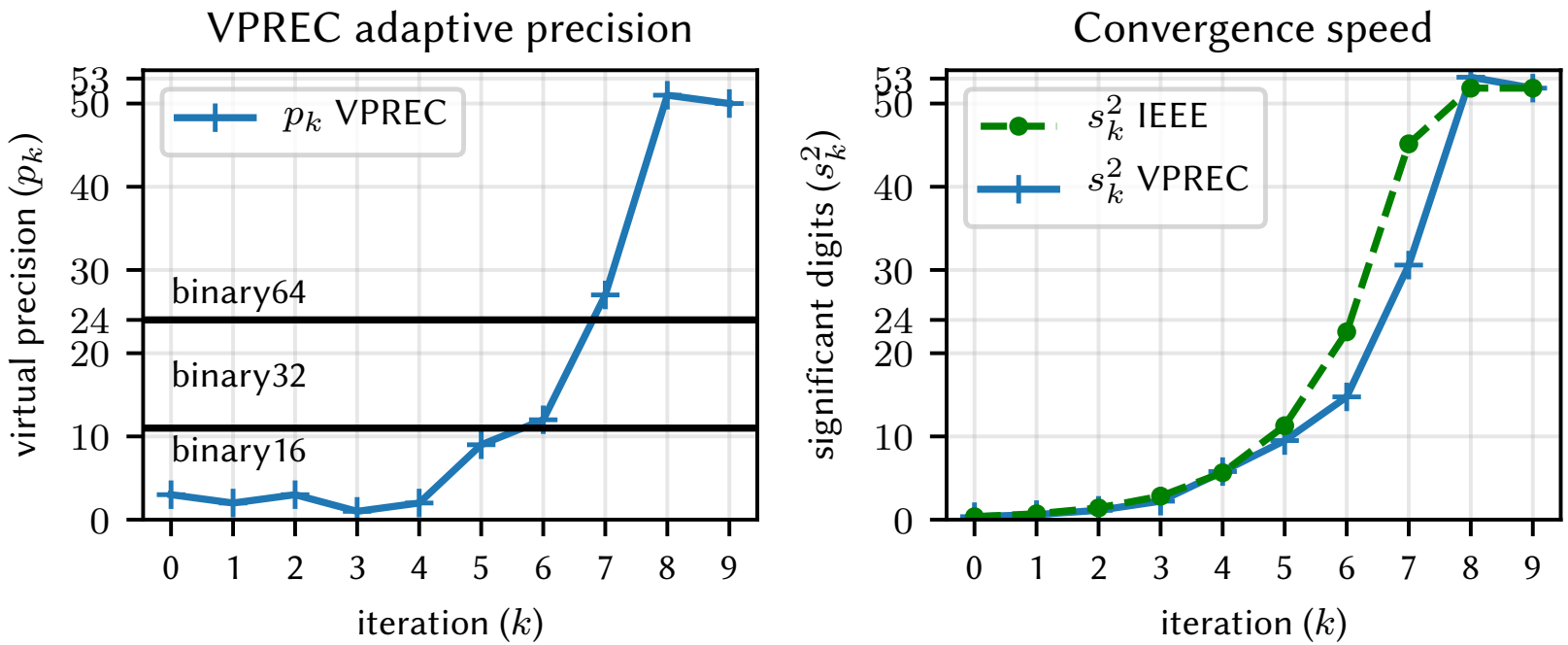
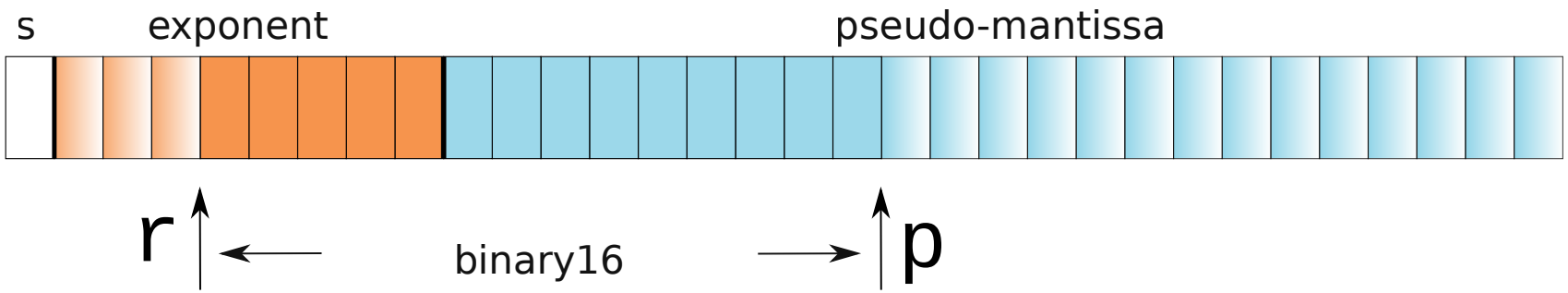


Figure: *(left)* Precision found with VPREC for Newton-Raphson. *(right)* Both the standard IEEE-754 binary64 version and the VPREC low precision configuration converge to 10^{-15} in nine iterations

Simulates any format that fits into a binary64

- Defined by couple (r, p), r=exponent size, p=pseudo-mantissa size
- Correctly *round to nearest ties to even*
- Handles `NaN`,`Inf` and denormals



By setting $r = 5$ and $p = 10$, VPREC simulates a *binary*16 embedded inside a *binary*32

VPREC rounding

Each FP operation $\bullet \in \{+, -, *, /\}$ is replaced by:

$$VPREC_{(r,p)}(x \bullet y) = \underbrace{\diamond_{(r,p)}}_{\text{OB}}(\underbrace{\diamond_{(r,p)}}_{\text{IB}}(x) \bullet \underbrace{\diamond_{(r,p)}}_{\text{IB}}(y))$$

where

$$\diamond_{(r,p)}(z) = \begin{cases} NaN_{b64} & \text{if } z = NaN_r \\ \pm\infty & \text{if } \lfloor \log |z| \rfloor > e_{max} \text{ or } z = \pm\infty \\ \pm 0 & \text{if } \lfloor \log |z| \rfloor < e_{min} - p \\ \circ_p(z) & \text{otherwise} \end{cases}$$

and

$$e_{max} = 2^{r-1} - 1, e_{min} = 1 - e_{max}$$

and

$\circ_p(x)$ is the *RoundToNearestTiesToEven* at precision p

Two modes : OutBound Precision (OB) or InBound Precision (IB)

VPREC rounding: example

Simulating `binary16` inside a `binary32`

- $r = 5, p = 10$ ($e_{max} = 15, e_{min} = -14$)
- $VPREC_{5,10}((2.903225 + 2.903225) * 256) =?$

$$
\begin{array}{rl}
 & 1.01110011100111001110000 \times 2^1 \\
+ & 1.01110011100111001110000 \times 2^1 \\
\hline
 & 1.01000010000100001000000 \\
\hline
\end{array}
$$

$$
\begin{array}{rl}
 & 1.01000010000000000000000 \times 2^2 \\
* & 1.00000000000000000000000 \times 2^8 \\
\hline
 & \\
\hline
\end{array}
$$

VPREC rounding: example

Simulating `binary16` inside a `binary32`

- $r = 5, p = 10$ ($e_{max} = 15, e_{min} = -14$)
- $VPREC_{5,10}((2.903225 + 2.903225) * 256) =?$

$$
\begin{array}{rl}
 & 1.01110011100111001110000 \times 2^{1} \\
+ & 1.01110011100111001110000 \times 2^{1} \\
\hline
\diamond_{5,10} & 1.01000010000100000010000000 \times 2^{2} \\
\hline
 & 1.0100001000 \times 2^{2}
\end{array}
$$

$$
\begin{array}{rl}
 & 1.01000010000000000000000 \times 2^{2} \\
* & 1.00000000000000000000000 \times 2^{8} \\
\hline
 & \\
\hline
\end{array}
$$

VPREC rounding: example

Simulating `binary16` inside a `binary32`

- $r = 5, p = 10$ ($e_{max} = 15, e_{min} = -14$)
- $VPREC_{5,10}((2.903225 + 2.903225) * 256) =?$

$$
\begin{array}{rl}
 & 1.01110011100111001110000 \times 2^{1} \\
+ & 1.01110011100111001110000 \times 2^{1} \\
\hline
\diamond_{5,10} & 1.01000010000100000010000000 \times 2^{2} \\
\hline
 & 1.0100001000 \times 2^{2}
\end{array}
$$

$$
\begin{array}{rl}
 & 1.01000010000000000000000 \times 2^{2} \\
* & 1.00000000000000000000000 \times 2^{8} \\
\hline
 & 1.01000010000000000000000 \\
\hline
\end{array}
$$

