Residual Replacement in Mixed-Precision Iterative Refinement for Sparse Linear Systems

Hartwig Anzt, Goran Flegar, Vedran Novakovic,

Enrique S. Quintana-Ortí, Andres E. Tomás

Mixed Precision Iterative Refinement

- Given a linear system $A\hat{x} = \hat{b}$, *iterative refinement* (IR) is a technique to improve the accuracy of an initial solution $\hat{x}_0$:

$$
\begin{aligned}
&\textbf{for } k := 0, 1, 2, \ldots \\
&\qquad r := \hat{b} - A\hat{x}_k && \text{Residual calculation} \\
&\text{Solve } Ay = r \text{ for } y && \text{Inner solver} \\
&\qquad \hat{x}_{k+1} := \hat{x}_k + y && \text{Solution update}
\end{aligned}
$$

- Any inner solver: dense/sparse factorization…
  even an iterative Krylov(-type) solver
- In machine precision u, provided $uk(A) \leq 1$, IR eventually produces an accurate solution to full precision u

Mixed Precision Iterative Refinement

- On many architectures, IR can be efficiently combined with mixed precision (single-double, half-double, half-single)

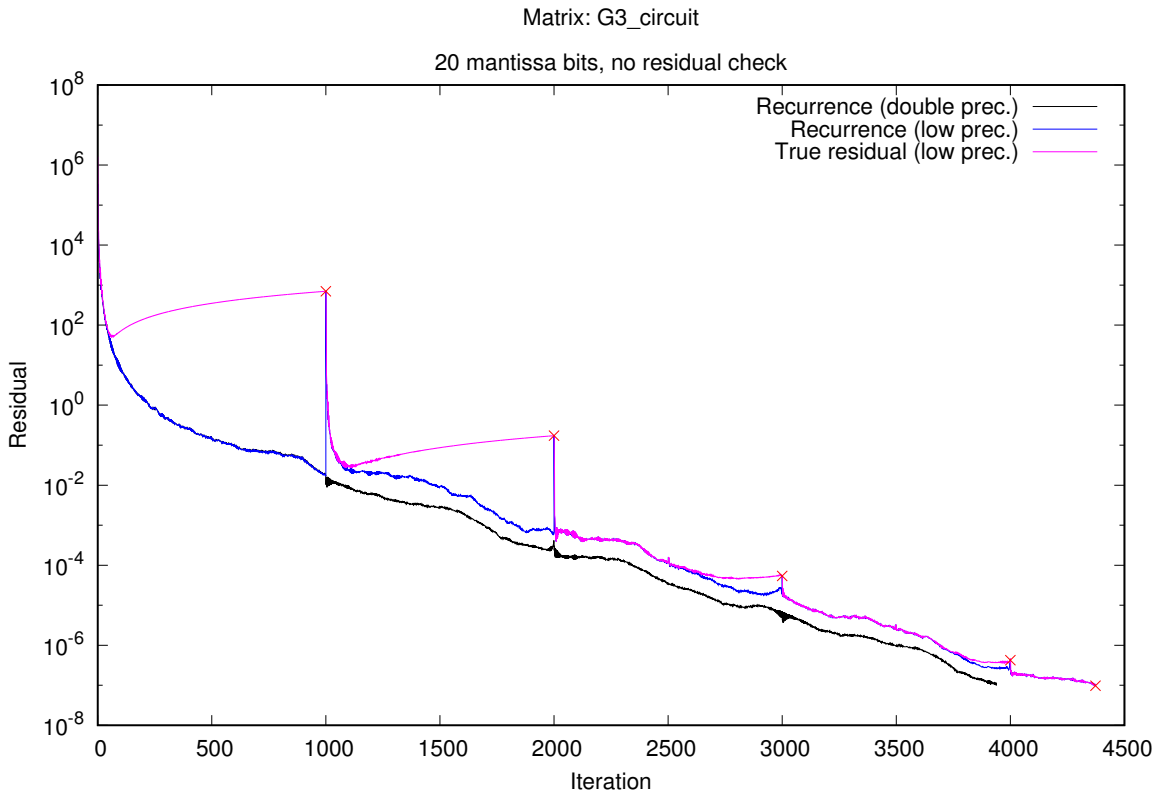
for $k := 0, 1, 2, \ldots$

$r := \hat{b} - A\hat{x}_k$	Residual calculation	Extended precision
Solve $Ay = r$ for y	Inner solver	Reduced precision
$\hat{x}_{k+1} := \hat{x}_k + y$	Solution update	Extended precision

