

EPI Forum

Barcelona, 9–10.10.2024.



VRP: A VARIABLE EXTENDED PRECISION ACCELERATOR FOR SCIENTIFIC COMPUTING APPLICATIONS



Authors:

Andrea Bocco,

A. Hoffmann,

E. Guthmuller,

C Fuguet,

Y. Durand,

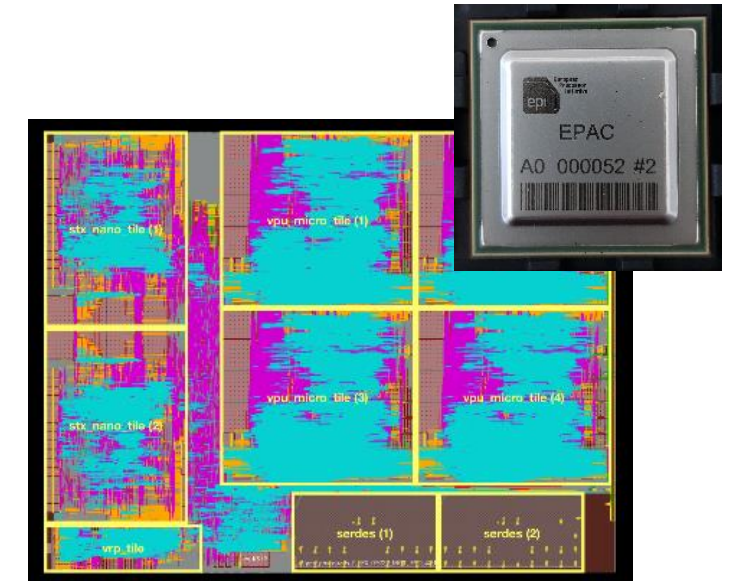
J. Fereyre,

I. Tahir,

R. Alidori,

T. Khandelwal,

N. Perbost



WHY EXTENDED PRECISION ?

- Radix-based floating point representation:

$$\left(\sum_{n=0}^{p-1} \textit{bit}_n \times 2^{-n} \right) \times 2^e$$

p-bit mantissa

k-bit exponent

WHY EXTENDED PRECISION ?

- Radix-based floating point representation:

$$\left(\sum_{n=0}^{p-1} \textit{bit}_n \times 2^{-n} \right) \times 2^e$$

p-bit mantissa

k-bit exponent

- VRP processor supports
 - $p=3 \rightarrow 512$ bits mantissa size
 - $k=2 \rightarrow 18$ bits exponent size
 - Specified at runtime (no recompilation)



So, why extending
computing precision?

WHY EXTENDED PRECISION ?

- Radix-based floating point representation:

$$\left(\sum_{n=0}^{p-1} \textit{bit}_n \times 2^{-n} \right) \times 2^e$$

p-bit mantissa

k-bit exponent

- VRP processor supports
 - $p=3 \rightarrow 512$ bits mantissa size
 - $k=2 \rightarrow 18$ bits exponent size
 - Specified at runtime (no recompilation)



So, why extending
computing precision?

- “1-bit LLMs Could Solve AI’s Energy Demands” ([1], IEEE Spectrum, 2024/05/30)

[1] <https://spectrum.ieee.org/1-bit-llm>

MOTIVATION: FLOATING POINT ISSUES

- Floating point arithmetic is done on fixed size hardware, which leads to rounding and absorption errors
- Example: approximation during mantissa alignment in floating point additions

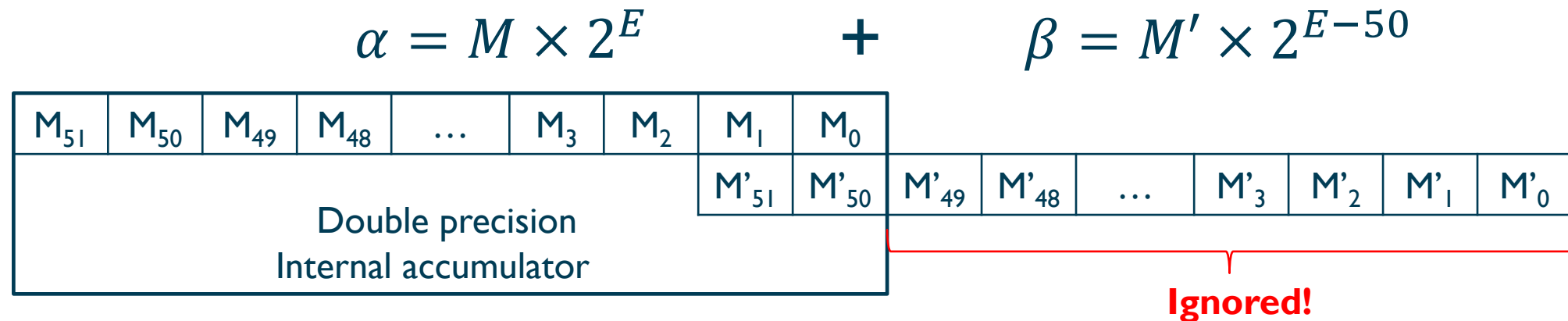
$$\alpha = M \times 2^E \quad + \quad \beta = M' \times 2^{E-50}$$

- $$\alpha = M \times 2^E \quad + \quad \beta = M' \times 2^{E-50}$$



MOTIVATION: FLOATING POINT ISSUES

- Floating point arithmetic is done on fixed size hardware, which leads to rounding and absorption errors
- Example: approximation during mantissa alignment in floating point additions



- ⇒ Floating point addition is not associative: $(\alpha + \beta) + \gamma \neq \alpha + (\beta + \gamma)$
- ⇒ Problematic for intensive calculations, especially iterative methods applied on large problems
- ⇒ Extended precision can alleviate these issues

APPLICATION DOMAIN: EXTENDED PRECISION FOR HPC

$$\tau_{yx} = -\mu \cdot \frac{dv_x}{dy}$$

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} = -f(r, \theta)$$

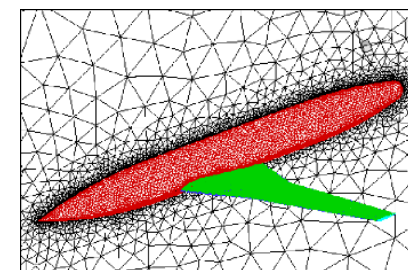
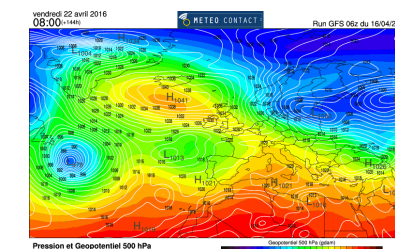
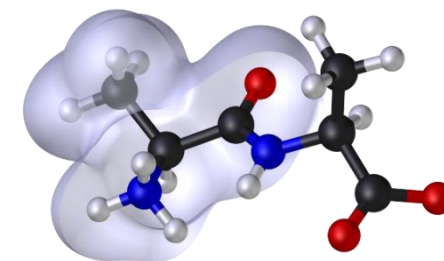
Starting point: symbolic manipulation of a set of PDEs from CFD, convection/diffusion, Electromagnetism, etc.