VPREC rounding: example

Simulating `binary16` inside a `binary32`

- $r = 5, p = 10$ ($e_{max} = 15, e_{min} = -14$)
- $VPREC_{5,10}((2.903225 + 2.903225) * 256) =?$

$$
\begin{array}{rl}
 & 1.01110011100111001110000 \times 2^{1} \\
+ & 1.01110011100111001110000 \times 2^{1} \\
\hline
\diamond_{5,10} & 1.01000010000100001000000 \times 2^{2} \\
\hline
 & 1.0100001000 \times 2^{2}
\end{array}
$$

$$
\begin{array}{rl}
 & 1.01000010000000000000000 \times 2^{2} \\
* & 1.00000000000000000000000 \times 2^{8} \\
\hline
\diamond_{5,10} & 1.01000010000000000000000 \times 2^{16} \\
\hline
 & +Inf
\end{array}
$$

Verificarlo github.com/verificarlo/verificarlo

- Verificarlo is a numerical toolbox for the floating point model:
  - **Numerical validation**: detect numerical bugs and ensure numerical accuracy during code modernization
  - **Numerical optimization**: reduce precision, harness new floating point formats and architectures
- Replaces every FP operations by a call to Verificarlo interface
- Interface allows using various backends during execution [Denis et al., 2016, Chatelain et al., 2018]
- Operates on optimized code: evaluates the impact of compiler optimizations

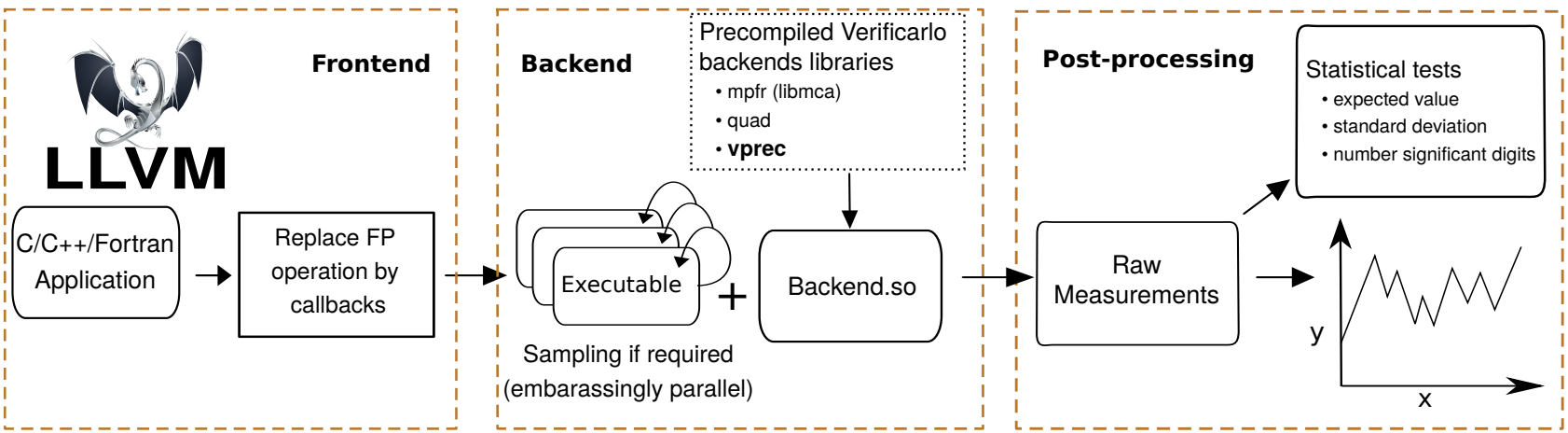


Figure: General Verificarlo exploration workflow with VPREC backend

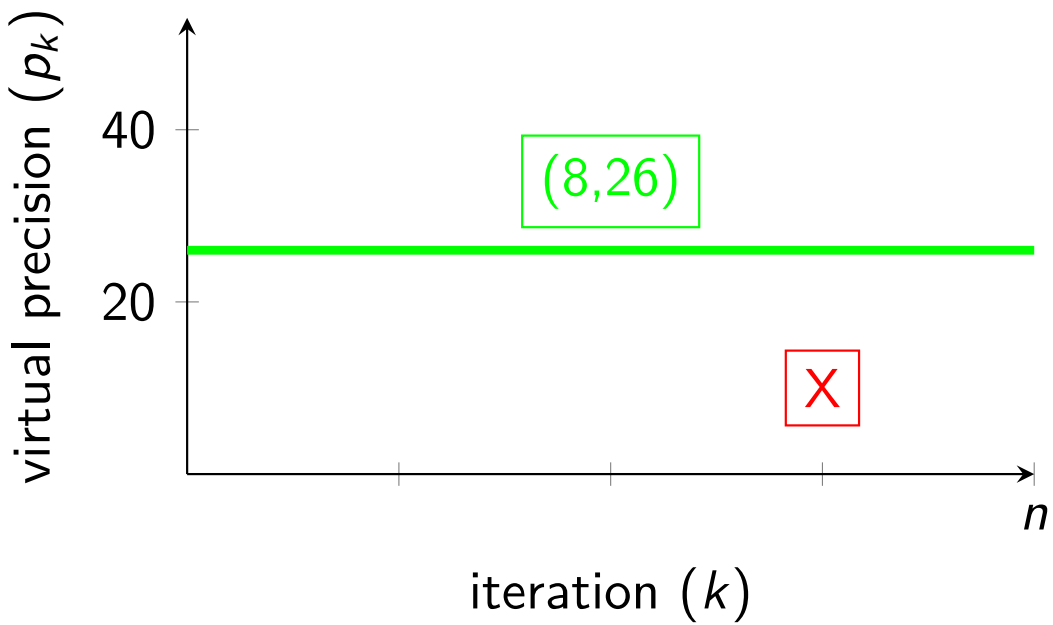
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



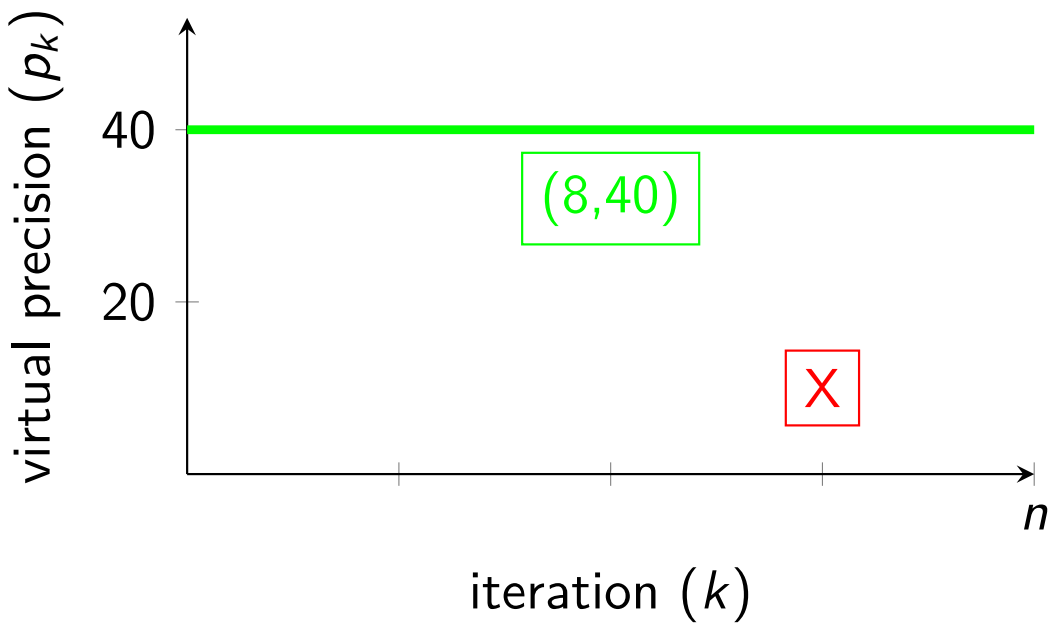
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



