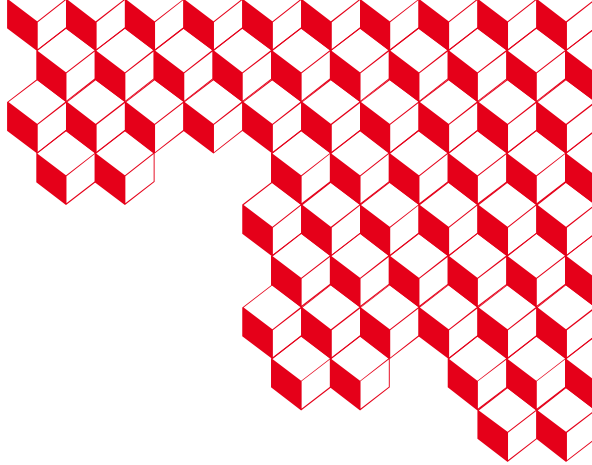
The Variable Precision Processor VXP: preliminary performance results on generic Krylov-based solvers

Yves Durand, Jérome Fereyre, César Fuguet Tortolero



The Variable eXtended Precision Accelerator VXP

preliminary performance results on generic Krylov-based solvers

Yves Durand, Jérôme Fereyre, Eric Guthmuller, CEA

yves.durand@cea.fr

Colloque Sparse Days 2023 ***19 – 20 Juin 2023***

VXP: Project Context

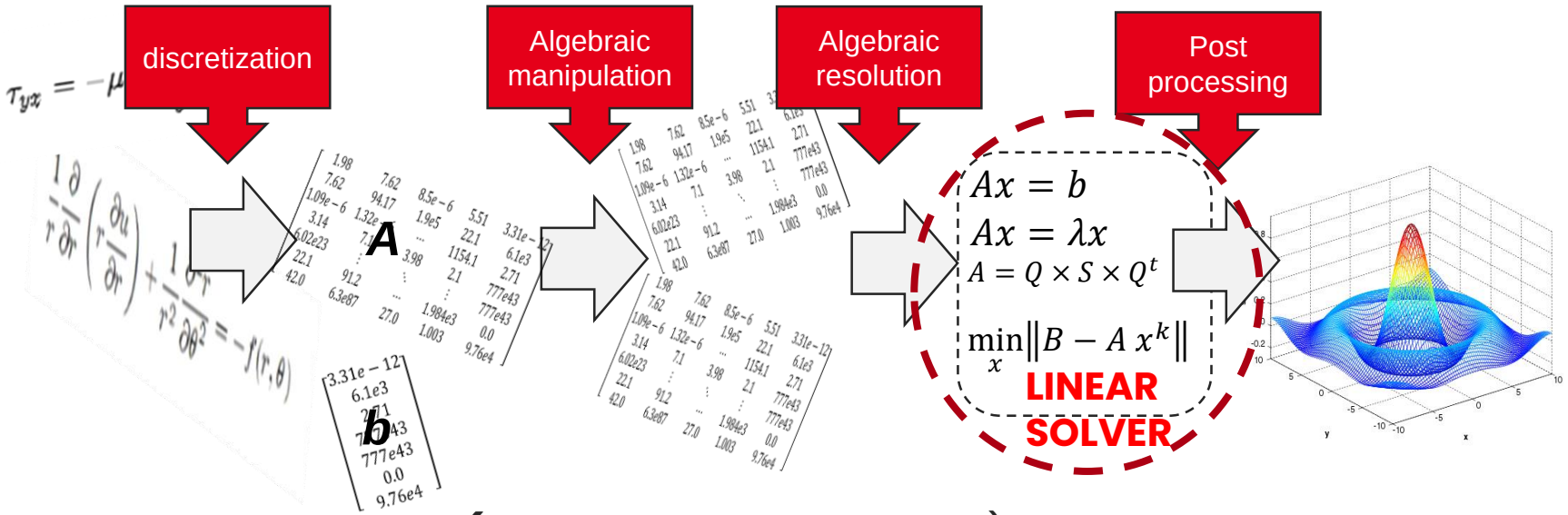
- Work initiated in 2016 at CEA Grenoble
- Original motivations :
  - Accelerate convergence of linear solvers for structural modeling, computational physics
  - Improve accuracy in computational geometry
  - → provide hardware support for 1/ arbitrary extended precision and 2/ interval arithmetics

- Current focus is arbitrary precision applied to Krylov-based iterative solvers
  - Goal : Improve stability of short-recurrence Krylov solvers / eigensolvers
  - Application to large size sparse matrices, dense vectors

- Sponsors
  - Initial support from ANR Imprenum project (With INSA Lyon and University Grenoble Alpes)
  - Current support with EPI (European Processor Initiative) https://www.european-processor-initiative.eu/

VXP: Motivation

- The focus is on linear solvers/eigensolvers which may take 80% of computing time on computing nodes



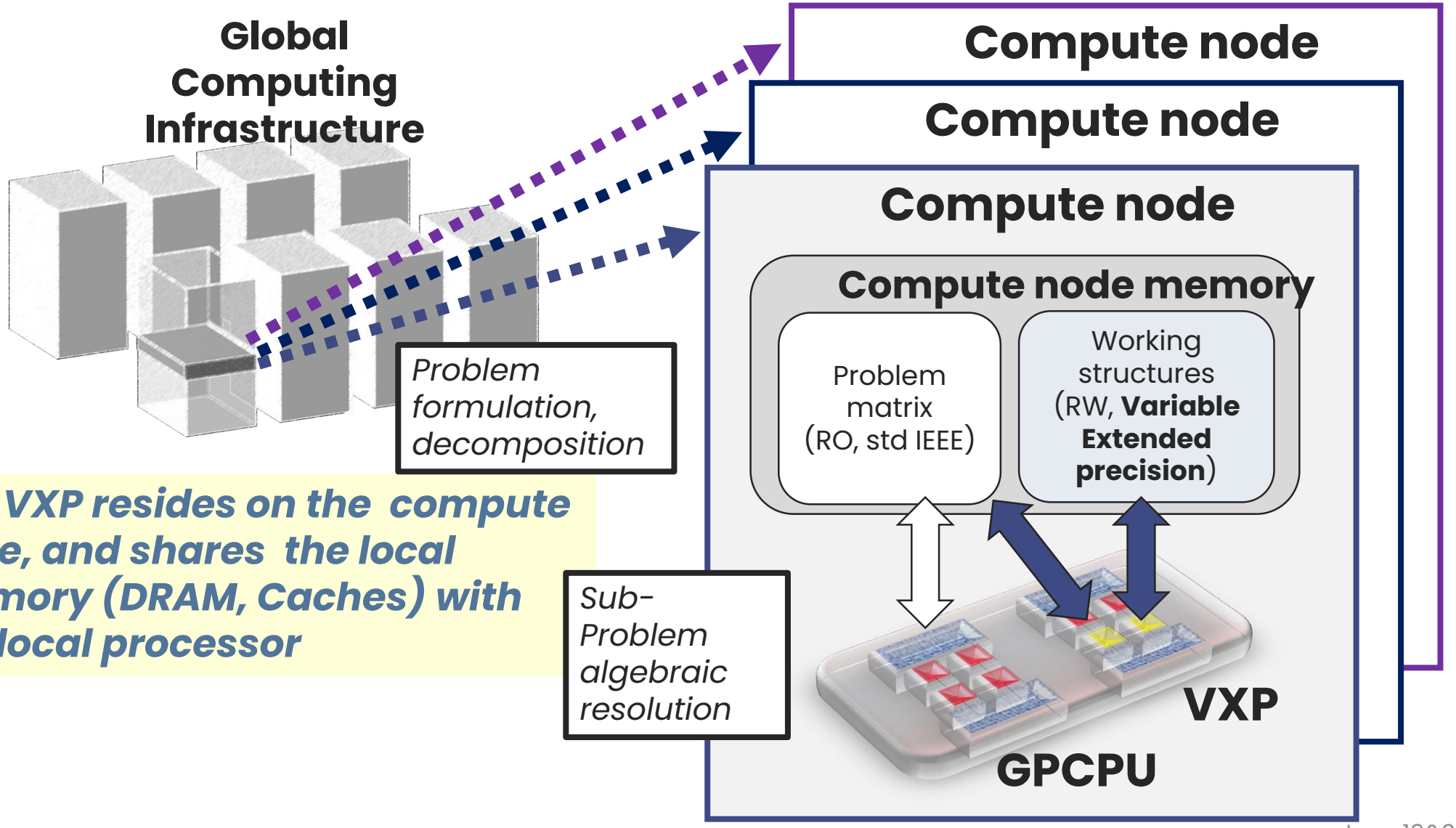
- **Using extended precision (**64-512 bits of mantissa) significantly improves convergence
  - and avoids/simplifies numerical pre-processing (preconditionning, orthogonalization)
- **However** software-based support for extended precision is not efficient
  - Memory pressure too high
- VXP provides **native** extended precision in hardware with low performance overhead