E.g. variational formulation



shutterstock.com · 151337624

- **Computational physics:** Blend of physics and chemistry, with applied mathematics, numerical linear algebra, numerical optimization, and parallel computing.
- Climate models, **weather prediction**, celestial physics, high energy physics, quantum mechanics
- More industrial topics such as electromagnetic, **fluid dynamics**.
- Some optimization algorithms (e.g. IPM barrier)

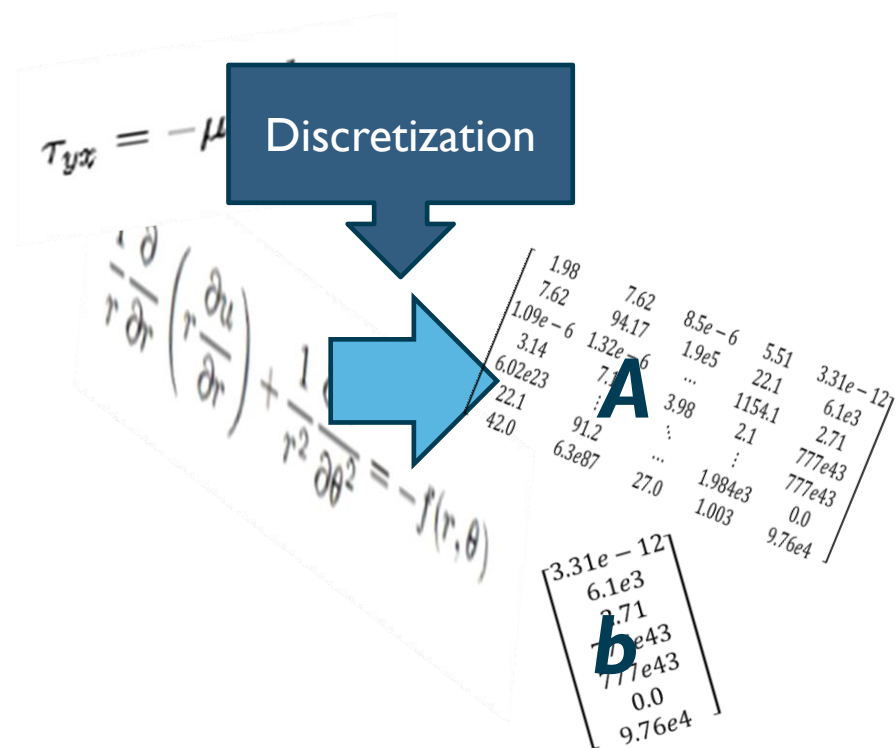


FOCUS ON LINEAR SOLVERS

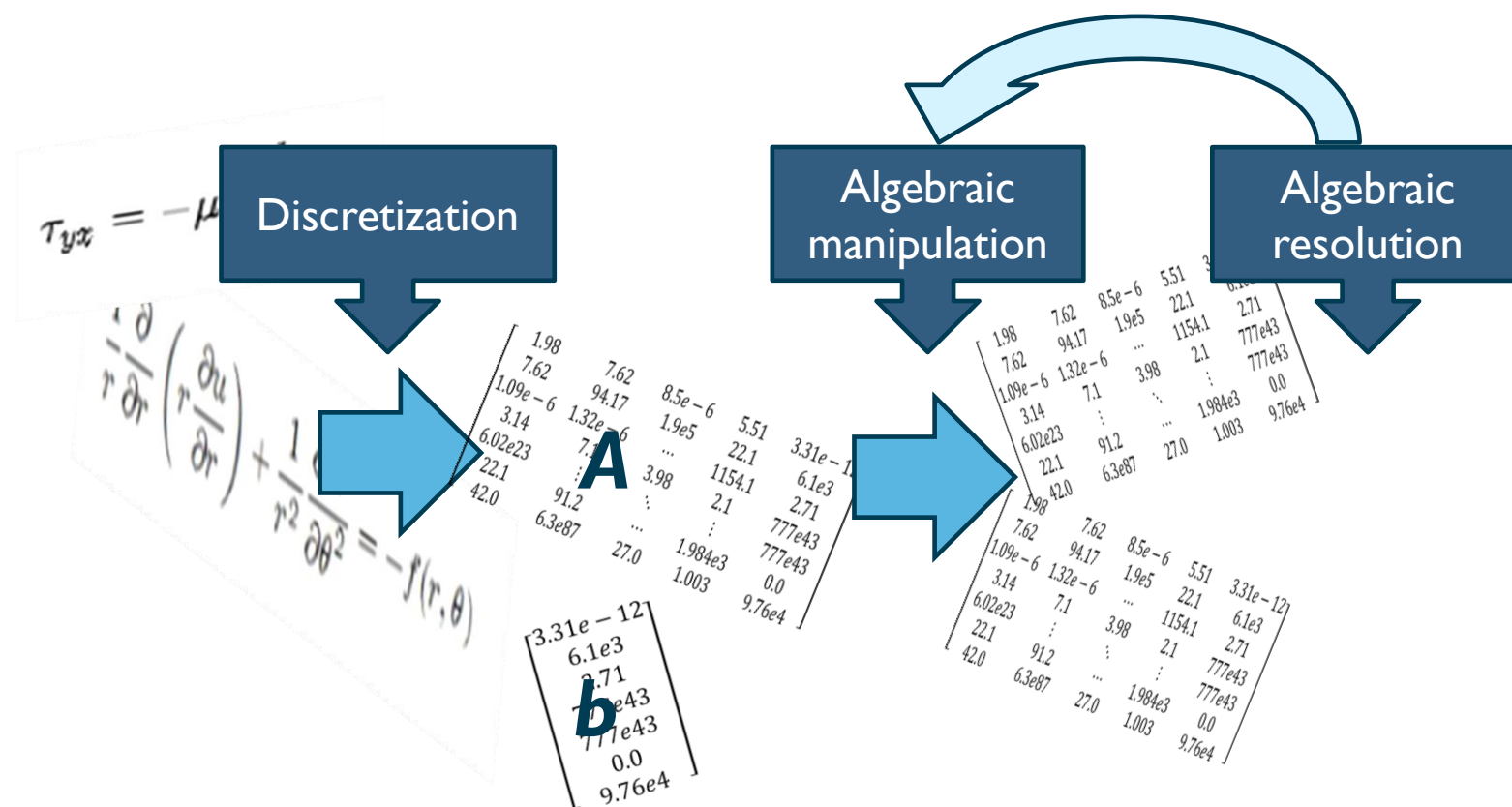
$$\tau_{yx} = -\mu \cdot \frac{dv_x}{dy}$$