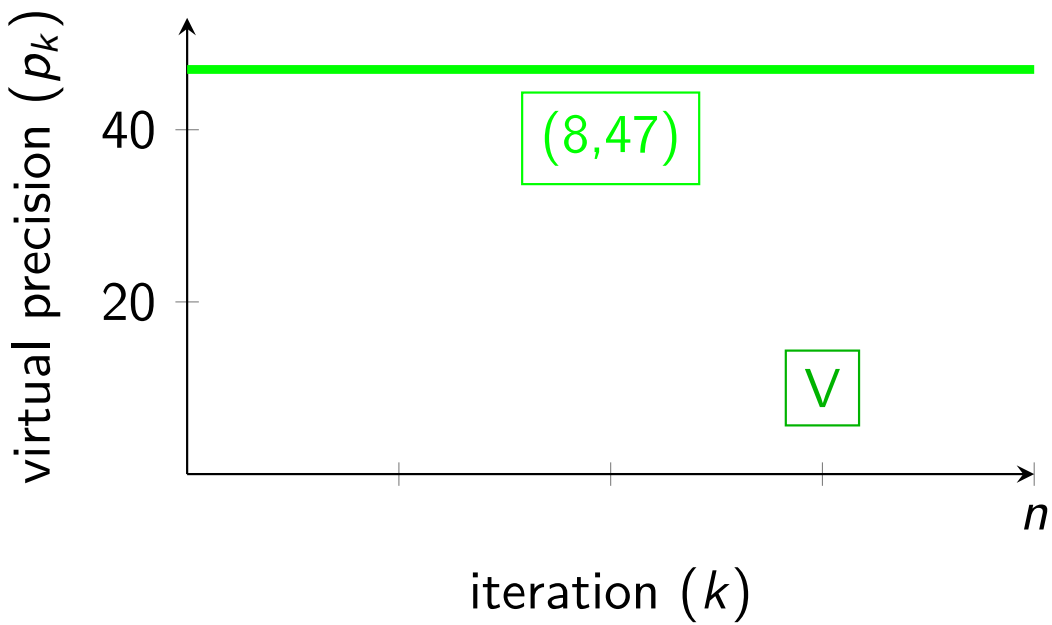
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



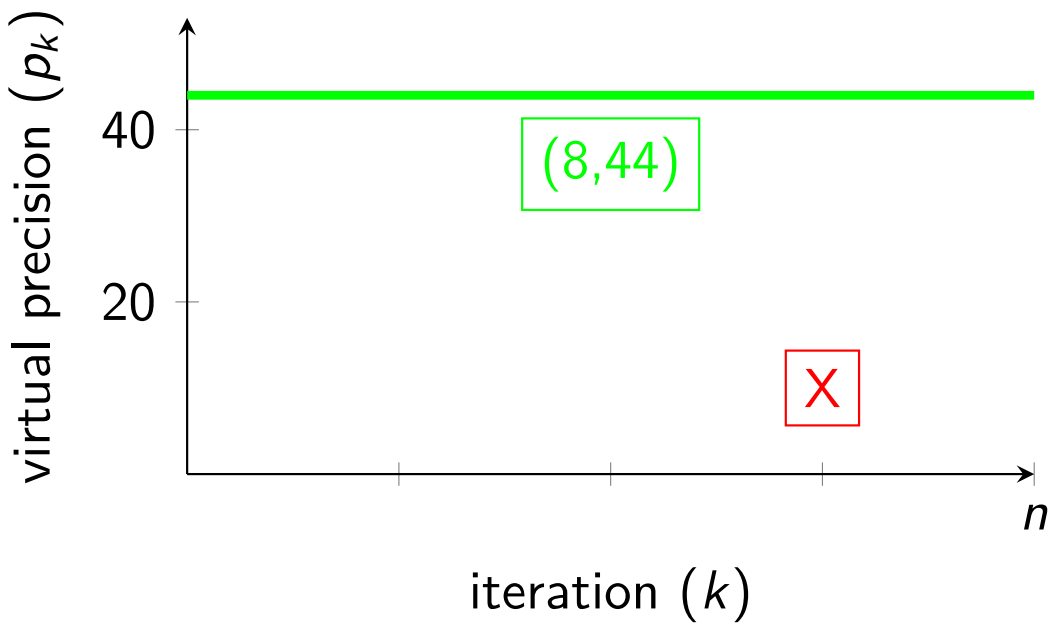
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



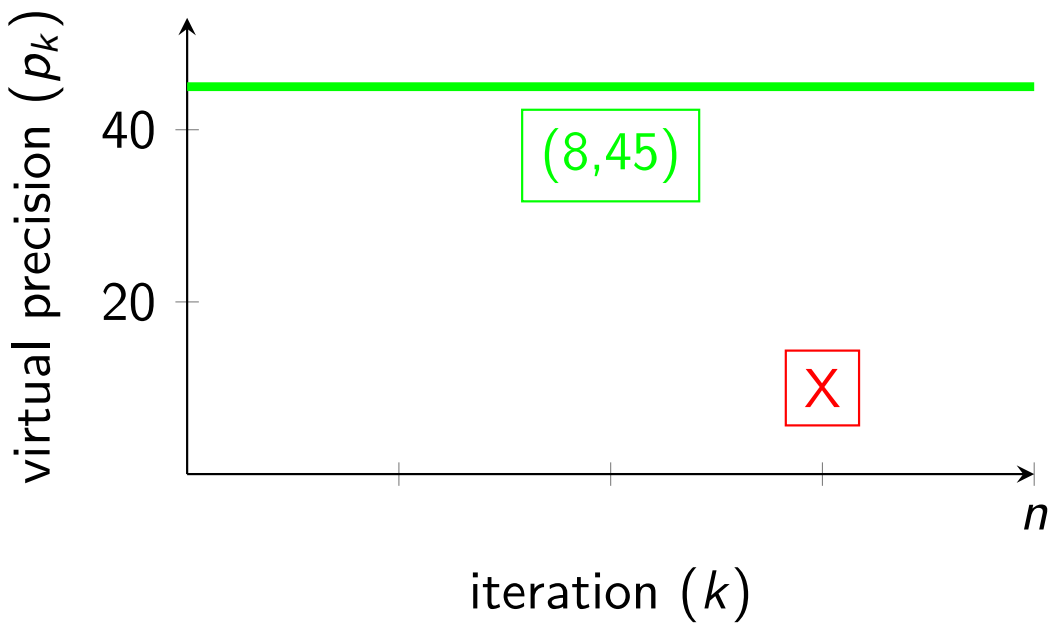
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



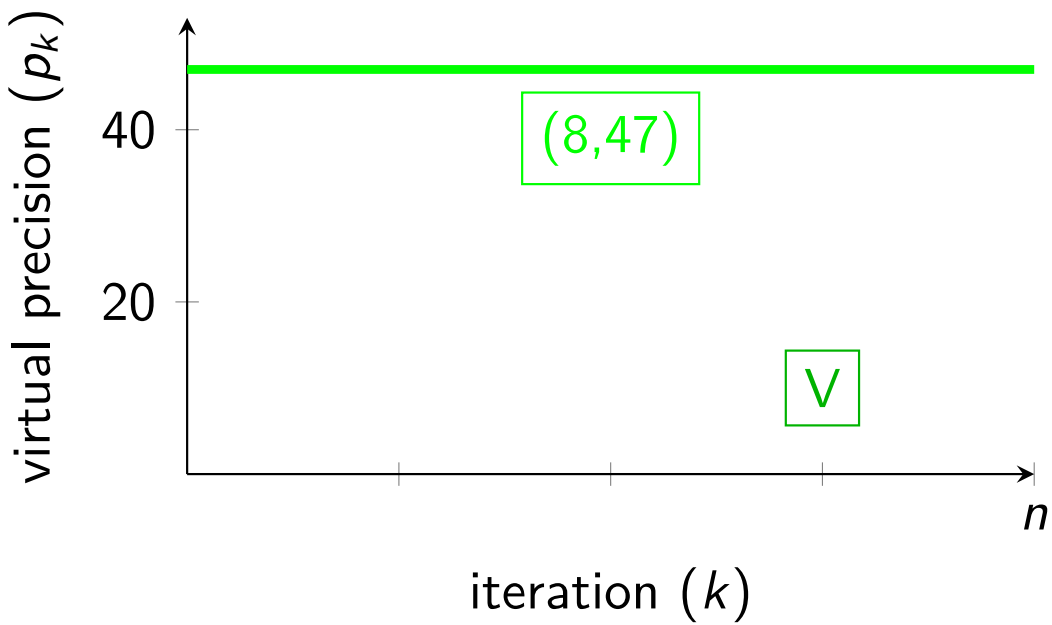
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