- Most of the cost is in the inner solver
- Accuracy is improved by the outer refinement process

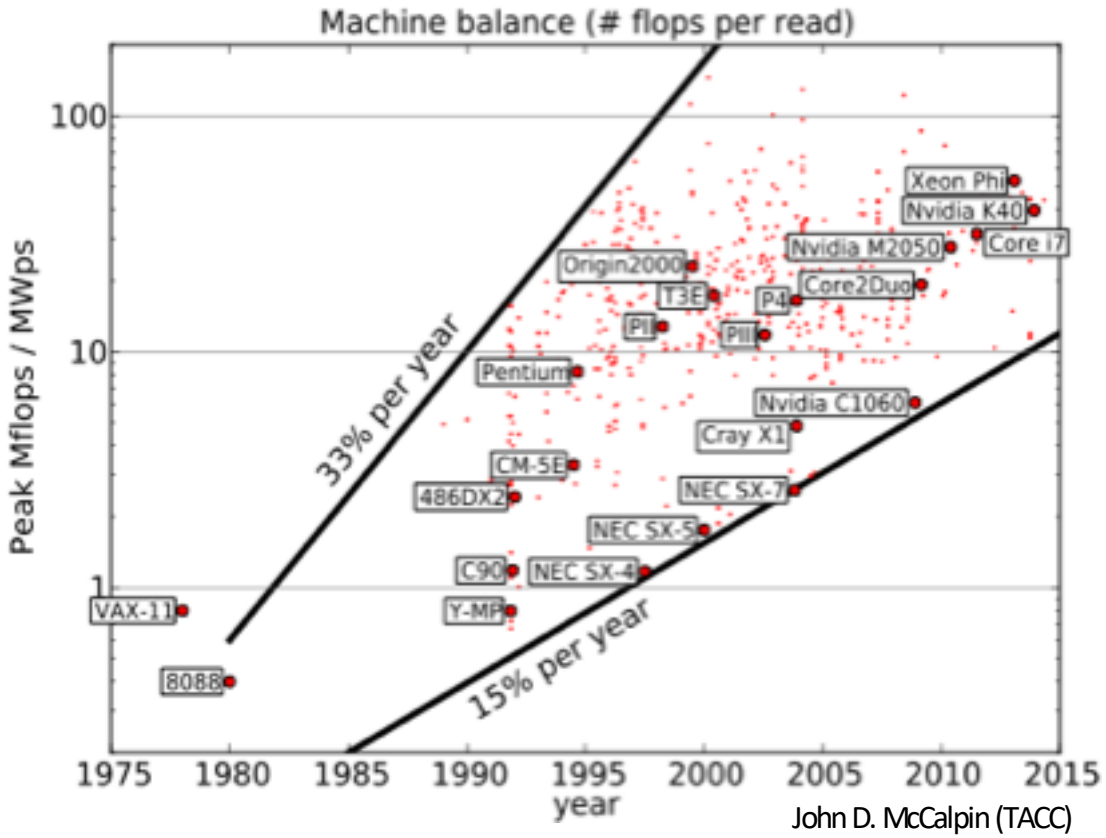
Mixed Precision Iterative Refinement

- Can MPIR be efficiently combined with an iterative Krylov inner solver?
  - Maintain **convergence rate** by avoiding numerical pitfalls in the recurrence residual due to finite precision



Mixed Precision Iterative Refinement

- What are the potential gains of mixed precision?
  - For Krylov solvers applied to sparse linear systems, **the theoretical cost/energy/time is in moving data, not in arithmetic**



John D. McCalpin (TACC)

Outine

- Residual replacement for Krylov solvers
- Cost model
- Explicit residual deviation with MPIR
- Cost evaluation
- Customized precision via mantissa segmentation

Residual Replacement for Krylov Solvers

- Preconditioned Conjugate Gradient (PCG)