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = -f(r, \theta)$$

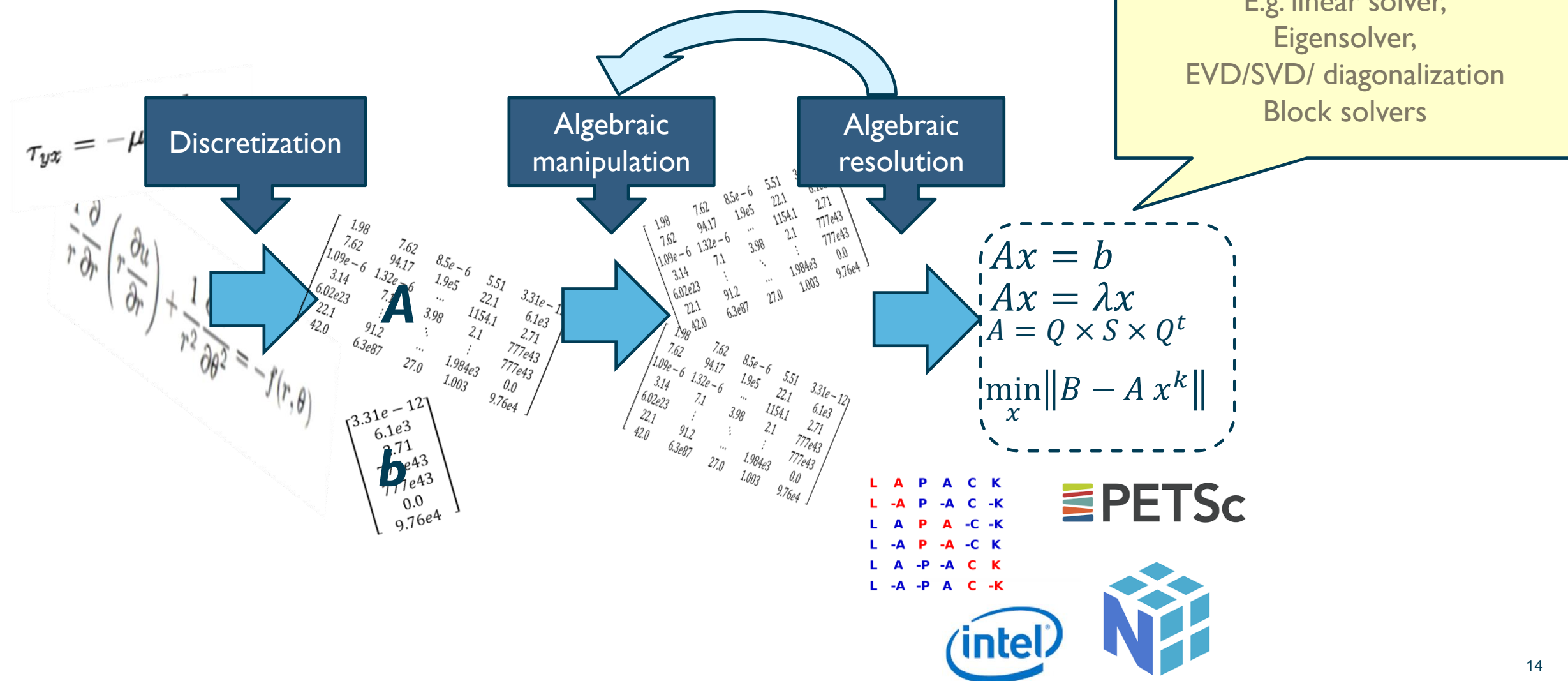
FOCUS ON LINEAR SOLVERS



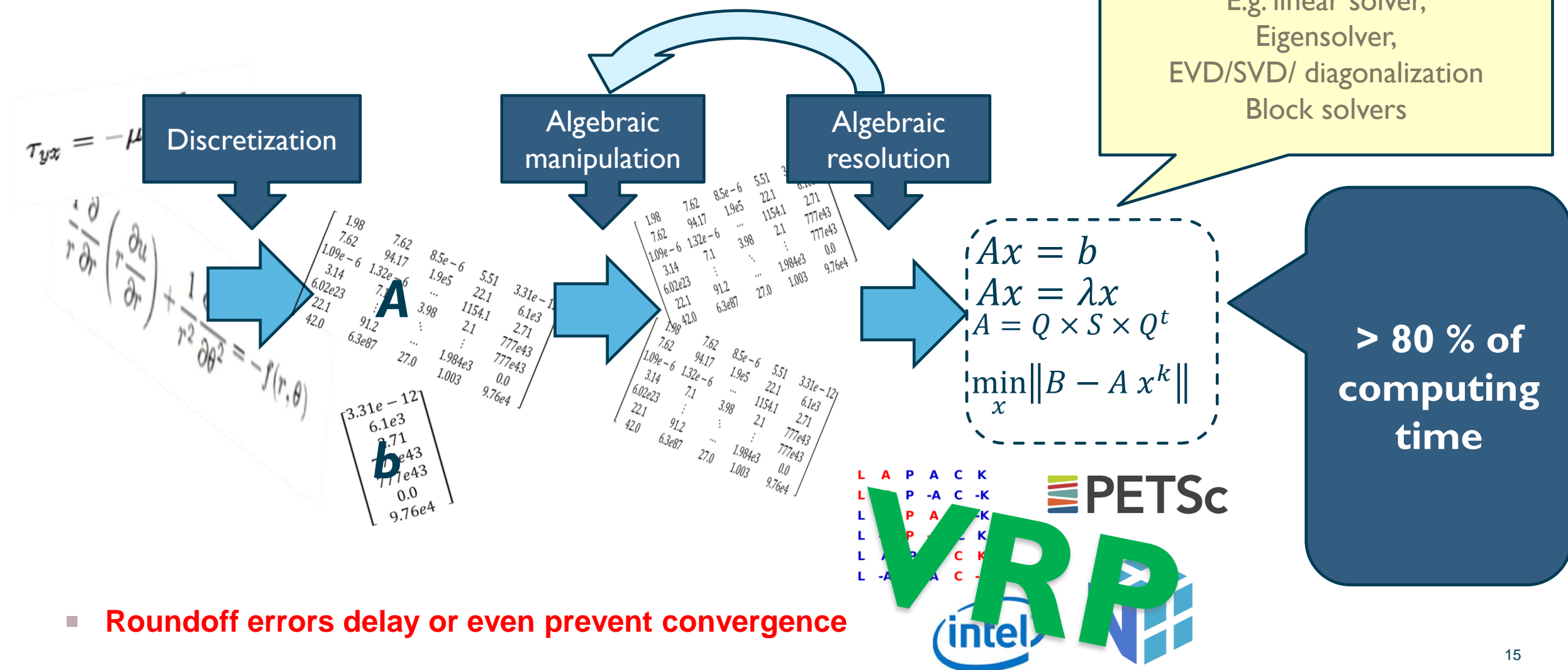
FOCUS ON LINEAR SOLVERS



FOCUS ON LINEAR SOLVERS



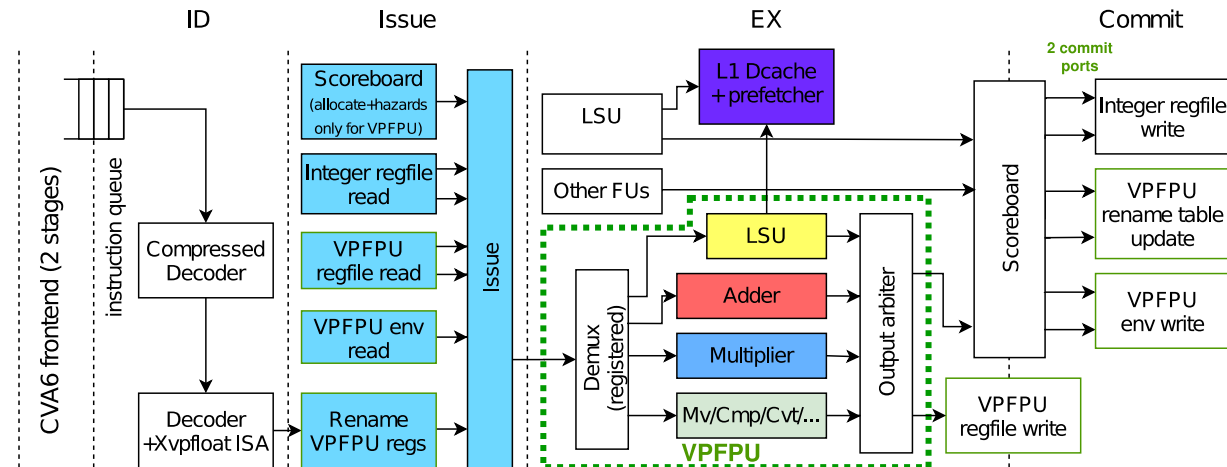
FOCUS ON LINEAR SOLVERS



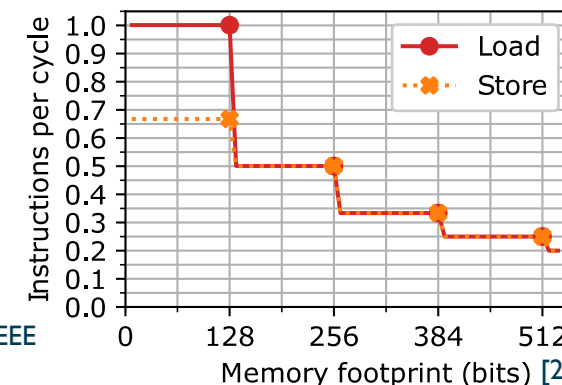
- Roundoff errors delay or even prevent convergence