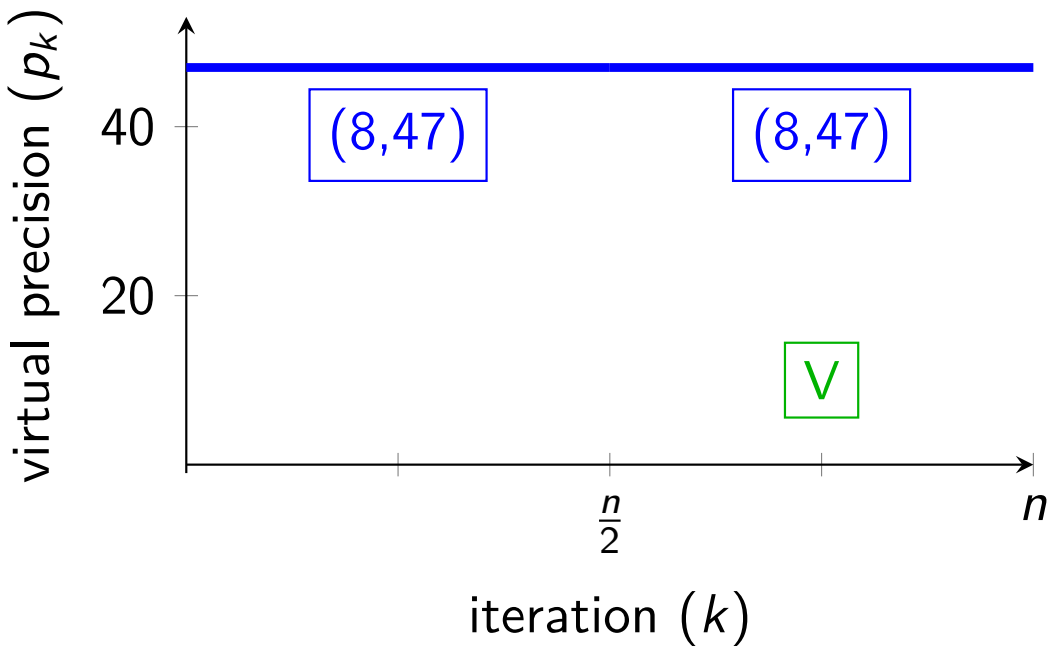
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



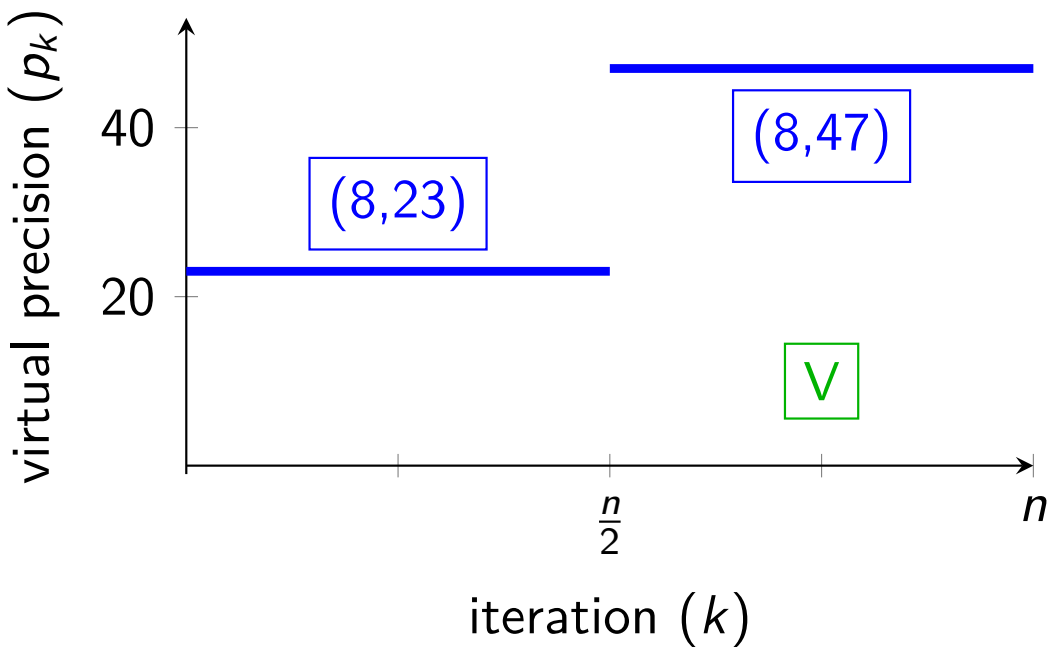
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



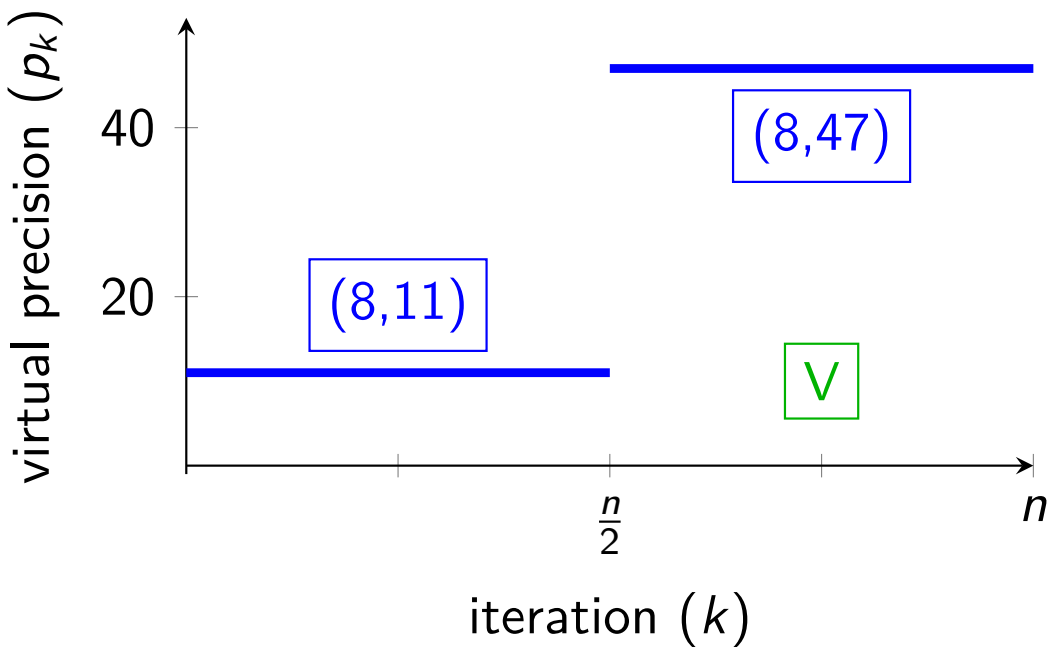
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



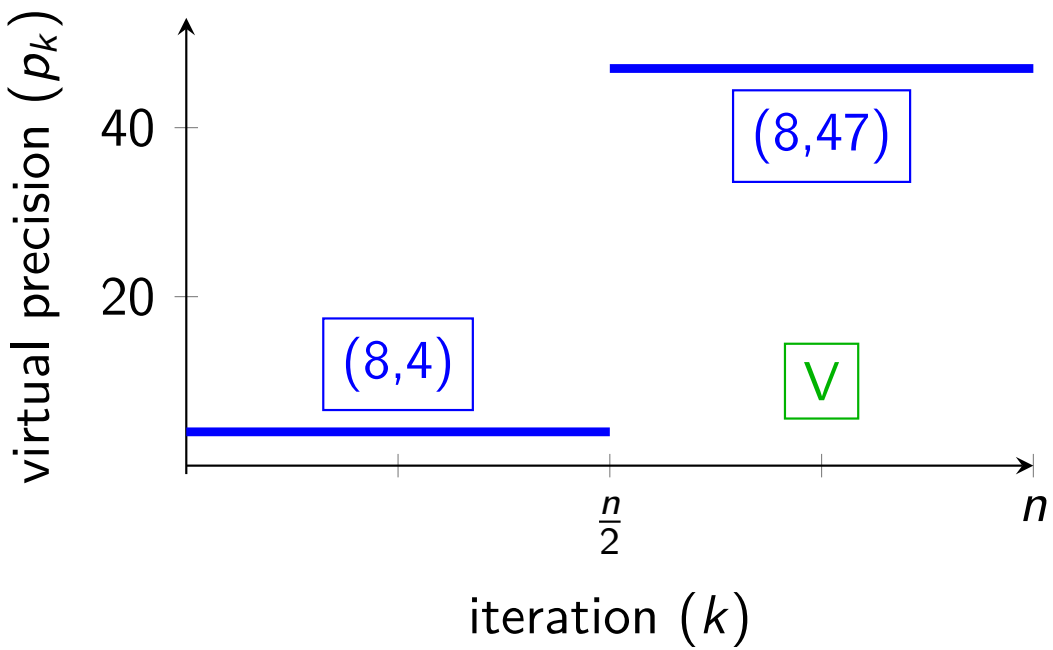
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