$r_0 := b - Ax_0,\ z_0 := M^{-1}r_0,\ d_0 := z_0,\ \beta_0 := r_0^T z_0,$	Initializations
$\tau_0 :=\| r_0 \|_2,\ j := 0$	
while $(\tau_j > \tau_{\max})$	**Iterative PCG solve**
$w_j := Ad_j$	SPMV
$\rho_j := \beta_j / d_j^T w_j$	DOT product
$x_{j+1} := x_j + \rho_j d_j$	AXPY
$r_{j+1} := r_j - \rho_j w_j$	AXPY
$z_{j+1} := M^{-1} r_{j+1}$	Preconditioning
$\beta_{j+1} := r_{j+1}^T z_{j+1}$	DOT product
$\alpha_j := \beta_{j+1} / \beta_j$	
$d_{j+1} := z_{j+1} + \alpha_j d_j$	AXPY-like
$\tau_{j+1} :=\| r_{j+1} \|_2$	2-norm
$j := j + 1$	
endwhile	

Residual Replacement for Krylov Solvers

- Preconditioned Conjugate Gradient (PCG)

$\tau_0 := \| r_0 \|_2,\ j := 0$
while $(\tau_j > \tau_{\max})$

$x_{j+1} := x_j + \rho_j d_j$

$\tau_{j+1} := \| r_{j+1} \|_2$
$j := j + 1$
endwhile

Residual Replacement for Krylov Solvers

- Preconditioned Conjugate Gradient (PCG)

$\tau_0 := \| r_0 \|_2,\ j := 0$
while $(\tau_j > \tau_{\max})$

$x_{j+1} := x_j + \rho_j d_j$
$r_{j+1} := r_j - \rho_j w_j$ Recurrence residual

$\tau_{j+1} := \| r_{j+1} \|_2$
$j := j + 1$
endwhile

Residual Replacement for Krylov Solvers

- Preconditioned Conjugate Gradient (PCG)

$\tau_0 := \| r_0 \|_2,\ j := 0$
while $(\tau_j > \tau_{\max})$

$x_{j+1} := x_j + \rho_j d_j$
$r_{j+1} := r_j - \rho_j w_j$ Recurrence residual

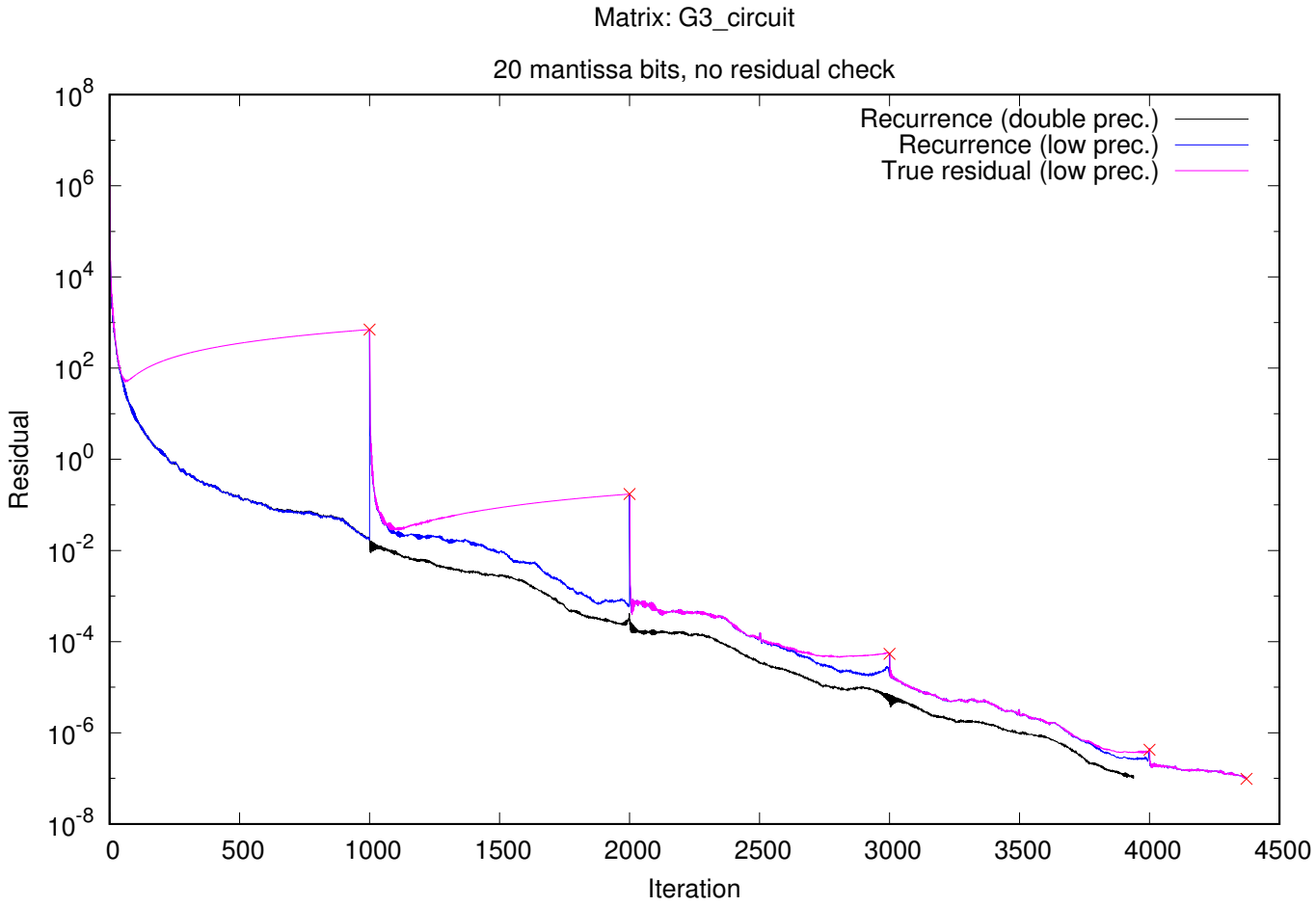
$\tau_{j+1} := \| r_{j+1} \|_2$
$j := j + 1$
endwhile