VRP: A VARIABLE EXTENDED PRECISION ACCELERATOR FOR SCIENTIFIC COMPUTING APPLICATIONS

- Motivation: Software emulation (e.g. MPFR) too slow
- Goal: be application-agnostic and limited by memory bandwidth instead of arithmetic
- ⇒ Variable extended Precision Floating Point Unit integration in modified RISC-V CVA6 processor



- Main CVA6 modifications [2]:
 - 32 logical/64 physical ~540-bit registers: 512b mantissa + 18b exponent + 1b sign + flags
 - New high-perf L1 cache (open sourced)
 - A custom VPFPU with different operators whose latency depends on precision





XVPFLOAT RISC-V ISA EXTENSION

- Opensource ISA extension (TRISTAN project) for dynamically variable precision computation

⇒ **No recompilation when changing precision**

Mnemonic	Operation
PGER rt, ea	rt = ea
PSER et, ra	et = ra
PLE pt, evpi, ra{, #index}	vpfloat<evpi> *tab = ra pt = tab[index*stride]
PSE pb, evpi, ra{, #index}	vpfloat<evpi> *tab = ra tab[index*stride] = pb
PMV.P.P pt, pa	pt = pa
PMV.Pfield.X rt, pa	rt = pa[field]
PMV.X.Pfield pt, ra	pt[field] = ra
PCVT.P.H rt, pa, efpi	rt = (float16_t) pa
PCVT.P.F rt, pa, efpi	rt = (float32_t) pa
PCVT.P.D rt, pa, efpi	rt = (float64_t) pa
PCVT.H.P pt, ra	pt = (float16_t) ra.H[0]
PCVT.F.P pt, ra	pt = (float32_t) ra.W[0]
PCVT.D.P pt, ra	pt = (float64_t) ra
PADD pt, pa, pb, eci	pt = pa+pb
PSUB pt, pa, pb, eci	pt = pa-pb
PRND pt, pa, eci	pt = pa+0.0
PMUL pt, pa, pb, eci	pt = pa*pb
PCMP.EQ rt, pa, pb	rt = pa == pb ? 1 : 0
PCMP.NEQ rt, pa, pb	rt = pa != pb ? 1 : 0
PCMP.GT rt, pa, pb	rt = pa > pb ? 1 : 0
PCMP.LT rt, pa, pb	rt = pa < pb ? 1 : 0
PCMP.GEC rt, pa, pb	rt = pa >= pb ? 1 : 0
PCMP.LEQ rt, pa, pb	rt = pa <= pb ? 1 : 0



XVPFLOAT RISC-V ISA EXTENSION

- Opensource ISA extension (TRISTAN project) for dynamically variable precision computation

⇒ **No recompilation when changing precision**

- Instructions parametrized with in-memory and output precision

PMUL pt, pa, pb, eci

pt=pa*pb;

Env register includes:

- Output precision (in bits)
- Rounding mode

Mnemonic	Operation
PGER rt, ea	rt = ea
PSER et, ra	et = ra
PLE pt, evpi, ra{, #index}	vpfloat<evpi> *tab = ra pt = tab[index*stride]
PSE pb, evpi, ra{, #index}	vpfloat<evpi> *tab = ra tab[index*stride] = pb
PMV.PP pt, pa	pt = pa
PMV.Pfield.X rt, pa	rt = pa[field]
PMV.X.Pfield pt, ra	pt[field] = ra
PCVT.PH rt, pa, efpi	rt = (float16_t) pa
PCVT.PF rt, pa, efpi	rt = (float32_t) pa
PCVT.PD rt, pa, efpi	rt = (float64_t) pa
PCVT.H.P pt, ra	pt = (float16_t) ra.H[0]
PCVT.F.P pt, ra	pt = (float32_t) ra.W[0]
PCVT.D.P pt, ra	pt = (float64_t) ra
PADD pt, pa, pb, eci	pt = pa+pb
PSUB pt, pa, pb, eci	pt = pa-pb
PRND pt, pa, eci	pt = pa+0.0
PMUL pt, pa, pb, eci	pt = pa*pb
PCMP.EQ rt, pa, pb	rt = pa == pb ? 1 : 0
PCMP.NEQ rt, pa, pb	rt = pa != pb ? 1 : 0
PCMP.GT rt, pa, pb	rt = pa > pb ? 1 : 0
PCMP.LT rt, pa, pb	rt = pa < pb ? 1 : 0
PCMP.GEC rt, pa, pb	rt = pa >= pb ? 1 : 0
PCMP.LEQ rt, pa, pb	rt = pa <= pb ? 1 : 0



XVPFLOAT RISC-V ISA EXTENSION

- Opensource ISA extension (TRISTAN project) for dynamically variable precision computation

⇒ **No recompilation when changing precision**

- Instructions parametrized with in-memory and output precision

PMUL pt, pa, pb, eci

pt=pa*pb;

- Indexed load/store instructions

PLE pt, evpi, ra{,#index}

vpfloat<evpi> *tab=ra; pt=tab[index];

- In-memory IEEE-754 2008 extendable format

Env register includes:

- Output precision (in bits)
- Rounding mode

Env register includes:

- In-memory size (in bits)
- Exponent size (in bits)
- Rounding mode
- Stride

Mnemonic	Operation
PGER rt, ea	rt = ea
PSER et, ra	et = ra
PLE pt, evpi, ra{,#index}	vpfloat<evpi> *tab = ra pt = tab[index*stride]
PSE pb, evpi, ra{,#index}	vpfloat<evpi> *tab = ra tab[index*stride] = pb
PMV.PP pt, pa	pt = pa
PMV.Pfield.X rt, pa	rt = pa[field]
PMV.X.Pfield pt, ra	pt[field] = ra
PCVT.PH rt, pa, efpi	rt = (float16_t) pa
PCVT.PF rt, pa, efpi	rt = (float32_t) pa
PCVT.PD rt, pa, efpi	rt = (float64_t) pa
PCVT.H.P pt, ra	pt = (float16_t) ra.H[0]
PCVT.F.P pt, ra	pt = (float32_t) ra.W[0]
PCVT.D.P pt, ra	pt = (float64_t) ra
PADD pt, pa, pb, eci	pt = pa+pb
PSUB pt, pa, pb, eci	pt = pa-pb
PRND pt, pa, eci	pt = pa+0.0
PMUL pt, pa, pb, eci	pt = pa*pb
PCMP.EQ rt, pa, pb	rt = pa == pb ? 1 : 0
PCMP.NEQ rt, pa, pb	rt = pa != pb ? 1 : 0
PCMP.GT rt, pa, pb	rt = pa > pb ? 1 : 0
PCMP.LT rt, pa, pb	rt = pa < pb ? 1 : 0
PCMP.GEC rt, pa, pb	rt = pa >= pb ? 1 : 0
PCMP.LEQ rt, pa, pb	rt = pa <= pb ? 1 : 0



XVPFLOAT RISC-V ISA EXTENSION

- Opensource ISA extension (TRISTAN project) for dynamically variable precision computation