Two details of importance…

1. We are dealing with the **working precision**, i.e. the precision of intermediate floating point computations (aka $\epsilon_m = 2^{1-prec}$)
   - and **not** with data precision
   - In numerical terms, working precision is linked the relative representation error, bounded by u (ulp) or δ (error analysis)
     - The distance of representation $\hat{x}$ to real x is bounded: $\frac{|x-\hat{x}|}{|x|} < u$

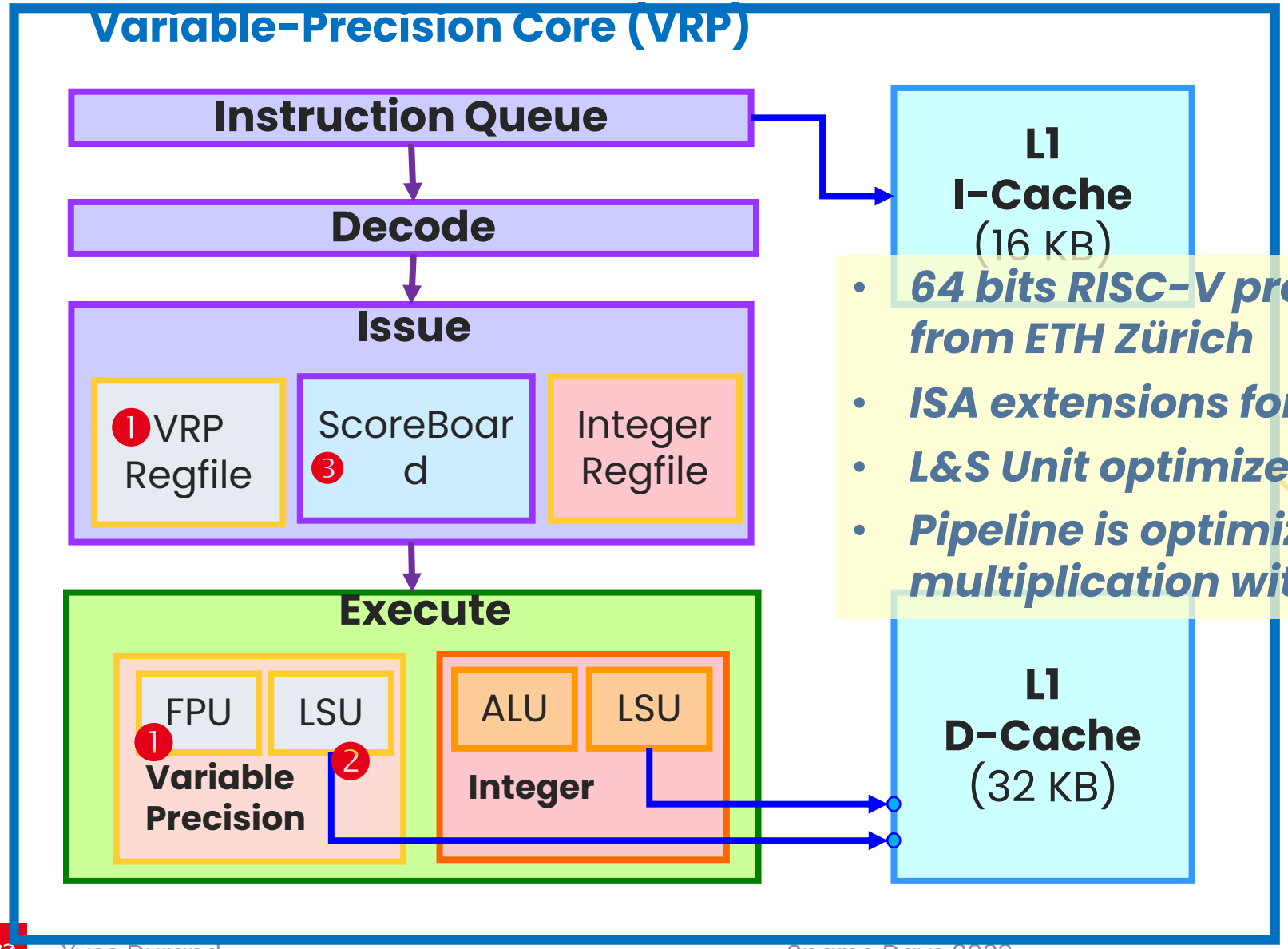
2. Precision applies to all floating point arithmetics (FP) numbers both
   - Inside the processor,
   - And in memory, using IEEE standard extendable format.

VXP in Computing Infrastructure



- ***The VXP resides on the compute node, and shares the local memory (DRAM, Caches) with the local processor***

Differences with a mainstream processor

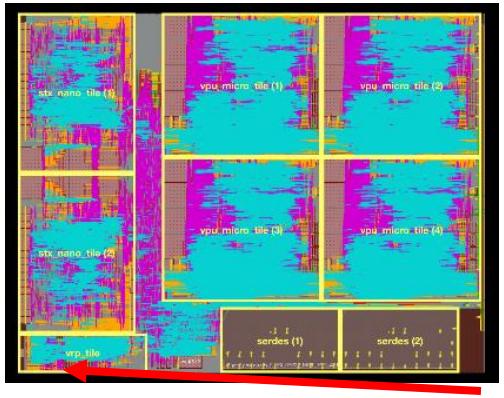


- *64 bits RISC-V processor based on « Ariane » from ETH Zürich*
- *ISA extensions for VP operations ❶*
- *L&S Unit optimized for unaligned transfers ❷*
- *Pipeline is optimized for (dense) Matrix-vector multiplication with 192 bit of precision ❸*