True residual

$$b - Ax_{j+1}$$

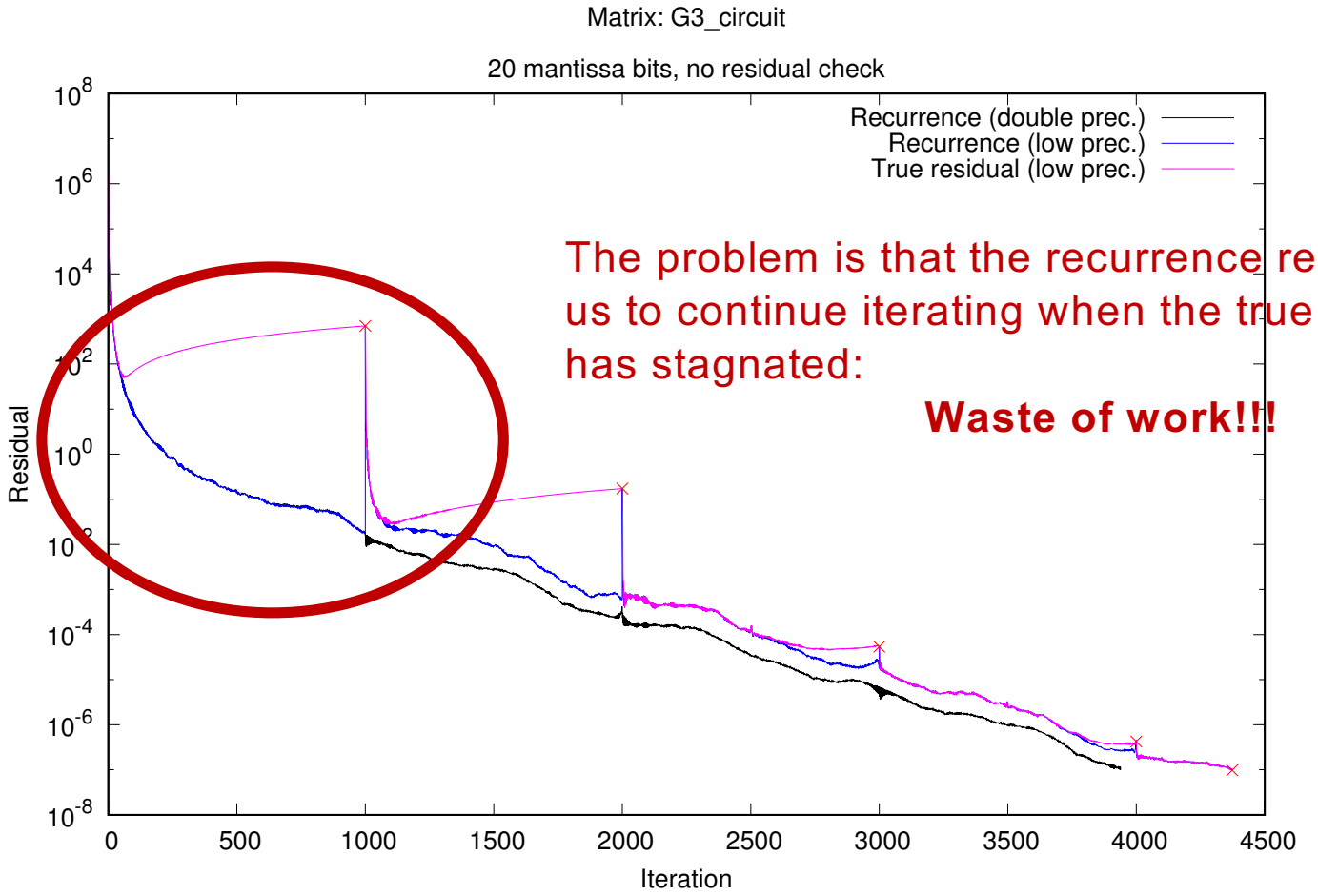
Residual Replacement for Krylov Solvers

- Divergence between **recurrence** vs **true** residuals



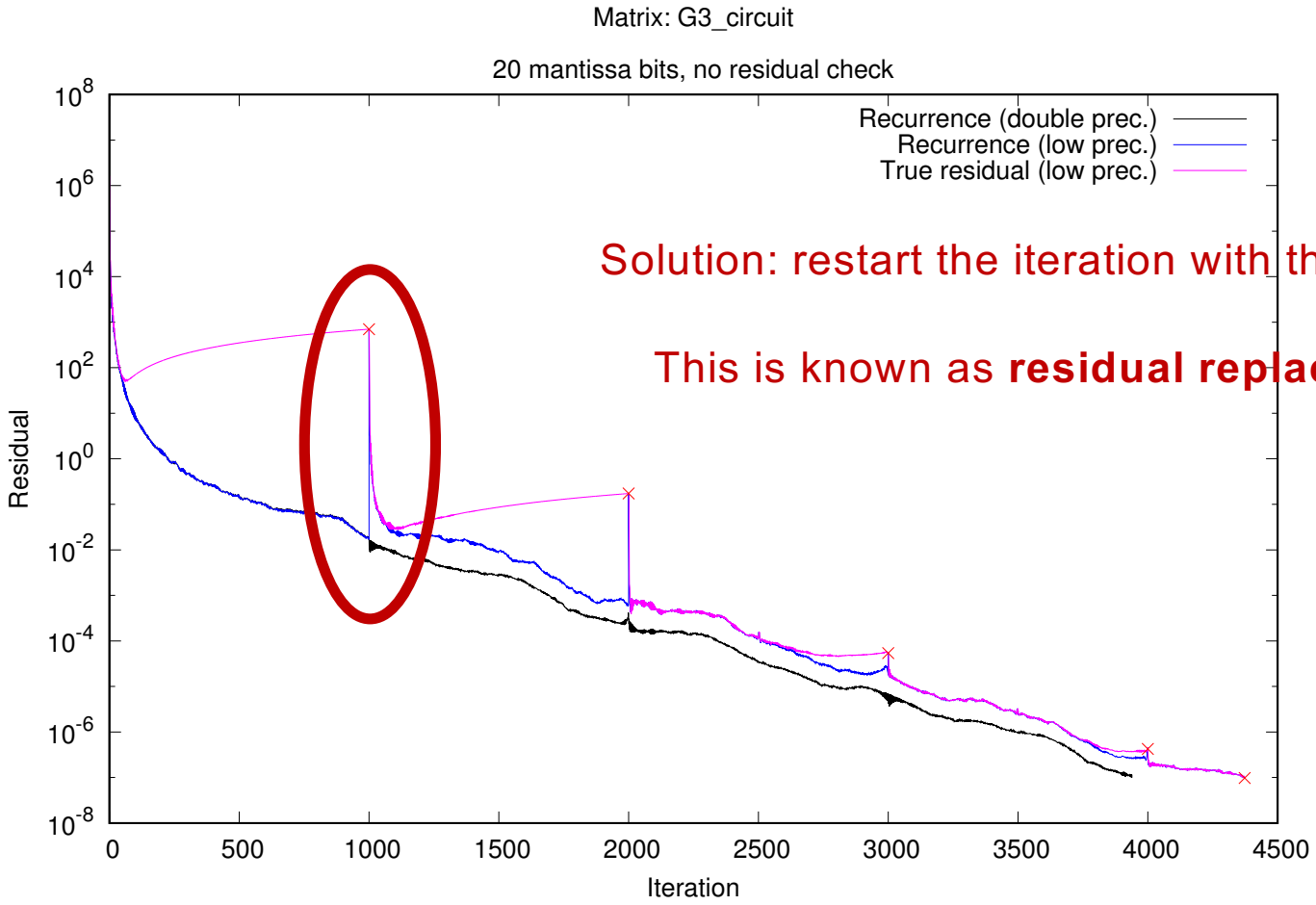
Residual Replacement for Krylov Solvers

- Finite precision causes divergence between recurrence vs true residuals



Residual Replacement for Krylov Solvers

- Finite precision causes divergence between recurrence vs true residuals



Solution: restart the iteration with the true residual

This is known as **residual replacement (RR)**

Residual Replacement for Krylov Solvers

- Divergence problem can be tackled via RR:
  - **Replace always** (at every iteration): 2 SpMV per iteration and may deteriorate the convergence of the iteration
  - **Replace periodically** (every t iterations): may deteriorate the convergence of the iteration
  - **Compute explicit deviation at every iteration and replace if needed:** 2 SpMV per iteration
  - **Estimate deviation and replace if needed:**

H. A. Van der Vorst, Q. Ye. "Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals." SIAM J. Sci. Comput., 22(3), 2000

Residual Replacement for Krylov Solvers

H. A. Van der Vorst, Q. Ye. "Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals." SIAM J. Sci. Comput., 22(3), 2000