⇒ **No recompilation when changing precision**

- Instructions parametrized with in-memory and output precision

PMUL pt, pa, pb, eci

pt=pa*pb;

- Indexed load/store instructions

PLE pt, evpi, ra{,#index}

vpfloat<evpi> *tab=ra; pt=tab[index];

Env register includes:

- Output precision (in bits)
- Rounding mode

Env register includes:

- In-memory size (in bits)
- Exponent size (in bits)
- Rounding mode
- Stride

- In-memory IEEE-754 2008 extendable format

- conversions, arithmetic (+,-,*), load/store, comparison

Mnemonic	Operation
PGER rt, ea	rt = ea
PSER et, ra	et = ra
PLE pt, evpi, ra{, #index}	vpfloat<evpi> *tab = ra pt = tab[index*stride]
PSE pb, evpi, ra{, #index}	vpfloat<evpi> *tab = ra tab[index*stride] = pb
PMV.P.P pt, pa	pt = pa
PMV.Pfield.X rt, pa	rt = pa[field]
PMV.X.Pfield pt, ra	pt[field] = ra
PCVT.P.H rt, pa, efpi	rt = (float16_t) pa
PCVT.P.F rt, pa, efpi	rt = (float32_t) pa
PCVT.P.D rt, pa, efpi	rt = (float64_t) pa
PCVT.H.P pt, ra	pt = (float16_t) ra.H[0]
PCVT.F.P pt, ra	pt = (float32_t) ra.W[0]
PCVT.D.P pt, ra	pt = (float64_t) ra
PADD pt, pa, pb, eci	pt = pa+pb
PSUB pt, pa, pb, eci	pt = pa-pb
PRND pt, pa, eci	pt = pa+0.0
PMUL pt, pa, pb, eci	pt = pa*pb
PCMP.EQ rt, pa, pb	rt = pa == pb ? 1 : 0
PCMP.NEQ rt, pa, pb	rt = pa != pb ? 1 : 0
PCMP.GT rt, pa, pb	rt = pa > pb ? 1 : 0
PCMP.LT rt, pa, pb	rt = pa < pb ? 1 : 0
PCMP.GEC rt, pa, pb	rt = pa >= pb ? 1 : 0
PCMP.LEQ rt, pa, pb	rt = pa <= pb ? 1 : 0



XVPFLOAT RISC-V ISA EXTENSION

- Opensource ISA extension (TRISTAN project) for dynamically variable precision computation

⇒ **No recompilation when changing precision**

- Instructions parametrized with in-memory and output precision

PMUL pt, pa, pb, eci

pt=pa*pb;

- Indexed load/store instructions

PLE pt, evpi, ra{,#index}

vpfloat<evpi> *tab=ra; pt=tab[index];

Env register includes:

- Output precision (in bits)
- Rounding mode

Env register includes:

- In-memory size (in bits)
- Exponent size (in bits)
- Rounding mode
- Stride

- In-memory IEEE-754 2008 extendable format

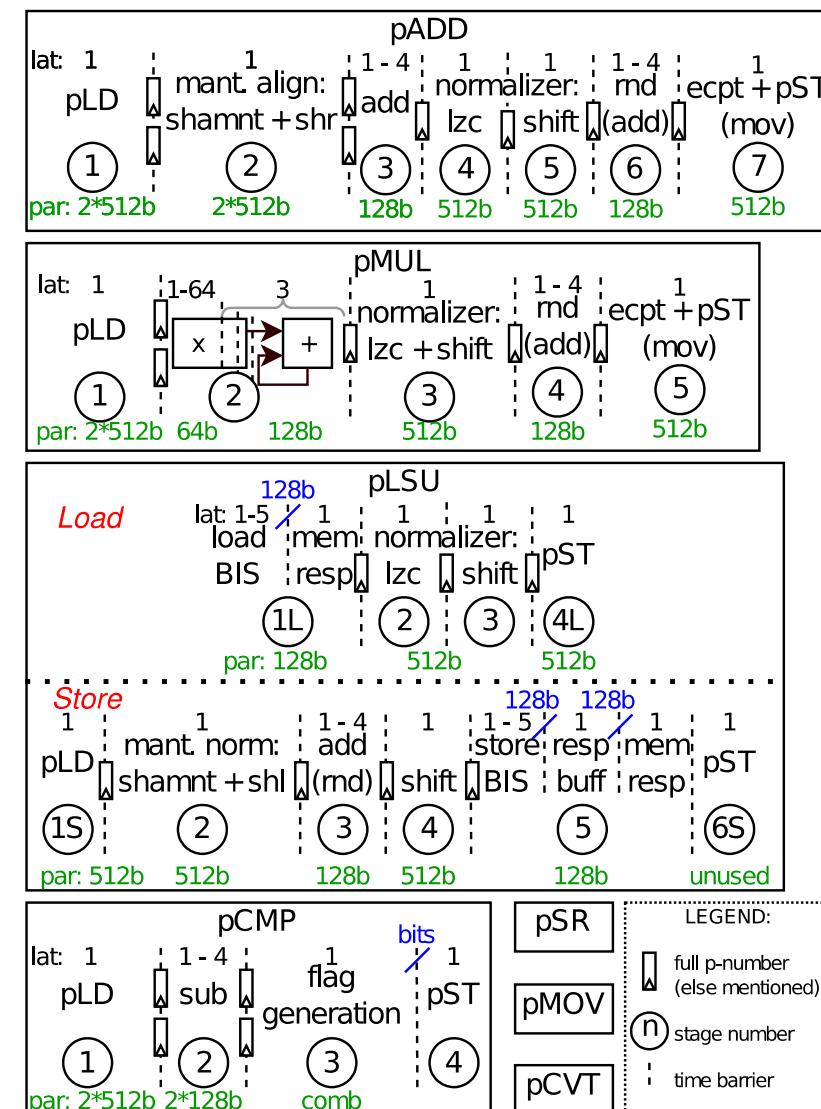
- conversions, arithmetic (+,-,*), load/store, comparison
- Compiler WIP (Binutils ready, prototype LLVM+Clang with *vpfloat* C type)

Mnemonic	Operation
PGER rt, ea	rt = ea
PSER et, ra	et = ra
PLE pt, evpi, ra{,#index}	vpfloat<evpi> *tab = ra pt = tab[index*stride]
PSE pb, evpi, ra{,#index}	vpfloat<evpi> *tab = ra tab[index*stride] = pb
PMV.PP pt, pa	pt = pa
PMV.Pfield.X rt, pa	rt = pa[field]
PMV.X.Pfield pt, ra	pt[field] = ra
PCVT.PH rt, pa, efpi	rt = (float16_t) pa
PCVT.PF rt, pa, efpi	rt = (float32_t) pa
PCVT.PD rt, pa, efpi	rt = (float64_t) pa
PCVT.H.P pt, ra	pt = (float16_t) ra.H[0]
PCVT.F.P pt, ra	pt = (float32_t) ra.W[0]
PCVT.D.P pt, ra	pt = (float64_t) ra
PADD pt, pa, pb, eci	pt = pa+pb
PSUB pt, pa, pb, eci	pt = pa-pb
PRND pt, pa, eci	pt = pa+0.0
PMUL pt, pa, pb, eci	pt = pa*pb
PCMP.EQ rt, pa, pb	rt = pa == pb ? 1 : 0
PCMP.NEQ rt, pa, pb	rt = pa != pb ? 1 : 0
PCMP.GT rt, pa, pb	rt = pa > pb ? 1 : 0
PCMP.LT rt, pa, pb	rt = pa < pb ? 1 : 0
PCMP.GEC rt, pa, pb	rt = pa >= pb ? 1 : 0
PCMP.LEQ rt, pa, pb	rt = pa <= pb ? 1 : 0