VXP: Current Status

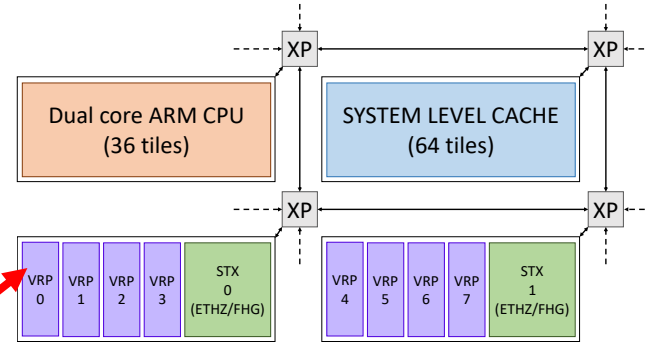
VXP is a standalone processor used as accelerator for linear kernels

FPGA Prototype (In use)



EPI Accelerator TestChip
GF22FDX
Chips available on Q2 2023

European Processor Initiative



VRP Accelerator

RHEA
TSMC7
Chips available on Q2 2024

FPGA prototype
2 VRP cores
Clock frequency: 83 MHz

EPACTC 1.5 (VXP features)
Up to 512 bits of significand
IEEE extendable memory format
8 dynamic data formats
Indexed load/store operations
High-Throughput memory subsystem
Clock frequency: 1.2 GHz

RHEA (VXP features)
Same features that in EPACTC 1.5
Two tiles with 4 VXP cores each
Clock frequency: 1.4 GHz

« benchmarking » the VXP 1/2

1. Key operation is the **matrix-vector multiplication** $y = A \times x$ (SparsexDense, DensexDense): we measure its **latency** in terms of MAC/cycle for different precisions

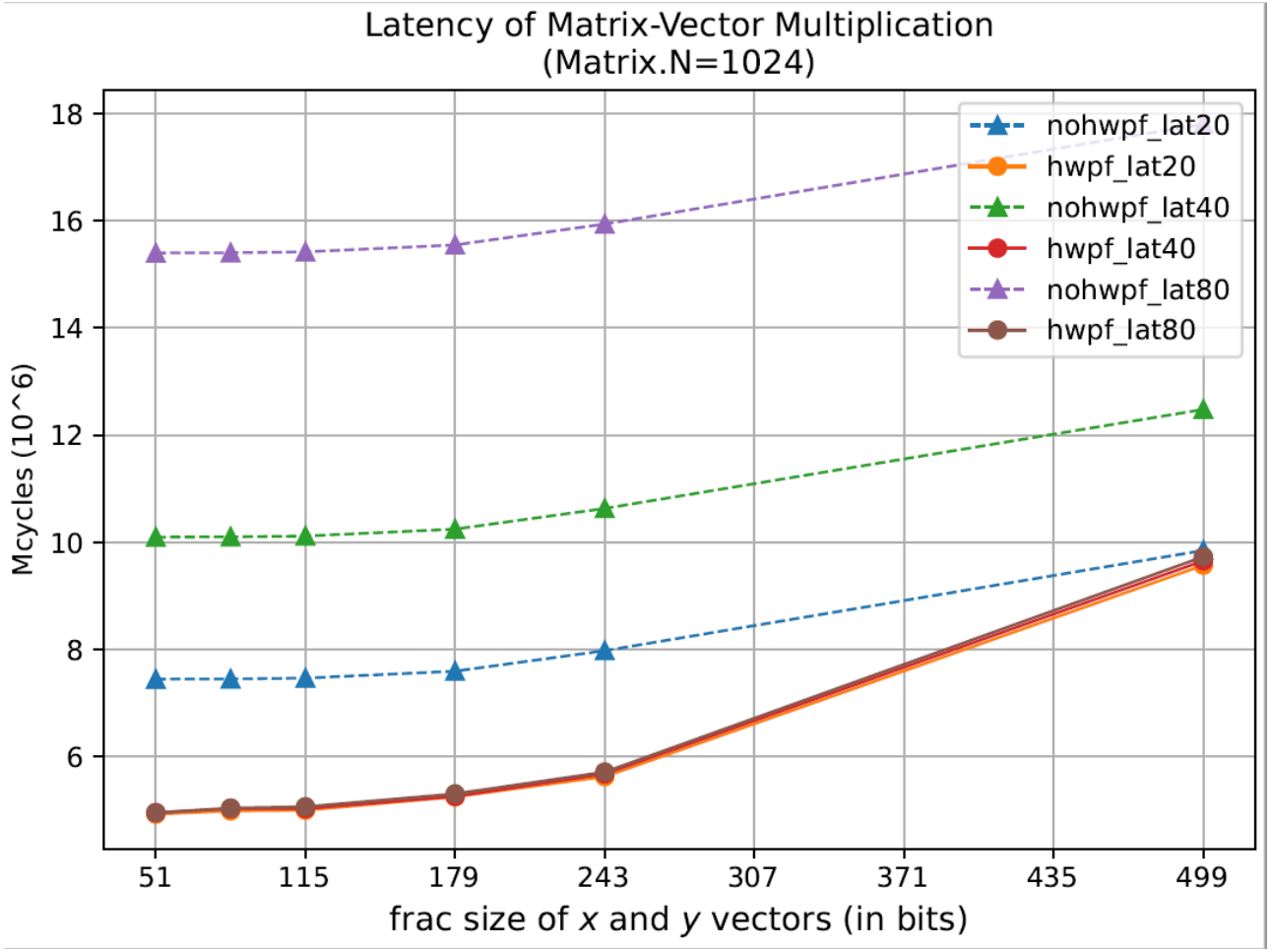
2. global performance of reference linear solvers ($A \times x = b$), e.g. PCG, BiCG :
   1. Comparaison of Convergence speed for different precisions.
      Expressed In terms of number of iterations to reach a predefined error threshold (*tolerance*)
   2. global latency (measured in clock cycles) with different precisions, same tolerance and same problems

« benchmarking » the VXP 2/2

- Problem matrix A are dense/sparse, in double (64b) format
- Vectors b are dense, double vectors
- Problems are taken from
  - classical benchmarks in SuiteSparse Matrix Collection https://sparse.tamu.edu/

  - Synthetic random dense matrices using « ransvd » method [Higham 2002] for dense matrices
  - Synthetic sparse matrices using Saad's finite difference scheme $-\Delta u + \gamma x \cdot \nabla u + \beta u = f$ [Chen 2016]