- **Van der Vorst and Ye** (VY), 2000. Keep track of accumulated deviation:

$$d_0 = d_{init} := u_r(||r_0|| + N||A||||x_0||),$$
$$d_{j+1} := d_j + u_r(||r_j|| + N||A||||\tilde{x}_j||), \quad j = 0, 1, 2, \ldots,$$

Then, perform RR if the following three conditions hold

$$d_j \leq \epsilon||r_j||, \qquad d_{j+1} > \epsilon||r_{j+1}||, \qquad d_{j+1} \geq 1.1\, d_{init}$$

- Compared with others, VY's RR technique:
  - Aims for small deviations between recurrence/true residuals
  - Preserves convergence mechanism of the iteration
  - It is cheap and easy to add to existing Krylov implementations

Residual Replacement for Krylov Solvers

H. A. Van der Vorst, Q. Ye. "Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals." SIAM J. Sci. Comput., 22(3), 2000

- Experiments proving effectiveness of VY's RR technique:
  - 28 matrices for Harwell-Boeing collection, BiCG and CGS methods
  - Only double precision arithmetic
  - #iterations not reported, only #RR
  - Convergence threshold set close to machine precision (likely impractical)

Residual Replacement for Krylov Solvers

H. A. Van der Vorst, Q. Ye. "Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals." SIAM J. Sci. Comput., 22(3), 2000

- Experiments proving effectiveness of VY's RR technique:
  - 28 matrices for Harwell-Boeing collection, BiCG and CGS methods
  - Only double precision arithmetic
  - #iterations not reported, only #RR
  - Convergence threshold set close to machine precision (likely impractical)
- Differences:
  - CG method
  - Integration in an outer-inner scheme for iterative refinement: MPIR
  - #iterations is important!
  - Convergence to more realistic residual bounds
  - Cost model

Cost Model

- Premises:
  - For a memory-bound algorithm, such as PCG applied to a sparse linear system, the “cost” is dominated by data movement while floating-point arithmetic is irrelevant
    - If cost = execution time, arithmetic cost is minor (memory wall) and can be overlapped with communication
    - If cost = energy, accesses to main memory are much more expensive than arithmetic

Operation	approximate energy cost
DP floating point multiply-add	100 pJ
DP DRAM read-to-register	4800 pJ

Cost Model

- Premises (cont'd):
  - After each particular operation, data does not remain in cache (large vectors)
  - Costs are linearly dependent on the bit-length of data
  - Problem of size n, with sparse matrix stored in CSR format consisting of n_z nonzero entries
  - Simple Jacobi preconditioner for CG