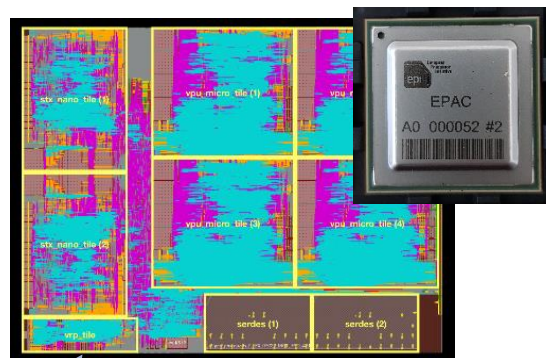
VPFPU: VARIABLE EXTENDED PRECISION FPU

■ Main take away:

- The VPFPU supports the FP operations defined by the xvpfloat modified ISA.
- Each multi-cycle pipeline stage corresponds to an iterative operation on the floating-point mantissa.
- 7 VPFPU functional units
 - Working iteratively on 64/128b chunks
 - Performance depends on precision
- Some pipeline stages work on full width to minimize the computation latency

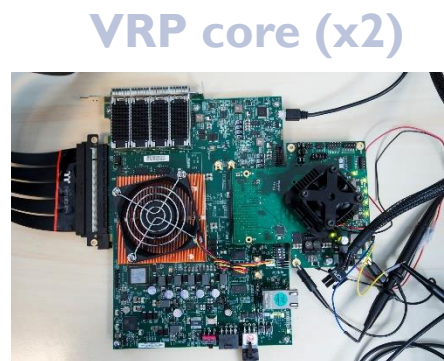


HARDWARE PROTOTYPES



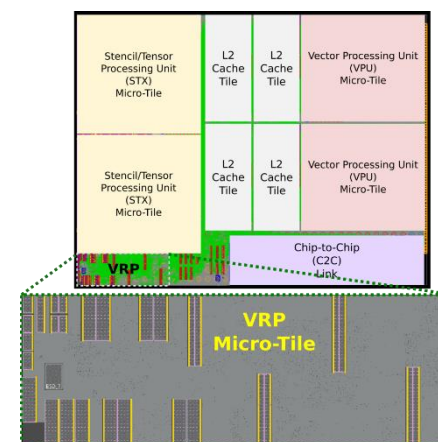
EPACTC 1.0

VRP core GF22FDX
Q2 2021



VRP core (x2)

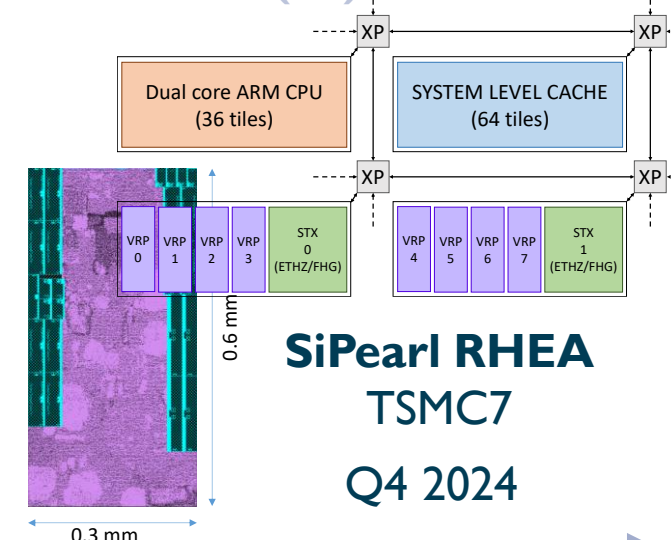
Dual-core VRP tile
FPGA
Q3 2021



EPACTC 1.5

VRP core GF22FDX
Q3 2022

VRP core (x8) **Jupiter HPC**



SiPearl RHEA
TSMC7
Q4 2024

EPACTC 1.0 (VRP features)

- Up to 256 bits of significand
- UNUM Type I memory format
- 2 dynamic data formats
- Clock frequency : 1 GHz

EPACTC 1.5 (VRP 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 GHz

RHEA (VRP features)

- Same features as in EPACTC 1.5
 - Two tiles with 4 VRP cores each
 - Clock frequency: 1.1 GHz
- FPGA prototype**
- 2 VRP cores
 - Clock frequency: 83 MHz

SOFTWARE STACK: SOLVER EXAMPLE

Application
(executed on host or natively on the VRP)

offloading

Solver
(VPFloat software)

(V)BLAS routine
(assembly)

Runtime

SW Emulation
(MPFR)

SW Emulation
(Spike)

HW
(FPGA/ASIC)

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

```
...
```

```
Nbiter = cg_vp(precision, Ndiag, X, A, B, tol);
```

```
// 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, ...)
```

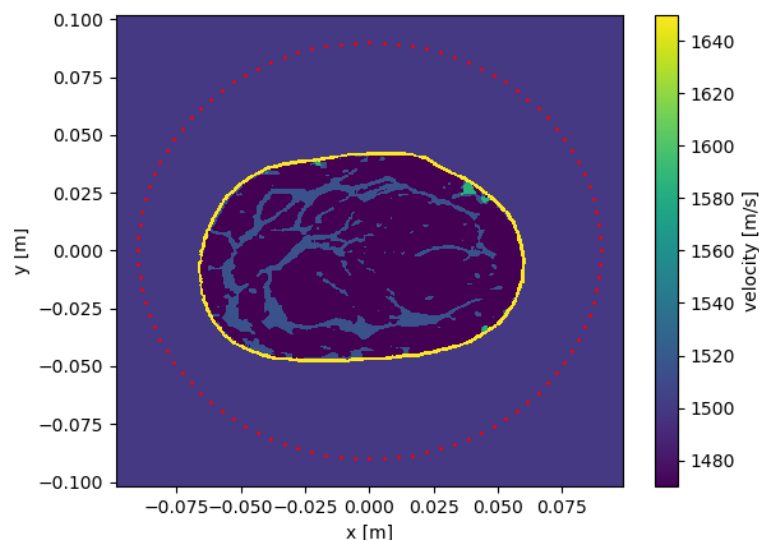
```
// y = A*x
```

```
void VBLAS::vgemv(int precision, ...){  
  pser_ec(precision, EC0);           // compute precision  
  pser_ev(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)
```