Matrix-vector multiplication latency 1/2

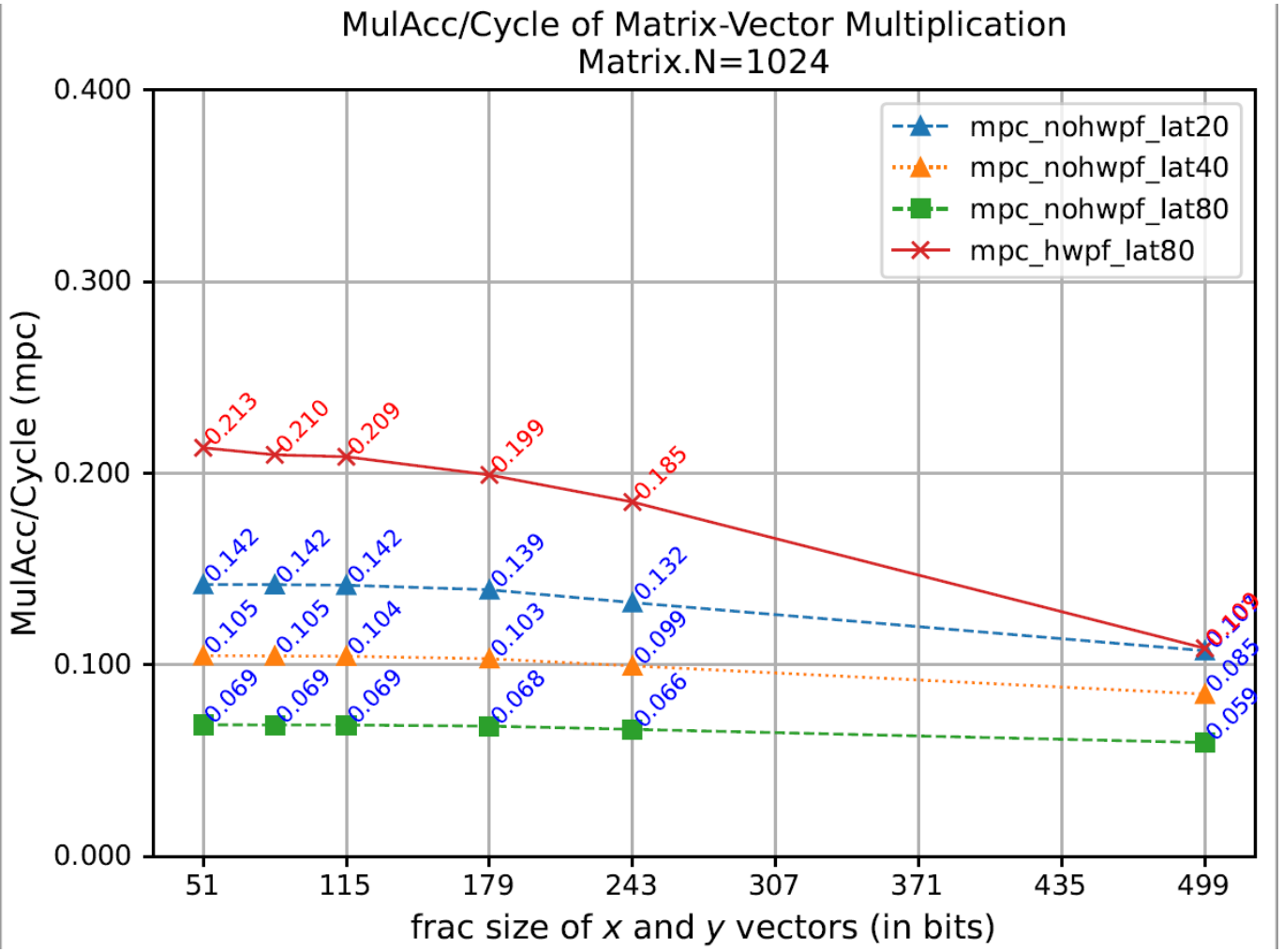


Latency of a single-core dense MV operation of size 1024x1024 in extended precision with varying precisions.

- Different lines represent different DRAM latencies
- Higher curves are measured without hardware prefetcher
- Bottom-level curves (almost identical) represent measure with the hardware prefetcher activated.

→ The prefetcher is key for masking cache miss latency

Matrix-vector multiplication latency 1/2



IPC: MAC Operation per cycle of a MV operation of size 1024x1024 in extended precision with varying precisions.

- Different lines represent different DRAM latencies
- bottom curves are measured without hardware prefetcher
- upper curve is measured with the hardware prefetcher activated.

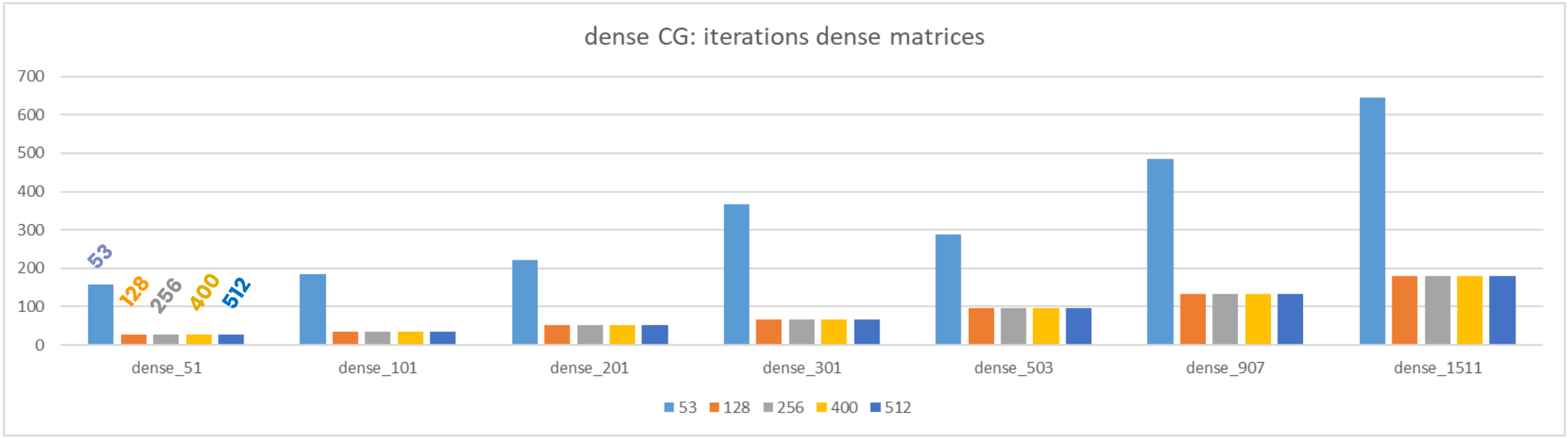
- **With prefetcher, MAC IPC ranges from 0.21 (in double) to 0.10 (for 499 bits of mantissa) → factor of 2**

Measuring precision impact & performance on solvers

- 2 main cases
  - **Preconditionned Conjugate Gradient** on symmetric real matrices
    - Random denses matrices: non preconditionner
    - Sparse, real cases matrices : Jacobi preconditionner
  - **Biconjugate Gradient** on unsymmetric matrices
    - Sparse, real and synthetic cases, with/without preconditionner
- Bonus: **lanczos tri-diagonalization** with periodic re-orthogonalization
  - Precision alone is not sufficient, Re-orthogonalization necessary
  - New criterion: number of re-orthogonalizations

CG on synthetic dense matrices : iteration count

The Diagram below represents iteration count of kernel CG (conjugate Gradient) running on random symmetric matrices with controlled eigenvalue profile (cliff)
Bars correspond to different precisions (e.g. 53/128/256/400/512)