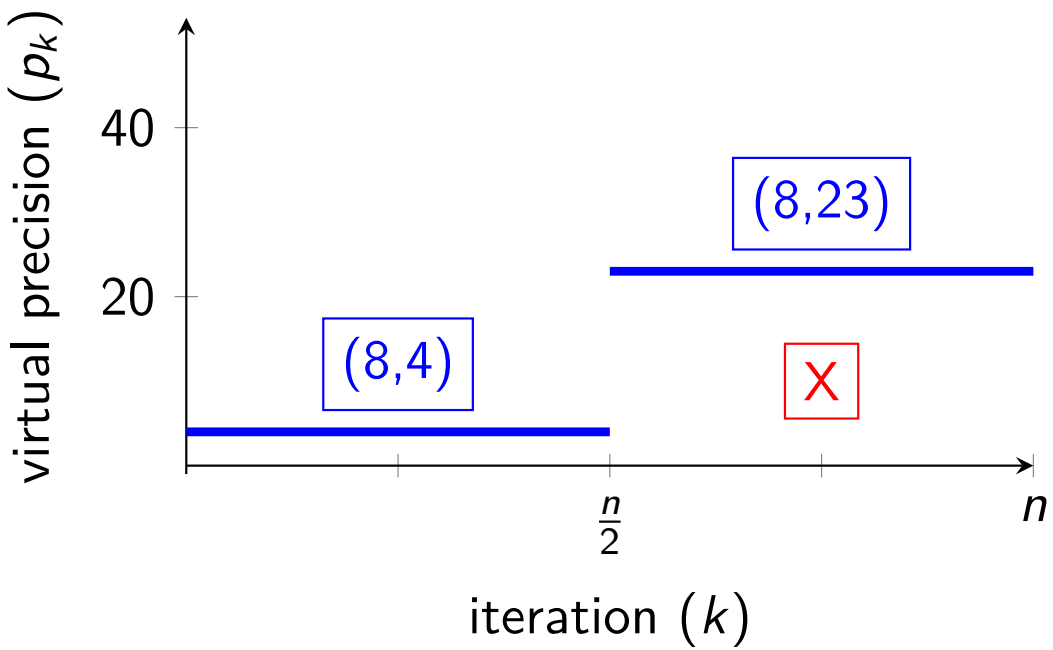
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



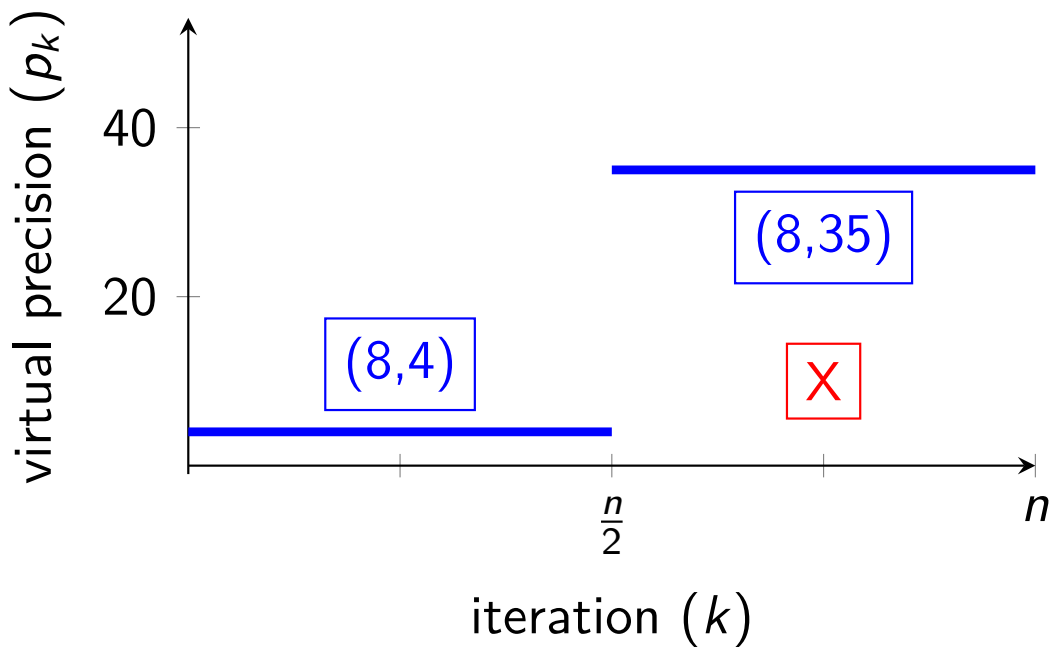
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



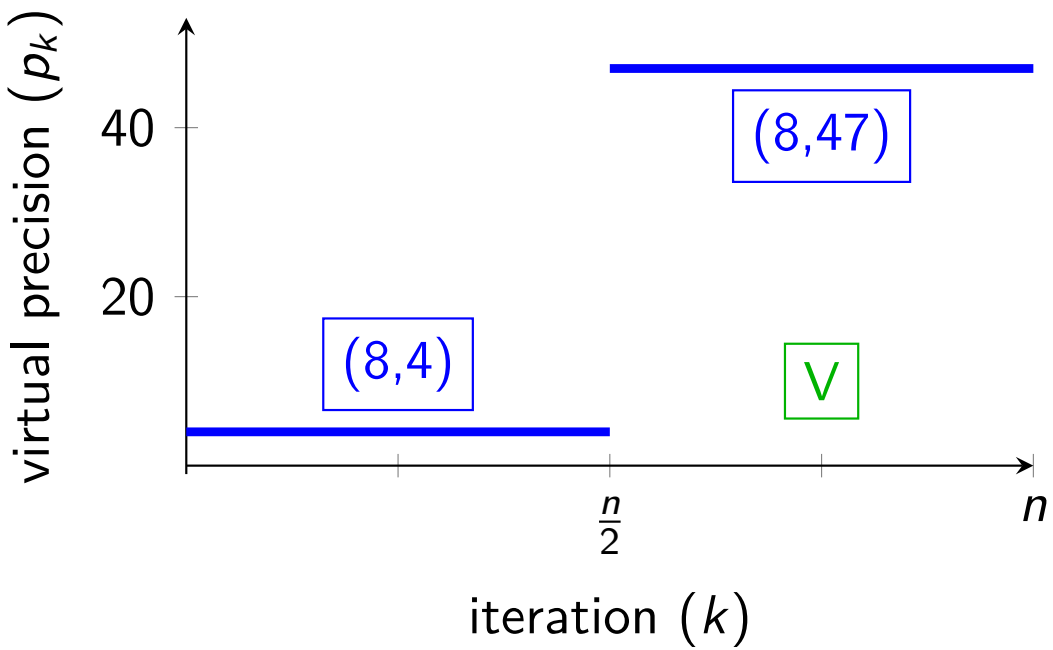
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



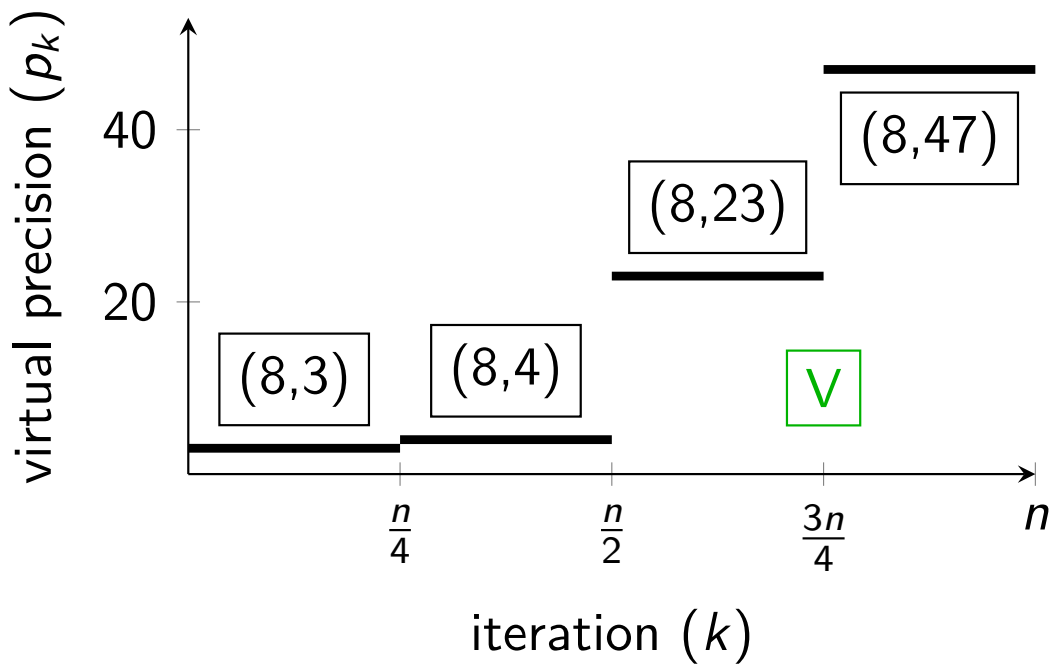
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