Cost Model

- SpMV, in CSR format, using data with xx bits in terms of bit transfers (or cost-units, cus):

```
y = A * x;
    CSR A:  row_ptr[n],
            col_idx[nz],
            val[nz]
```

```
for (i=0; i<n; i++) {
  tmp = 0;
  for (j=row_ptr[i]; j<row_ptr[i+1]; j++)
    tmp += val[j] * x[col_idx[j]];
  y[i] = tmp;
}
```

$$\mathcal{C}_{\mathrm{SPMV}}(\mathrm{xx}) = \underbrace{(n + 2n_z) \cdot \mathsf{fpxx}}_{\text{Vector, matrix entries}} + \underbrace{(n + n_z) \cdot \mathsf{int32}}_{\text{indices}} \quad \mathrm{cus}$$

Cost Model

- PCG solver operating with xx bits:

$r_0 := b - Ax_0,\ z_0 := M^{-1}r_0,\ d_0 := z_0,\ \beta_0 := r_0^T z_0,$	Initializations
$\tau_0 :=\| r_0 \|_2,\ j := 0$	
while $(\tau_j > \tau_{\max})$	**Iterative PCG solve**
$w_j := Ad_j$	SPMV
$\rho_j := \beta_j / d_j^T w_j$	DOT product
$x_{j+1} := x_j + \rho_j d_j$	AXPY
$r_{j+1} := r_j - \rho_j w_j$	AXPY
$z_{j+1} := M^{-1} r_{j+1}$	Preconditioning
$\beta_{j+1} := r_{j+1}^T z_{j+1}$	DOT product
$\alpha_j := \beta_{j+1} / \beta_j$	
$d_{j+1} := z_{j+1} + \alpha_j d_j$	AXPY-like
$\tau_{j+1} :=\| r_{j+1} \|_2$	2-norm
$j := j + 1$	
endwhile	

$$\mathcal{C}_{\text{PCG}}^{\text{iter}}(\text{xx}) = \underbrace{14n \cdot \text{fpxx}}_{\text{vector ops.}} + \mathcal{C}_{\text{SPMV}}(\text{xx}) + \underbrace{3n \cdot \text{fpxx}}_{\text{preconditioner appl.}} \quad \text{cus}$$

Cost Model

- For MPIR-VY, cost depends on:
  - #IS: number of iterations of inner solver
  - #RR: total number of residual replacements
  - #RS: number of refinement steps
- For example, using (32,64) mixed precision:

$$\mathcal{C}^{\mathrm{VY}}_{\mathrm{MPIR}}(32{,}64) = \underbrace{\mathcal{C}^{\mathrm{iter}}_{\mathrm{PCG}}(32) \cdot \#\mathrm{IS}}_{\text{Plain inner PCG solver}} + \underbrace{n \cdot \mathsf{fp32} \cdot \#\mathrm{IS}}_{\text{Replacement condition test}}$$
$$+ \underbrace{(4n \cdot \mathsf{fp32} + \mathcal{C}_{\mathrm{SPMV}}(32)) \cdot \#\mathrm{RR}}_{\text{RRs in inner PCG solver}}$$
$$+ \underbrace{(6n \cdot \mathsf{fp64} + n \cdot \mathsf{fp32} + \mathcal{C}_{\mathrm{SPMV}}(64)) \cdot \#\mathrm{RS}}_{\text{Refinement steps}} \quad \text{cus}$$

Explicit Residual Deviation with MPIR

- Our RR technique: **Explicit residual deviation** (ERD)
  - Test periodically (i.e., every t iterations)

$$\|r_{k+1}^{true}\|_2 / \|r_{k+1}^{rec}\|_2 \geq \tau$$

Computing the residual explicitly is expensive (SpMV), but it can be done **in reduced precision**

Cost can be further reduced by performing the residual calculation together with SpMV for inner solver

  - If deviation exceeds the threshold, stop the inner solver and start a new iteration of refinement (outer level) → enforces a residual replacement **in extended precision**

Explicit Residual Deviation with MPIR

- Cost of ERD-RR:

$$\mathcal{C}^{\text{ERD}}_{\text{MPIR}}(32,64) = \underbrace{\mathcal{C}^{\text{iter}}_{\text{PCG}}(32) \cdot \#\text{IS}}_{\text{Plain inner PCG solver}} + \underbrace{(4n \cdot \text{fp32} + \mathcal{C}_{\text{SPMV}}(32)) \cdot \#\text{IS}/t}_{\text{Residual tests in inner PCG solver}} + \underbrace{(6n \cdot \text{fp64} + n \cdot \text{fp32} + \mathcal{C}_{\text{SPMV}}(64)) \cdot \#\text{RS}}_{\text{Refinement steps}} \quad \text{cus}$$