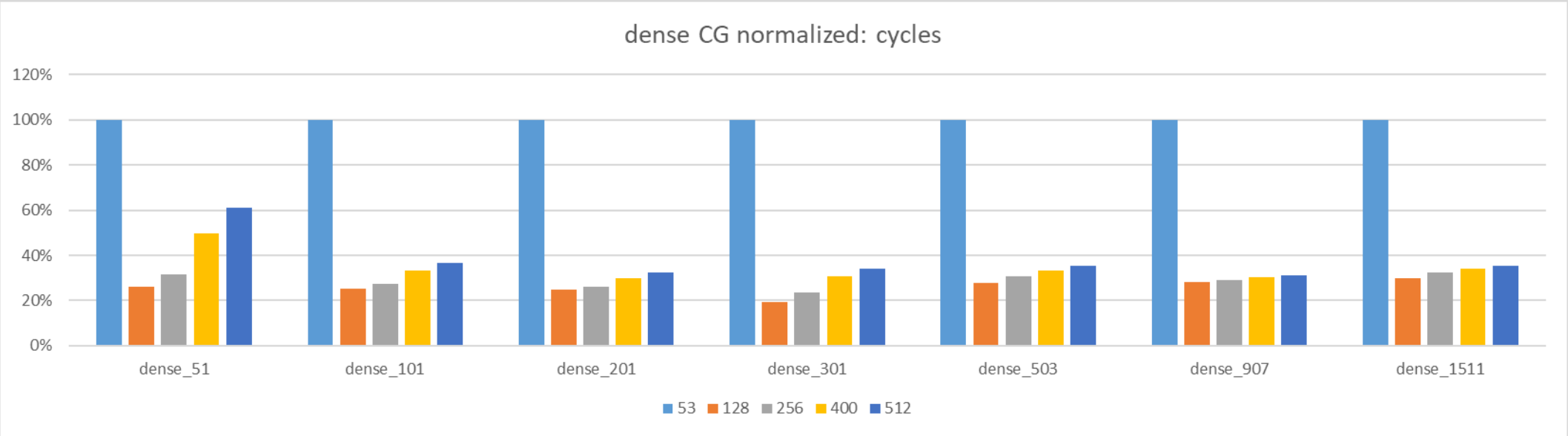


- Tolerance is set to 10^{-12}
- Value below 0 means that the run did not converge
- Conclusions

1. Iteration count decreases consequently with precision starting at 128
2. Effect of precision is more visible than with sparse matrices

CG on synthetic dense matrices : cycle count (values from June, 2023)

The Diagram below represents cycle count of kernel CG (conjugate Gradient) running on random symmetric matrices with controlled eigenvalue profile (cliff)
Bars correspond to different precisions (e.g. 53/128/256/400/512)

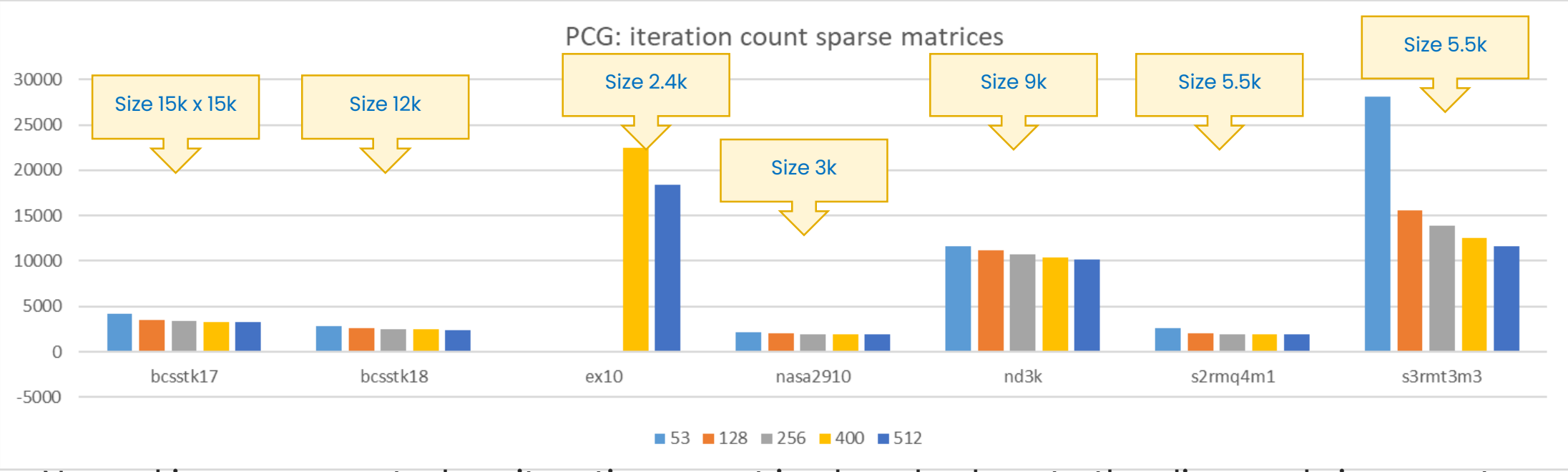


- Value are normalized against precision 53 (double)
- Conclusions

1. Sweet spot for latency around precision 128
2. Latency is reduced by a factor of 4÷

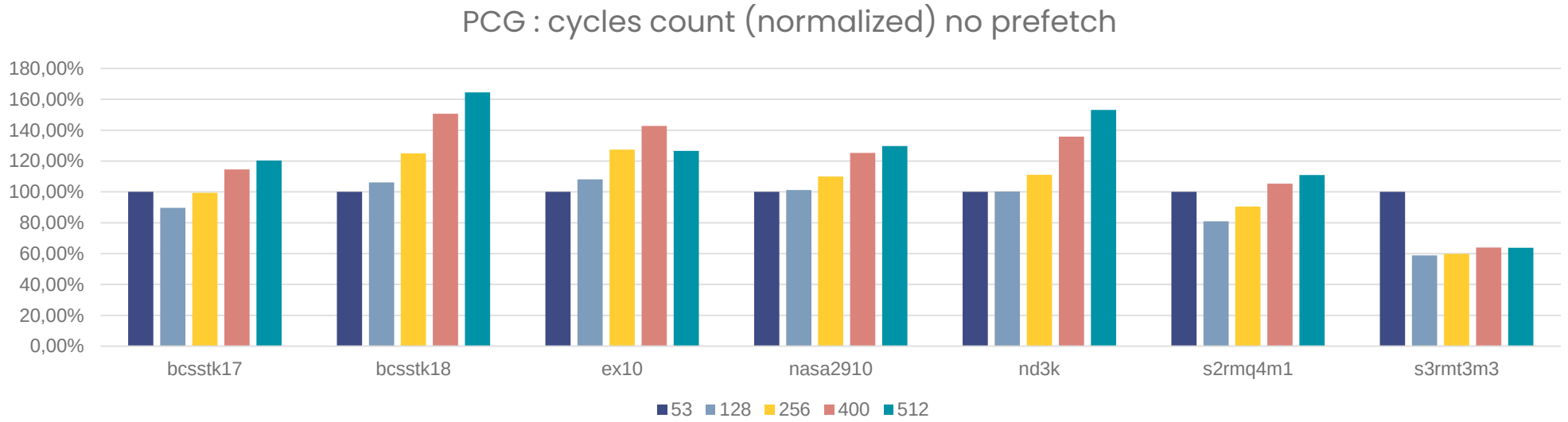
Preconditionned CG on Matrix Market sparse matrices : iteration count

- The Diagram below represents iteration count of kernel PCG (conjugate Gradient, Jacobi preconditionning) running on MatrixMarket matrices Bars correspond to different precisions (e.g. 53/128/256/400)
- Tolerance is set to 10^{-12} , Value below 0 means that the run did not converge



- No real improvement when iteration count is already close to the diagonal size → not surprising
- **Extended precision improves the iteration count when it is >> diagonal size**