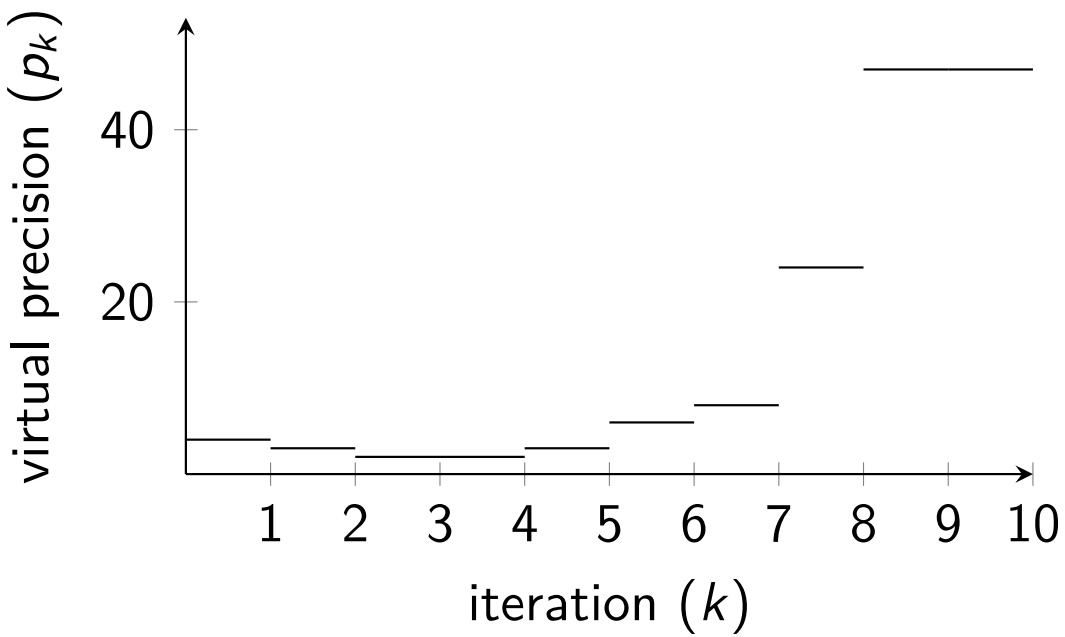
Lowest precision exploration

Objective: Find lowest valid configuration by enforcing four principles:

- Avoid combinatorial explosion ($(rp)^n$ combinations)
- Avoid configurations that quickly oscillate
- Distribute the reduction among all iterations
- Reduce more the first iterations than last iterations

Valid configuration

- Configuration $c = [(r_0, p_0), ..., (r_n, p_n)]$
- Preserves accuracy and convergence



Exploration on an industrial use case: YALES2

Coria-CNRS and Safran, Solvay, GDF-Suez, www.coria-cfd.fr

Solver for two-phase combustion

- From primary atomization to pollutant prediction

Deflated Preconditioned Conjugate Gradient

- A fine grid is geometrically projected on a coarse grid
- A CG is applied on the coarse grid and on the fine grid

Preccinsta burner use case

- Evaluation on 1.75 million mesh elements
- Exploring reduction on *entire application* and *deflated grid*

Deflated Preconditioned Conjugate Gradient

A-DEF2 using preconditioner M^{-1} and deflation matrix W.

- In blue the coarse grid and green the fine grid
- deflated part = blue, entire application = green + blue

Require : A, b, M^{-1}, W
initialization
for k = 0 ,1 ,... until required convergence **do**

$$\alpha_{k+1} = \frac{r_k^T w_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_{k+1} p_k$$

$$r_{k+1} = r_k - \alpha_{k+1} A p_k$$

Solve $\hat{A} d_{k+1} = W^T (A M^{-1} - I) r_{k+1}$

$$w_{k+1} = M^{-1} r_{k+1} - W d_{k+1}$$

$$\beta_{k+1} = \frac{r_k^T w_{k+1}}{r_k^T w_k}$$

$$p_{k+1} = w_{k+1} + \beta_{k+1} p_k$$

end for

Piecewise constant exploration on *preccinsta 1.75M*

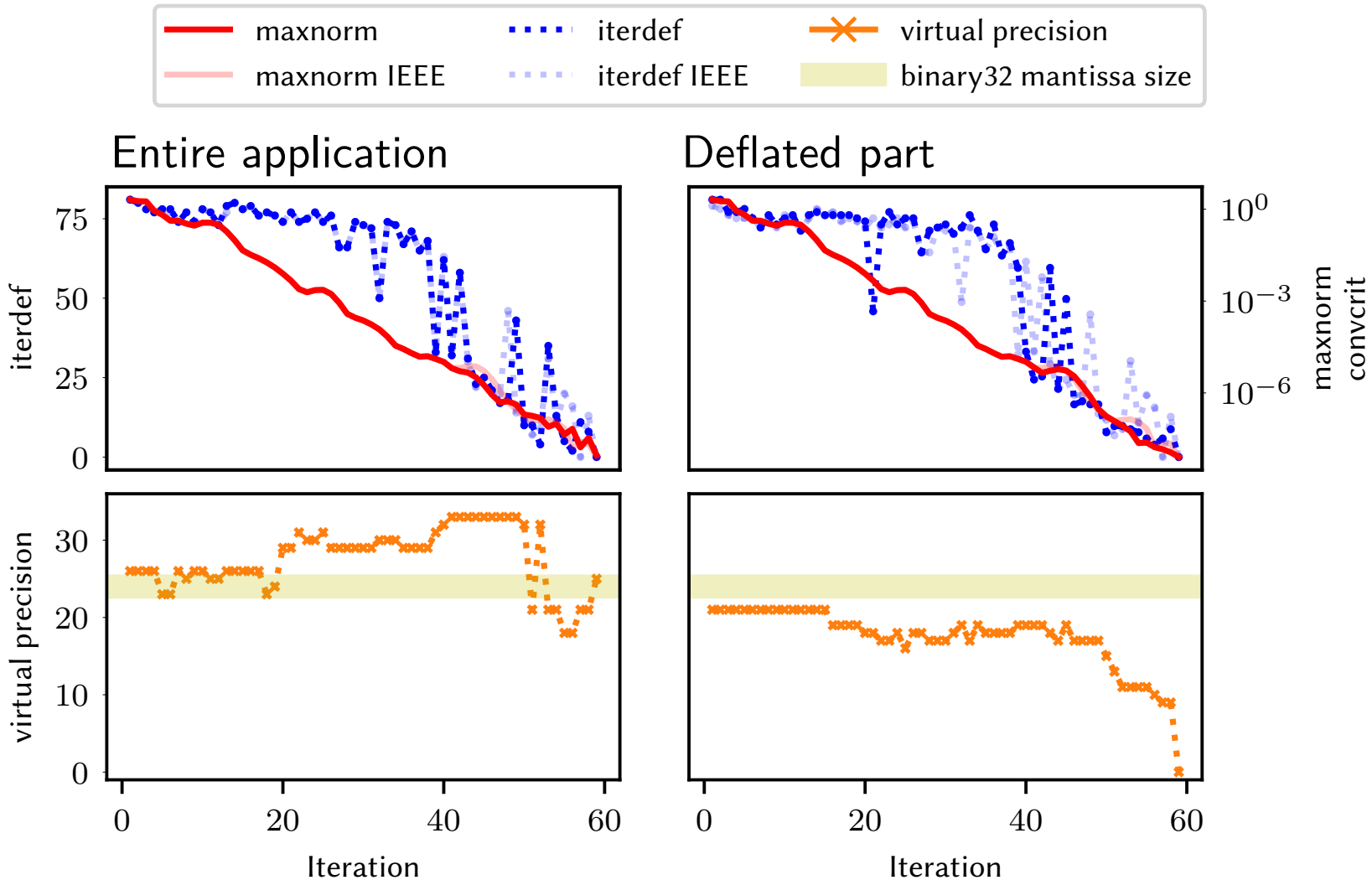


Figure: Adaptive precision searching on YALES2's DPCG on 1 MPI. $\mathbf{r} = \mathbf{8}$. Reduced precision solution follows the reference IEEE convergence profile