CASE STUDIES: CG, BICG, BLOCK-BICG SOLVERS

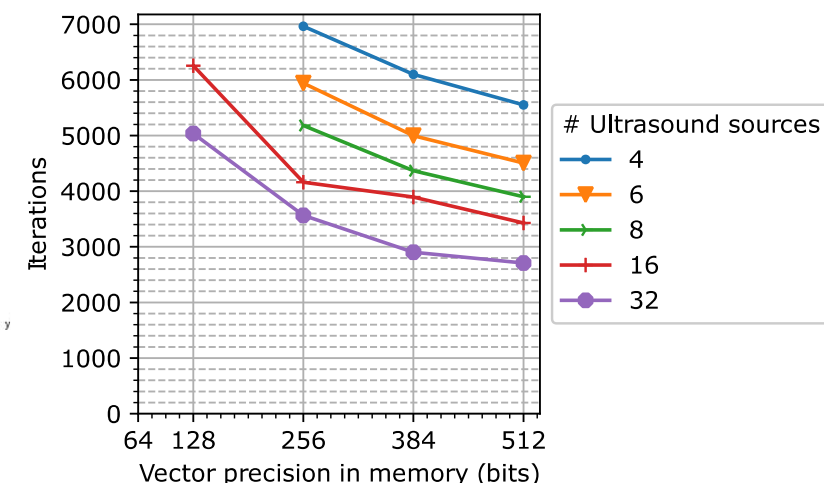
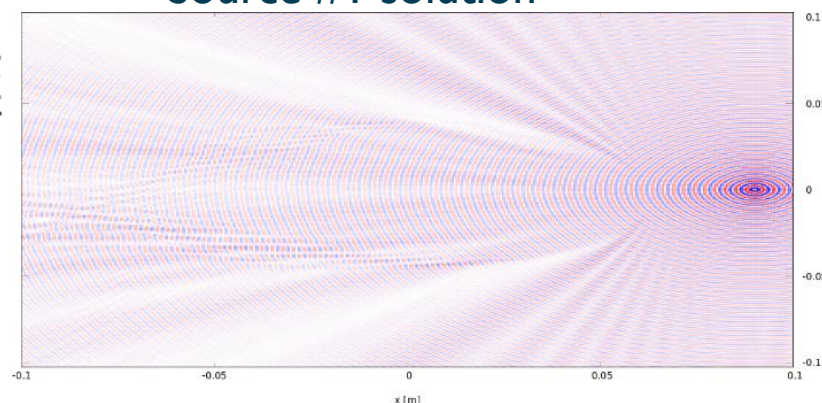
- Ultrasonic frequency-domain breast tomography (Block-BiCG) (SoA: time-domain)

$$AX = B$$

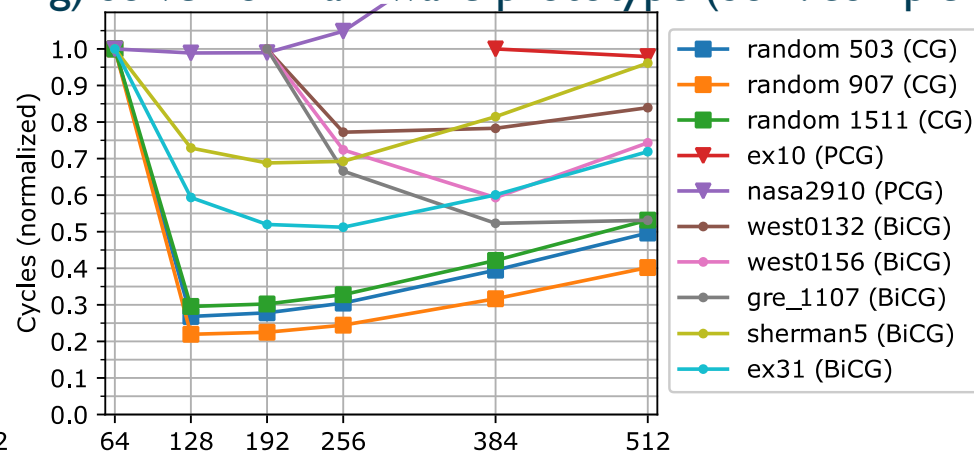
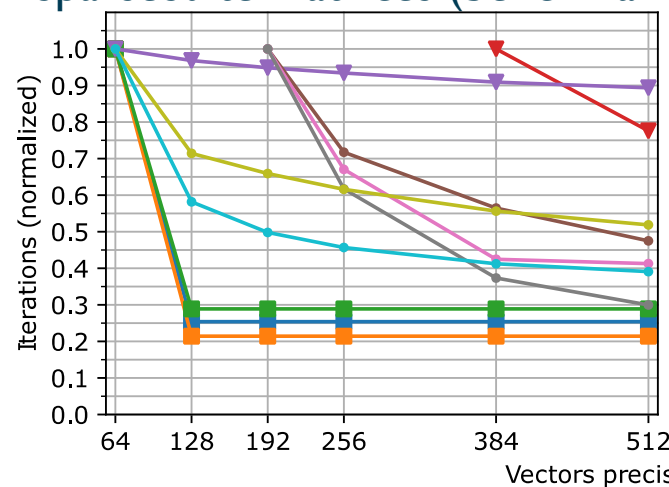


181k x 181k matrix

Source #1 solution



- SparseSuite matrices (benchmarking) solved on hardware prototype (SoA: complex preconditioners)



$$Ax = b$$

WHAT'S NEXT ?

Applications

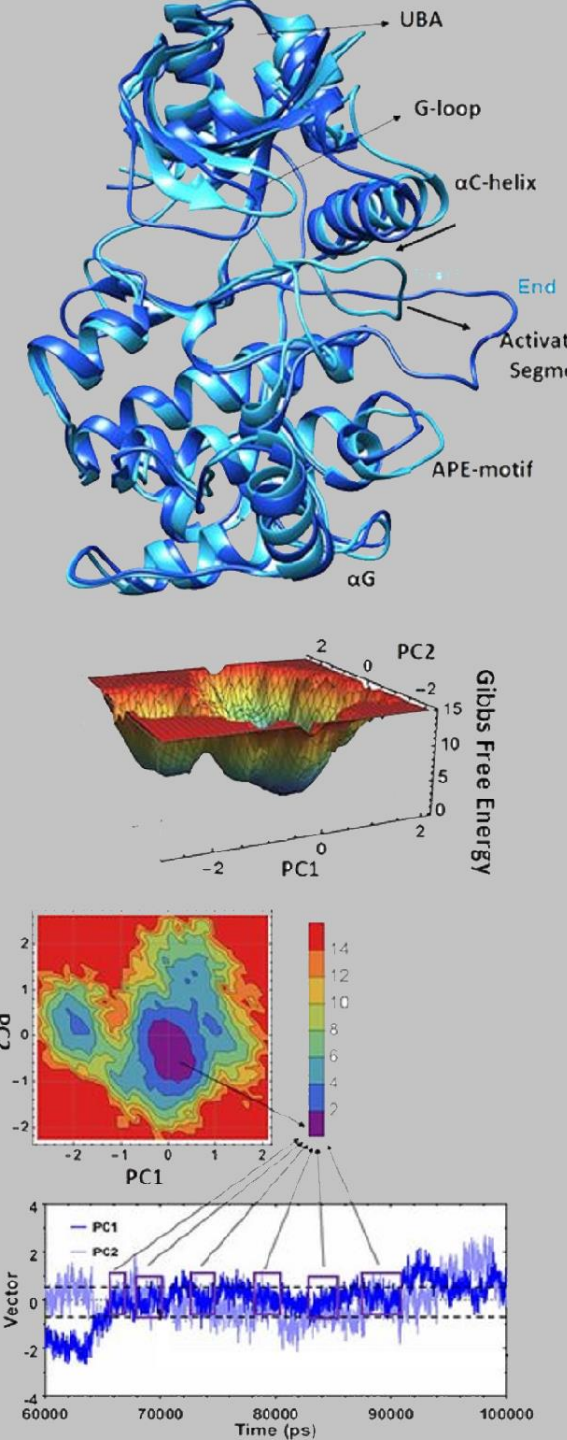
- Eigensolvers => $O(\text{eigenvalues})$ for fewer operations