Preconditionned CG on Matrix Market sparse matrices : cycle count



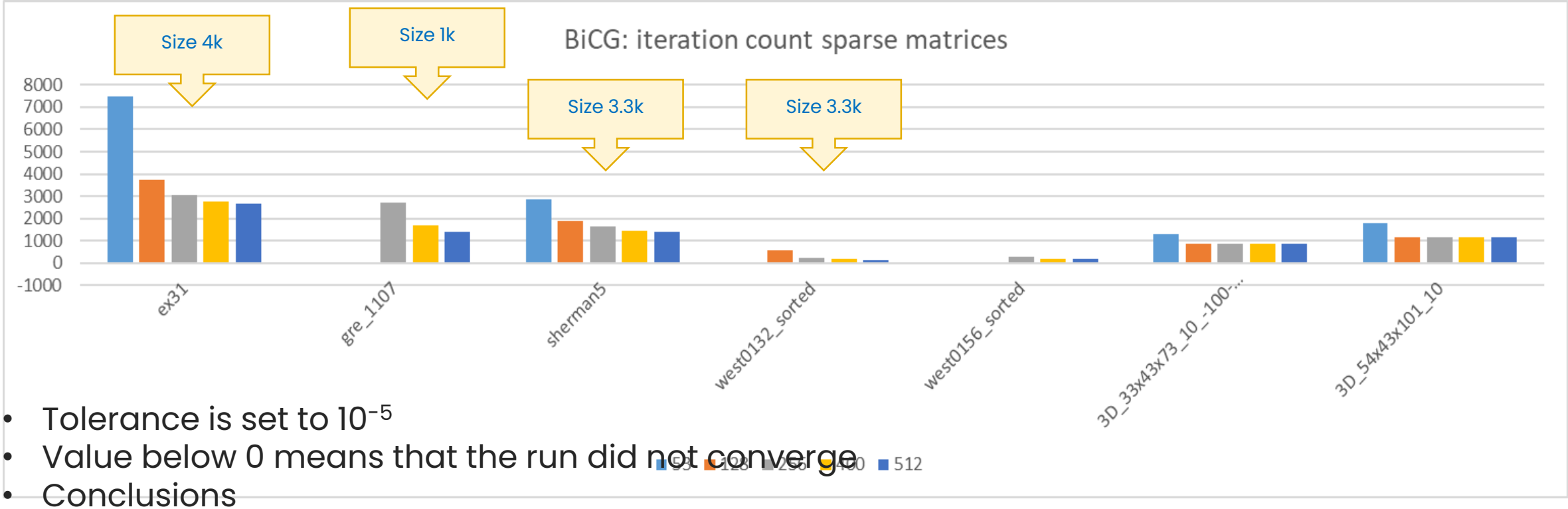
Conclusion:

- No decisive latency improvement for small matrices, better with larger (>5K)
- However, these measures do not activate prefetching : presumably, memory access latencies dominate arithmetic latencies

→ we are revisiting our spMV routines to minimize/anticipate caches misses

BiCG on Matrix Market matrices : iteration count

The Diagram below represents iteration count of kernel BiCG (Biconjugate Gradient) running on matrices from the SuiteSparse collection

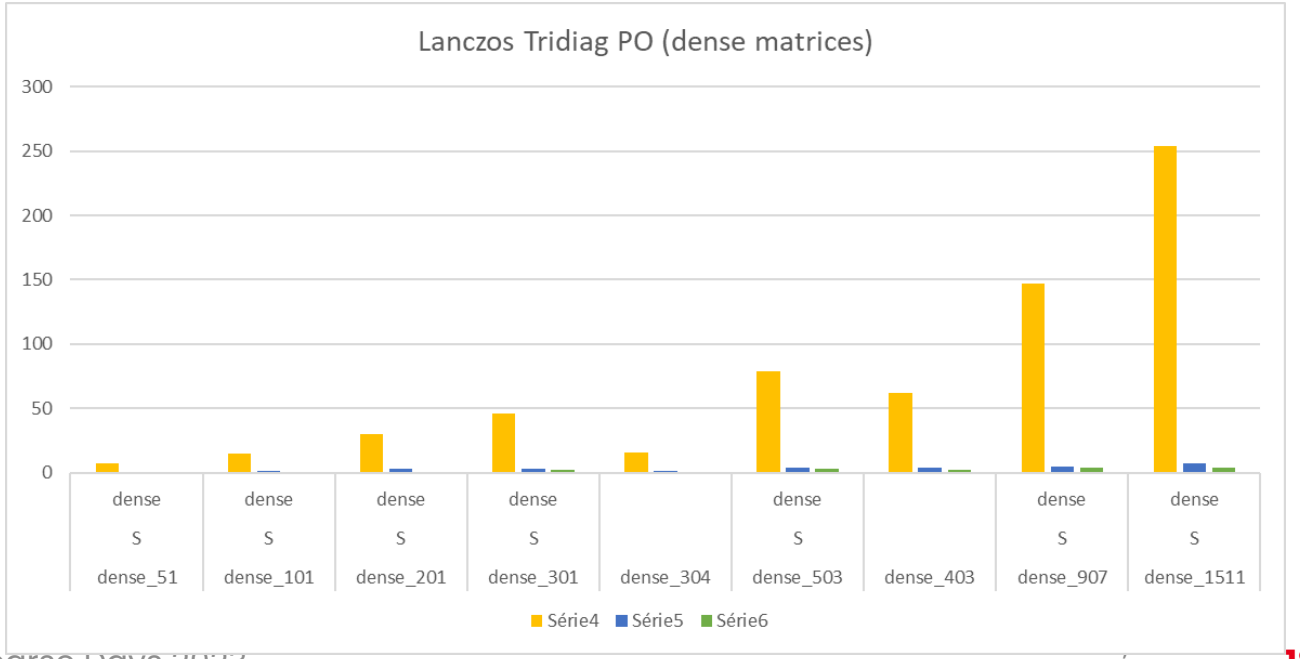


- Tolerance is set to 10^{-5}
- Value below 0 means that the run did not converge
- Conclusions

1. In several cases, convergence is not attained with precision 53 (double) but works at higher precisions : bp_1000, gre_1107, lmpcol_a, lns__131
2. Iteration count decreases consequently with precision

Lanczos tridiagonalization on synthetic dense matrices

- Lanczos tridiagonalization is the first step of eigendecomposition → key numerical technique
- It systematically requires reorthogonalization : we use the Periodic reorthogonalization algorithm from Simon (aka PO TD) [Simon 2000]
- This algorithm estimates the loss of orthogonality ($\omega_{i,j=} q_i' q_j$) at each step j and performs a partial reorthogonalisation when $max(\omega_{i,j}) \geq \sqrt{\epsilon_p}$
- The Diagram below represents reorthogonalization counts of kernel PO TD running on dense matrices
- Bars correspond to different precisions (e.g. 53//256/512)