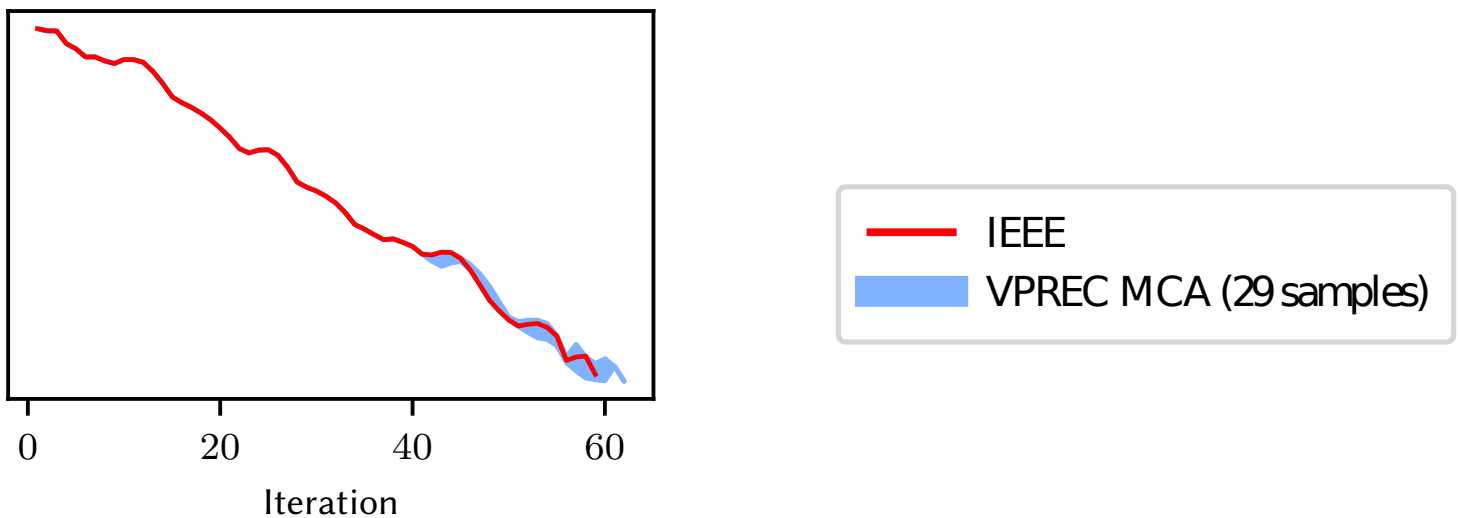
Validating resiliency to round-off errors

VPREC is a deterministic computation model

- How to validate configuration's robustness to rounding errors?

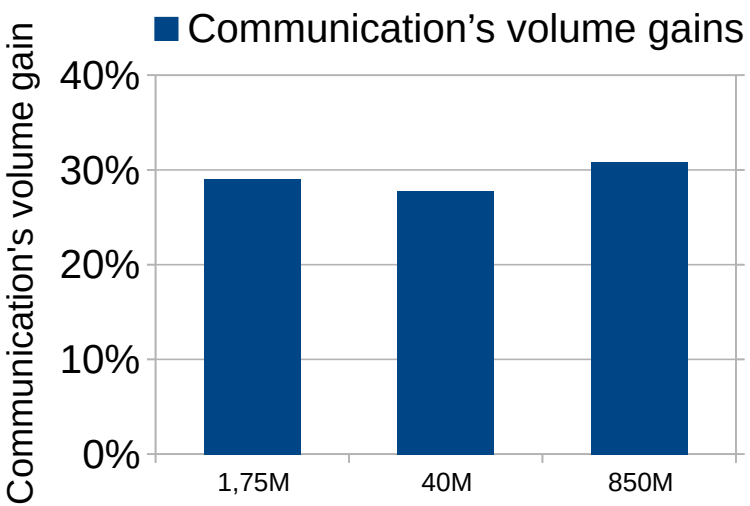
Using Monte Carlo Arithmetic [Parker, 1997]

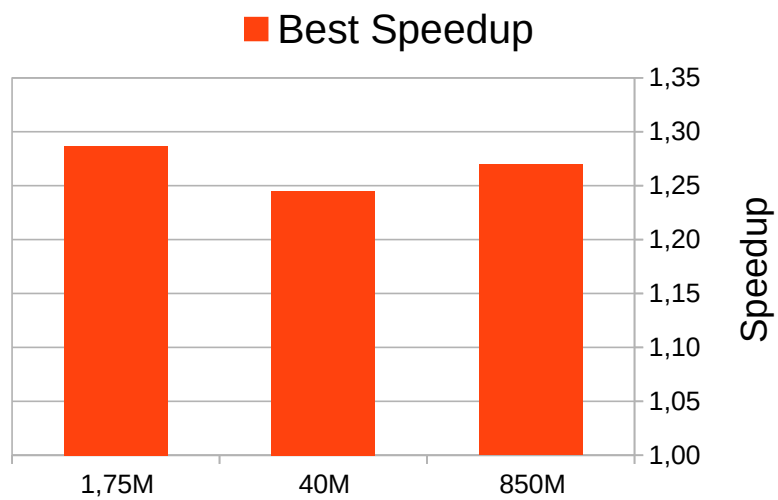
- Introduces random noise to model round-off errors at each FP ops
- Used as second step for validating VPREC configurations
- 29 converged MCA samples gives 90% of probability with 0.95 confidence level based on Bernouilli trials [Sohier et al., 2018] that configuration is robust to round-off errors



Performance validation on mixed-precision version

- Mixed-version to use both `binary32` and `binary64` formats in the deflated operator
- Switch from `binary32` to `binary64` when convergence criteria is below 10^{-7}
- Run up to 560 cores on CRIANN cluster (66 bisocket Intel Xeon E5-2680 nodes and Intel Omnipath interconnect)





Conclusion

VPREC

- Easily tests custom formats on HPC codes

Find lowest precision

- Automatic search to find lowest precision
- Reduces working precision while preserving accuracy and convergence
- Robustness to rounding errors thanks to MCA

Validation on industrial code Yales2

- Mixed-precision version scaling up to 560 cores
- Average communications' volume reduction by **30%** and best speedup of **1.28**

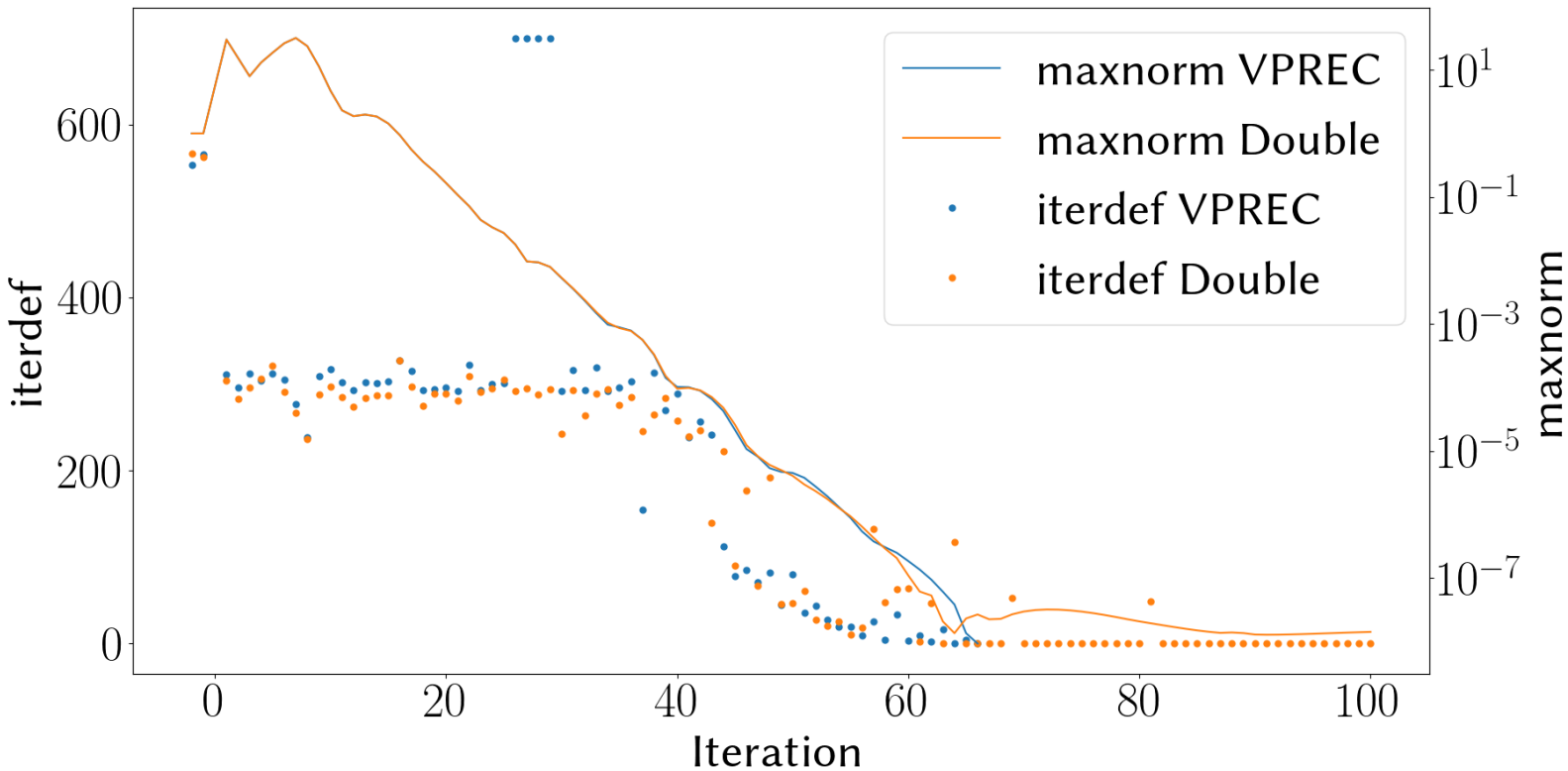
Backup slides

IEEE-754 formats

Name	Exponent (bits)	Pseudo-Mantissa (bits)	e_{min}, e_{max}
binary16	5	10	(-14,+15)
binary32	8	23	(-126,+127)
binary64	11	52	(-1022,+1023)
binary128	15	112	(-16382,+16383)