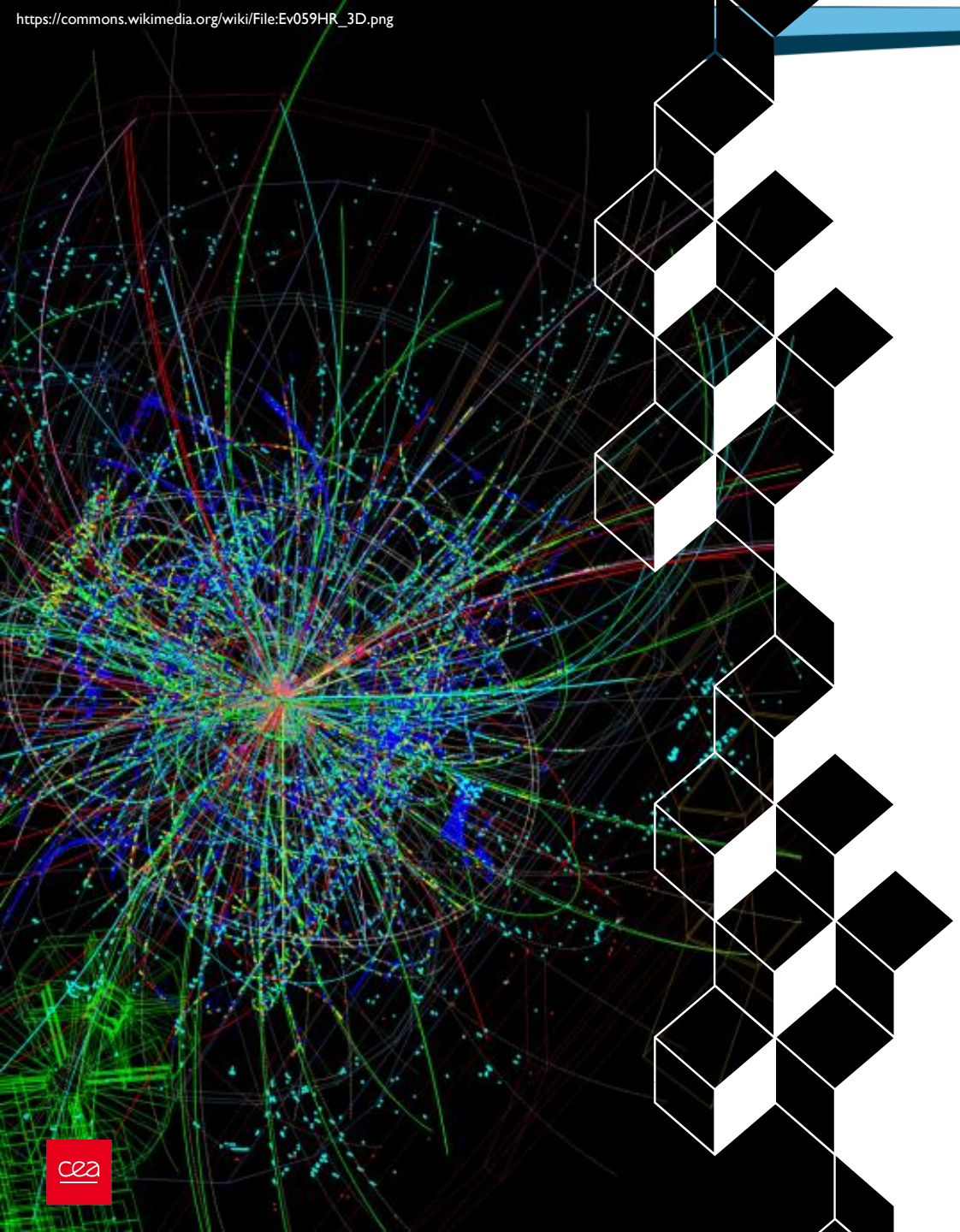
Software

- SDK based on Eigen+MPFR
- C compiler supporting vfloat variable precision type

Hardware

- SiPearl Rhea tapeout with 8x VRP cores (Q4 2024)
- Design on the next VRP generation (VXP):
 - OoO execution
 - Better support for sparse matrices





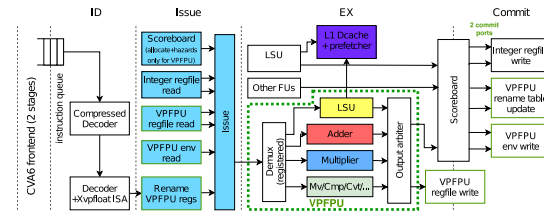
OPEN QUESTIONS

Applications

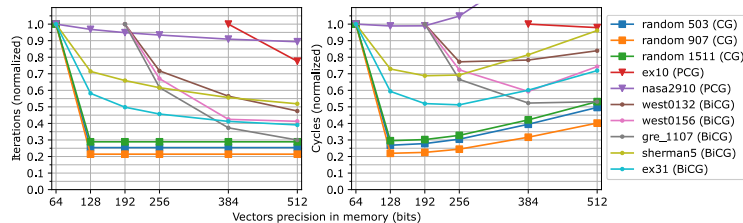
- Impact of higher precision of result ?
- How to choose the right precision ?
- Granularity of variable precision ?
- ⇒ Open to collaborations, working on building a community of potential users

CONCLUSION

- We proposed a hardware accelerator with a full software stack based on RISC-V

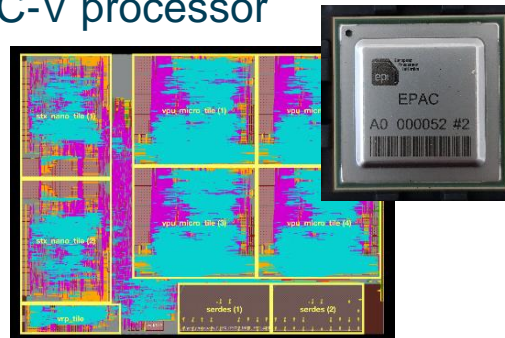


- We demonstrated the benefits of extended precision



- Optimized hardware based on RISC-V processor

- Can be used in any application
- Silicon-proven



PUBLICATIONS

- E. Guthmuller *et al.*, "Xvpfloat: RISC-V ISA Extension for Variable Extended Precision Floating Point Computation," in *IEEE Transactions on Computers*, doi: 10.1109/TC.2024.3383964.
- Hoffmann, Alexandre & Durand, Yves & Fereyre, Jérôme. "Accelerating Spectral Elements Method with Extended Precision: A Case Study." 2024, *International Journal of Applied Physics and Mathematics*. 14. 45-58. 10.17706/ijapm.2024.14.2.45-58.
- Y. Durand, E. Guthmuller, C. Fuguet, J. Fereyre, A. Bocco and R. Alidori, "Accelerating Variants of the Conjugate Gradient with the Variable Precision Processor," *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, Lyon, France, 2022, pp. 51-57, doi: 10.1109/ARITH54963.2022.00017.
- T. T. Jost, Y. Durand, C. Fabre, A. Cohen and F. Pérot, "Seamless Compiler Integration of Variable Precision Floating-Point Arithmetic," *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Seoul, Korea (South), 2021, pp. 65-76, doi: 10.1109/CGO51591.2021.9370331.
- Tiago Jost, Yves Durand, Christian Fabre, Albert Cohen, and Frédéric Pétrot. "VP Float: First Class Treatment for Variable Precision Floating Point Arithmetic." 2020, in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT '20)*. Association for Computing Machinery, New York, NY, USA, 355–356. <https://doi.org/10.1145/3410463.3414660>
- A. Bocco, Y. Durand and F. de Dinechin, "Dynamic Precision Numerics Using a Variable-Precision UNUM Type I HW Coprocessor," *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, Kyoto, Japan, 2019, pp. 104-107, doi: 10.1109/ARITH.2019.00028.
- A. Bocco, T. T. Jost, A. Cohen, F. de Dinechin, Y. Durand and C. Fabre, "Byte-Aware Floating-point Operations through a UNUM Computing Unit," *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, Cuzco, Peru, 2019, pp. 323-328, doi: 10.1109/VLSI-SoC.2019.8920387.

EPI FORUM



EuroHPC
Joint Undertaking

PLATINUM SPONSORS



EVIDEN



GOLD SPONSORS