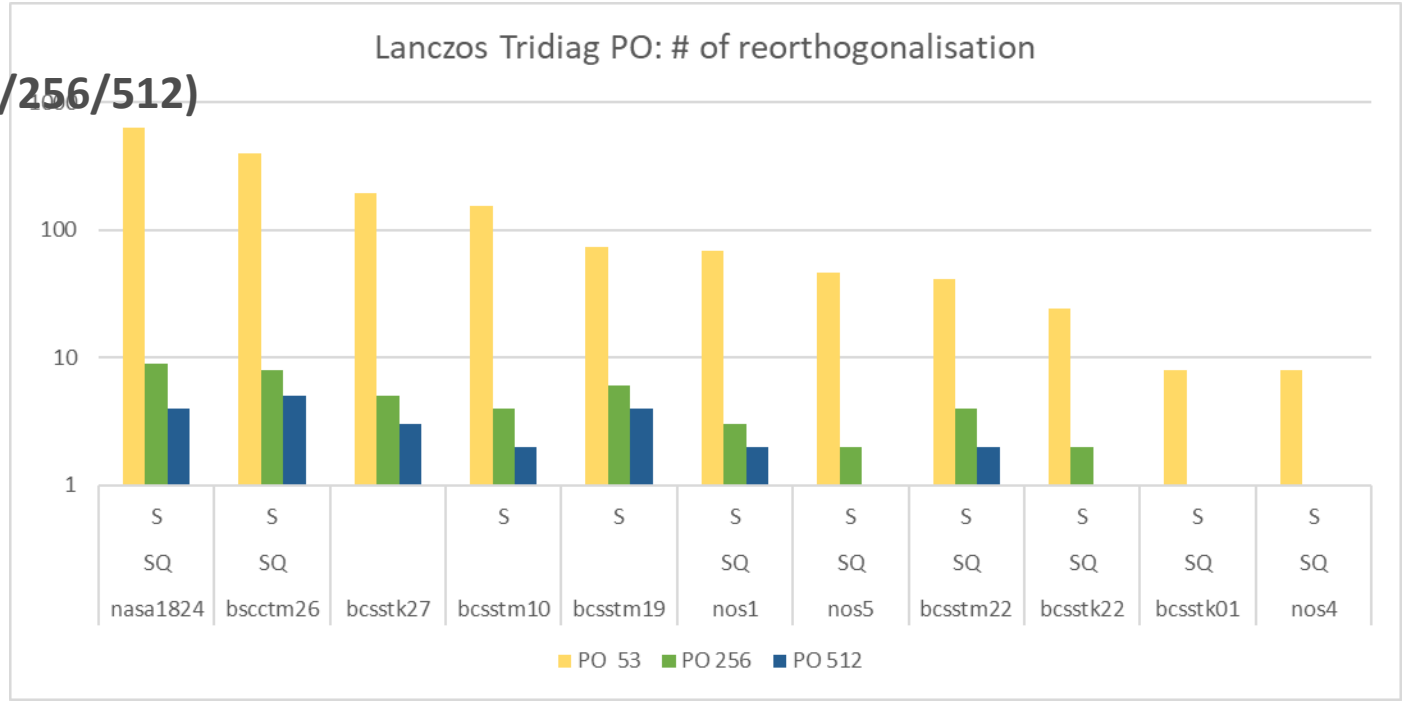


Conclusion:

- Using 256/400 bits mantissas saves a lot of reorthogonalization work
- Besides, the orthogonality estimator used here is consistent with actual orthogonality computation

Lanczos tridiagonalization on SparseMatrix matrices

- Lanczos tridiagonalization is the first step of eigendecomposition → key numerical technique
- It systematically requires reorthogonalization : we use the Periodic reorthogonalization algorithm from Simon (aka PO TD)
- This algorithm estimates the loss of orthogonality ($\omega_{i,j=} q_i' q_j$) at each step j and performs a partial reorthogonalisation when $max(\omega_{i,j}) \geq \sqrt{\epsilon_p}$
- The Diagram below represents reorthogonalization counts of kernel PO TD running on SparseMatrix matrices
- Bars correspond to different precisions (e.g. 53//256/512)



Conclusion:

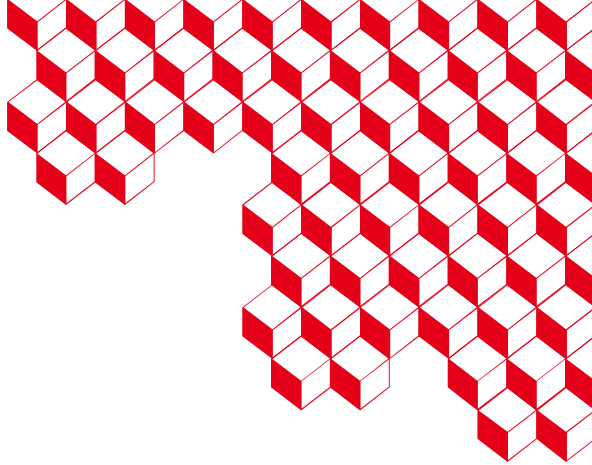
- Using 256/400 bits mantissas saves a lot of reorthogonalization work
- Besides, the orthogonality estimator used here is not really reliable for full range of eigenvalues of sparse matrices

Conclusions

- Variable Extended Precision gives opportunities for using simpler algorithms, improves convergence →improve efficiency
  - Standard FP format exists already, Numerical analysis background is there
  - Surpringly, BiCG benefits from precision, becomes reliable → our prefered choice for real-life cases (up to 20 K size so far)
- Extended precision does not replace usual techniques
  - It may simplify preconditionning
  - It unlocks the door for larger (eigen)problem sizes (future work)
- « Effective » means usable
  - consistent support in memory, reasonably fast execution
  - … and full software support
- Our work proves the feasibility in silicon
  - Fast Arithmetics for up to 512(/1024) bits of mantissa
  - Effective support in memory for unaligned FP arrays

references

- [Durand 2022] Y. Durand, E. Guthmuller, C. Fuguet, J. Fereyre, A. Bocco, and R. Alidori. *Accelerating variants of the conjugate gradient with the variable precision processor*. In 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)
- [Trevisan 2021] Tiago Trevisan Jost, Yves Durand, Christian Fabre, Albert Cohen, and Frédéric Petrot. *Seamless compiler integration of variable precision floating-point arithmetic*. In 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)
- [Higham 2002] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, second edition, 2002.
- [Chen 2016] Jie Chen, Lois C. Mcinnes, and Hong Zhang. *Analysis and practical use of flexible bicgstab*. J. Sci. Comput., 68(2):803–825, aug 2016.
- [Ruhe 1999] Bai, Demmel, Dongarra. 1999. *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide (Draft, May 21, 1999)*. (Draft, May 21, 1999).



Thanks for your attention !

Contact:
Yves.durand@cea.fr

What is the difference between arbitrary extended precision and « mixed precision » ?

- **Mixed precision combines different fixed sizes representations,** e.g. floats (32 bits) and doubles (64 bits)
  - May also alternatively be done with double / quad

- Mixed precision for iterative refinement:
  - **Mixed precision fits well for iterative refinement methods**:
  - Inner Solve/factorization done in lower precision
  - Residual computation and correction done in higher precision