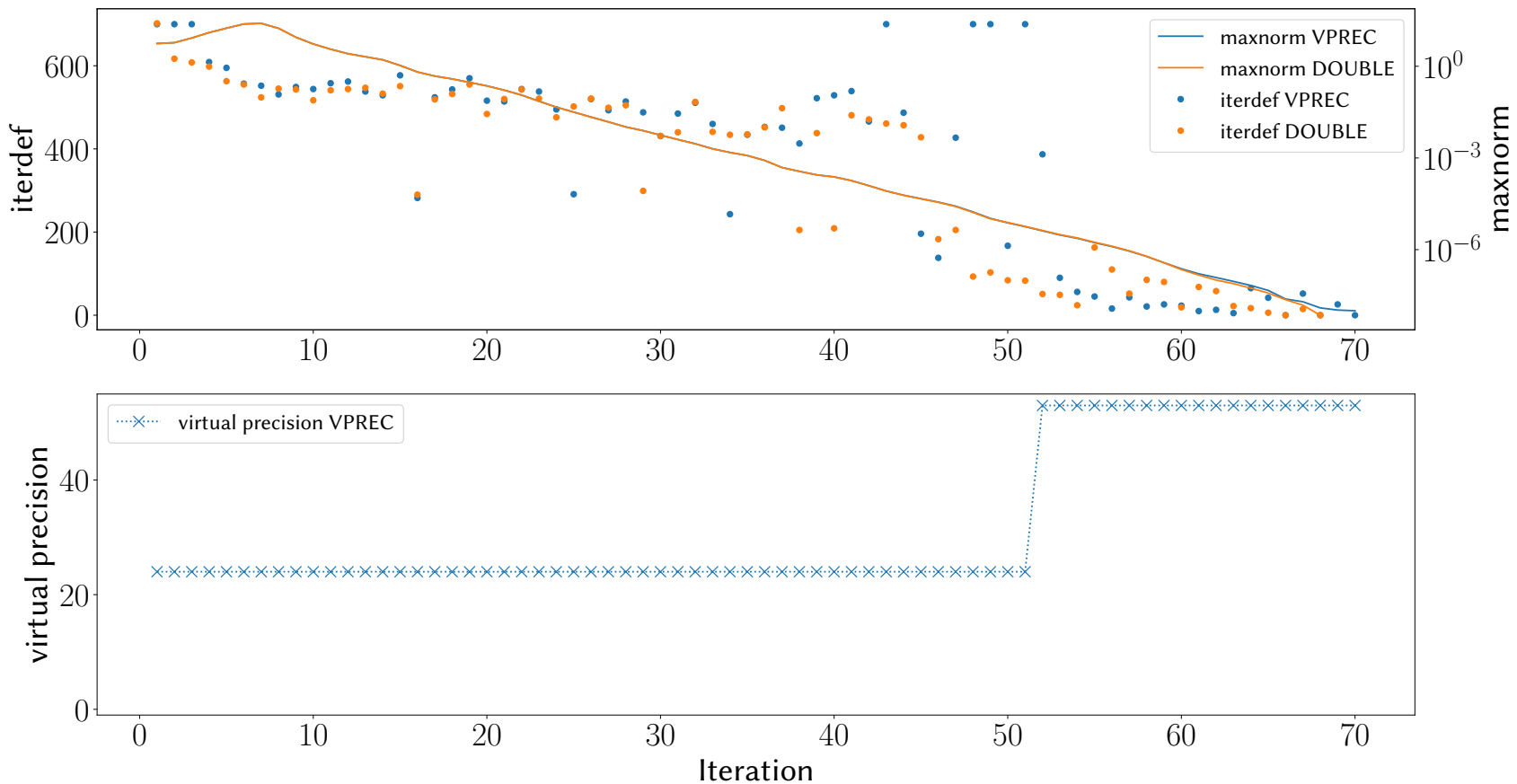
Table: Binary formats of the IEEE-754 norme

Accelerate the convergence



Exploration on *preccinsta* 850M on 112 MPI cores on deflated part. Solution found by the exploration (*VPREC*) converges in 66 iterations while the original (*Double*) converges in 100 iterations.
Reducing precision can be seen as a high-pass filter.
75% of solutions found by VPREC used less iterations than the IEEE version

Billion mesh elements



Exploration on *preccinsta* with billion of elements on 336 MPI cores. VPREC exploration shows that we need more precision at the end to reach the convergence. Fix $r = 8$

References I

[dlf,] Dlfloat: A 16-b floating point format designed for deep learning training and inference.

[Anzt et al., 2017] Anzt, H., Dongarra, J., Flegar, G., Higham, N. J., and Quintana-Ortí, E. S. (2017).
Adaptive precision in block-jacobi preconditioning for iterative sparse linear system solvers.
Concurrency and Computation: Practice and Experience, page e4460.

[Chatelain et al., 2018] Chatelain, Y., Castro, P. D. O., Petit, E., Defour, D., Bieder, J., and Torrent, M. (2018).
Veritracer: Context-enriched tracer for floating-point arithmetic analysis.
In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 61–68. IEEE.

References II

[Denis et al., 2016] Denis, C., de Oliveira Castro, P., and Petit, E. (2016).
Verificarlo: checking floating point accuracy through monte carlo arithmetic.
In *Computer Arithmetic (ARITH), 23nd Symposium on*, pages 55–62. IEEE.

[Fleischer et al., 2018] Fleischer, B., Shukla, S., Ziegler, M., Silberman, J., Oh, J., Srinivasan, V., Choi, J., Mueller, S., Agrawal, A., Babinsky, T., Cao, N., Chen, C., Chuang, P., Fox, T., Gristede, G., Guillorn, M., Haynie, H., Klaiber, M., Lee, D., Lo, S., Maier, G., Scheuermann, M., Venkataramani, S., Vezyrtzis, C., Wang, N., Yee, F., Zhou, C., Lu, P., Curran, B., Chang, L., and Gopalakrishnan, K. (2018).
A scalable multi- teraops deep learning processor core for ai trainina and inference.
In *2018 IEEE Symposium on VLSI Circuits*, pages 35–36.

References III

[Graillat et al., 2016] Graillat, S., Jézéquel, F., Picot, R., Févotte, F., and Lathuiliere, B. (2016).
Promise: floating-point precision tuning with stochastic arithmetic.
In *Proceedings of the 17th International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN)*, pages 98–99.

[Haidar et al., 2018] Haidar, A., Tomov, S., Dongarra, J., and Higham, N. J. (2018).
Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 47. IEEE Press.

References IV

[Jouppi et al., 2017] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017).
In-datacenter performance analysis of a tensor processing unit.
In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE.

[Lam et al., 2013] Lam, M. O., Hollingsworth, J. K., de Supinski, B. R., and LeGendre, M. P. (2013).
Automatically adapting programs for mixed-precision floating-point computation.
In *Proc. of the 27th International conference on supercomputing*, pages 369–378. ACM.

[LLC, 2019] LLC, G. (2019).
Using bfloat16 with tensorflow models.
`https://cloud.google.com/tpu/docs/bfloat16`.

References V

[Parker, 1997] Parker, D. S. (1997).
Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic.
University of California. Computer Science Department.

[Rao, 2018] Rao, N. (2018).
Intel® nervana™ neural network processors (nnp) redefine ai silicon.
Intel https://ai. intel. com/intel-nervana-neural-network-processors-nnp-redefine-ai-silicon.

[Rubio-González et al., 2016] Rubio-González, C., Nguyen, C., Mehne, B., Sen, K., Demmel, J., Kahan, W., Iancu, C., Lavrijsen, W., Bailey, D. H., and Hough, D. (2016).
Floating-point precision tuning using blame analysis.
In *Proceedings of the 38th International Conference on Software Engineering*, pages 1074–1085. ACM.

References VI

[Rubio-González et al., 2013] Rubio-González, C., Nguyen, C., Nguyen, H. D., Demmel, J., Kahan, W., Sen, K., Bailey, D. H., Iancu, C., and Hough, D. (2013).
Precimonious: Tuning assistant for floating-point precision.
In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 27. ACM.

[Sohier et al., 2018] Sohier, D., De Oliveira Castro, P., Févotte, F., Lathuilière, B., Petit, E., and Jamond, O. (2018).
Confidence Intervals for Stochastic Arithmetic.
working paper or preprint.

[Wilkinson, 1994] Wilkinson, J. H. (1994).
Rounding errors in algebraic processes.
Courier Corporation.