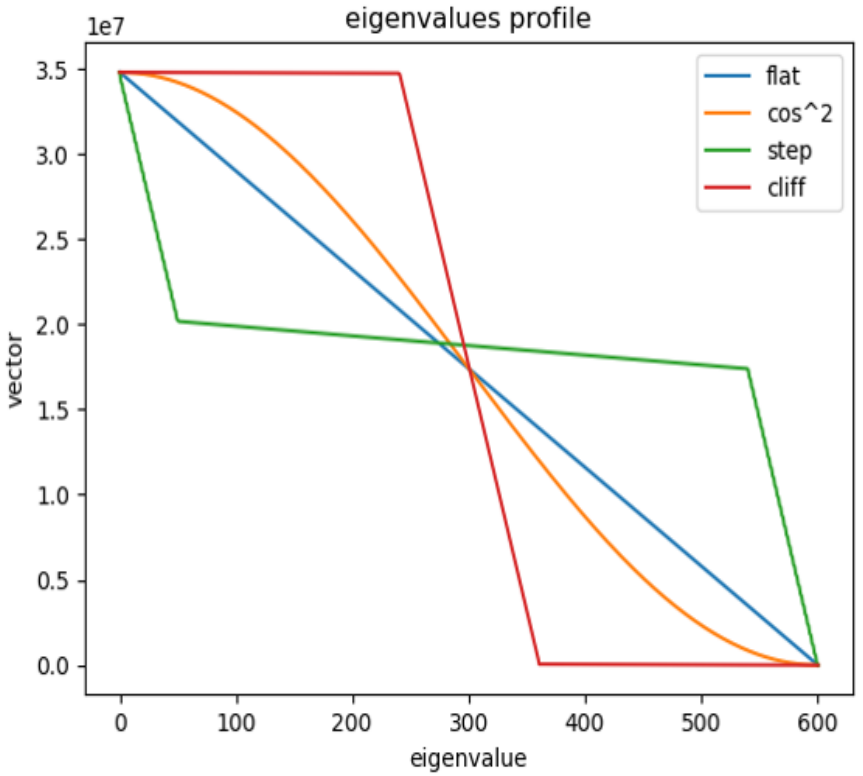
- However, the solve/ factorization is usually a direct method (eg a LU) → cost in $\mathcal{O}(n^2)$ in memory even if matrix is sparse
- Alternatively, Krylov method may be used as solver (but not in mixed precision)
  - (cf *Mixed Precision Iterative Refinement Methods for Linear Systems: Convergence Analysis Based on Krylov Subspace Methods*, Hartwig Anzt, Vincent Heuveline & Björn Rocker)

Profiles for the ransvd method

We generate a random matrix M by a simple multiplication M=σ.U ×D×V^T (U = V for symmetric matrices) where

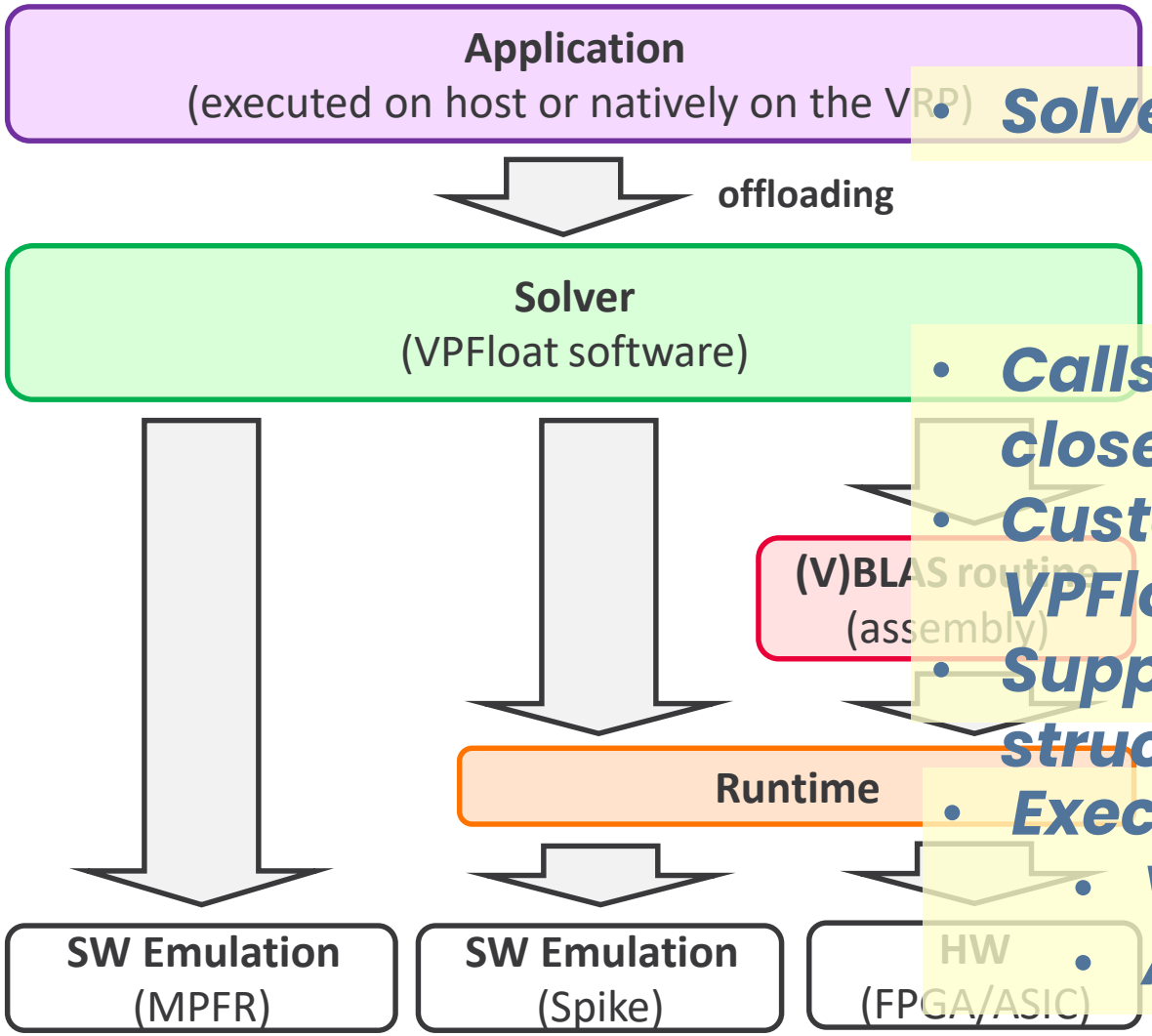
- σ is a scaling factor,
- U and V random orthonormal matrices and
- D a diagonal matrix whose eigenvalues follow a specific distribution.

Specifically, we use two distributions “cliff” (resp. “step”) which consist in three abutted segments with slopes 0:1; 10; 0:1 (resp. 10, 0.1, 10).



forced eigenvalues, sorted in decreasing order, for a 600x600 matrix. Scaling is defined arbitrarily.

VXP software stack




```
VPFloatArray X(EXP_SZ, FRAC_SZ, Ndiag);
…
```

```
// A*x = b
int cg_vp(int precision, int Ndiag,
          VPFloatArray & x, double *A,
          VPFloatArray b, double tolerance) {
…
    while (relerror < tolerance) {
…
        VBLAS::vgemv(precision, n, n, A, p_k, Ap_k, …)
```

```
// y = A*x
void VBLAS::vgemv(int precision, …){
  pser_evp(X.es(), X.fs(), EVP1); // memory precision
  for (int i = 0; i < m; i++) {
    pcvt_d_p(P24, 0);             // acc = 0
    for (int j = 0; j < n; j++) {
      ple(P0, a_ptr, 0, EVP0); // A(m)(n)
      ple(P1, x_ptr, 0, EVP1); // x(n)
      pmul(P0, P0, P1, EC0);   // A(m)(n)*x(n)
      padd(P24, P24, P0, EC0); // acc += A(m)(n)*x(n)
```

- ***Solvers interface close to PETsC***
- ***Calls to dense/sparse library close to BLAS***
- ***Custom C++ types VPFloat and VPFloatArray***
- ***Support of OSKI sparse structures***
- ***Execution backends:***
  - ***VRP***
  - ***Also MPFR, Spike emulation***