Explicit Residual Deviation with MPIR

- VY vs EDR:
  - VY-RR incurs detection overhead at each iteration (test replacement condition) and pays correction overhead in case RR is necessary
  - EDR-RR incurs detection overhead only every t iterations (periodicity of the test), risking to waste work in case of stagnation from last test
  - Detection techniques are different and, therefore, also are numerical effects and overhead:

Apply RR in the inner solver, and continue with iteration

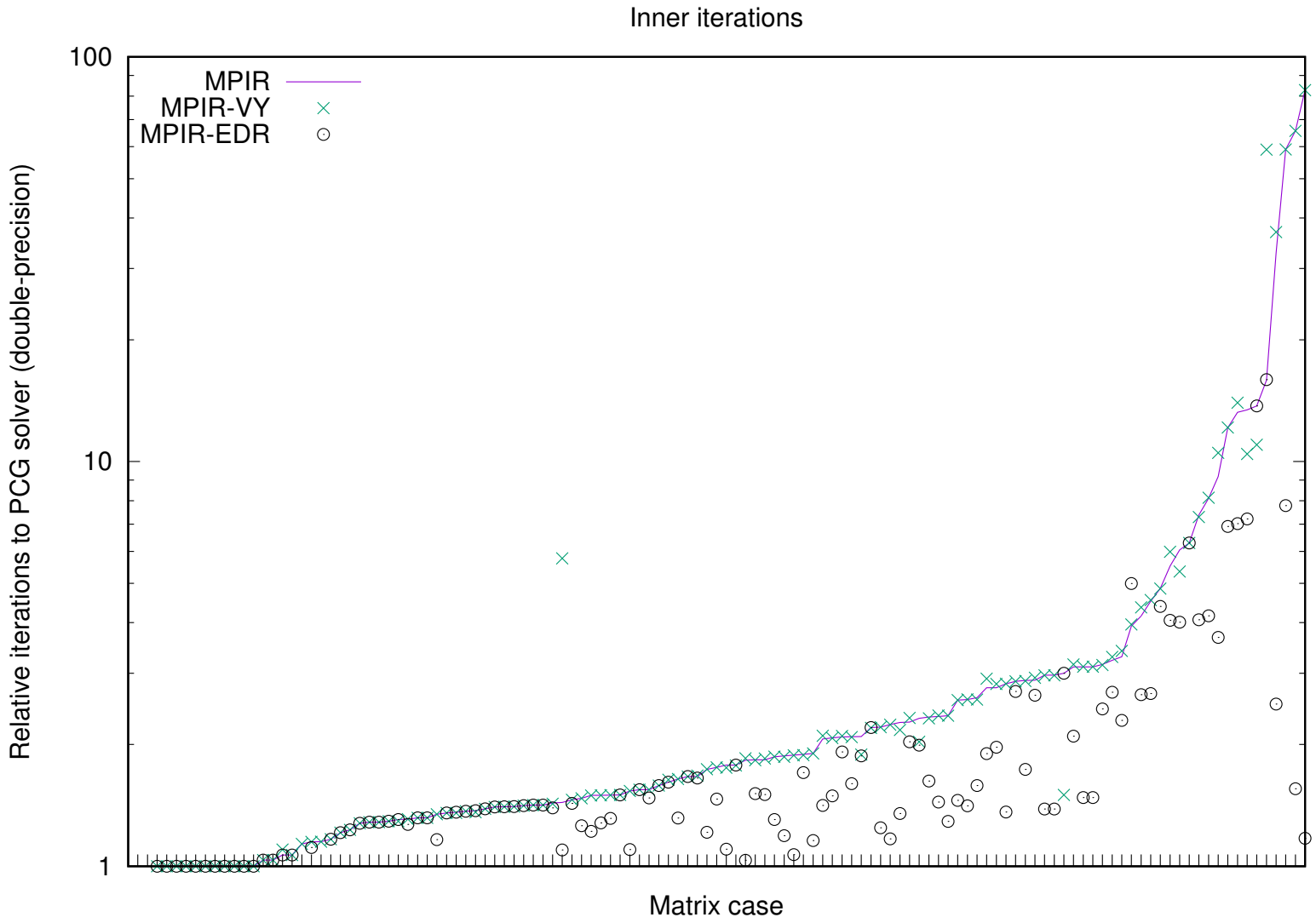
vs

Apply RR by ”moving” to the outer solver

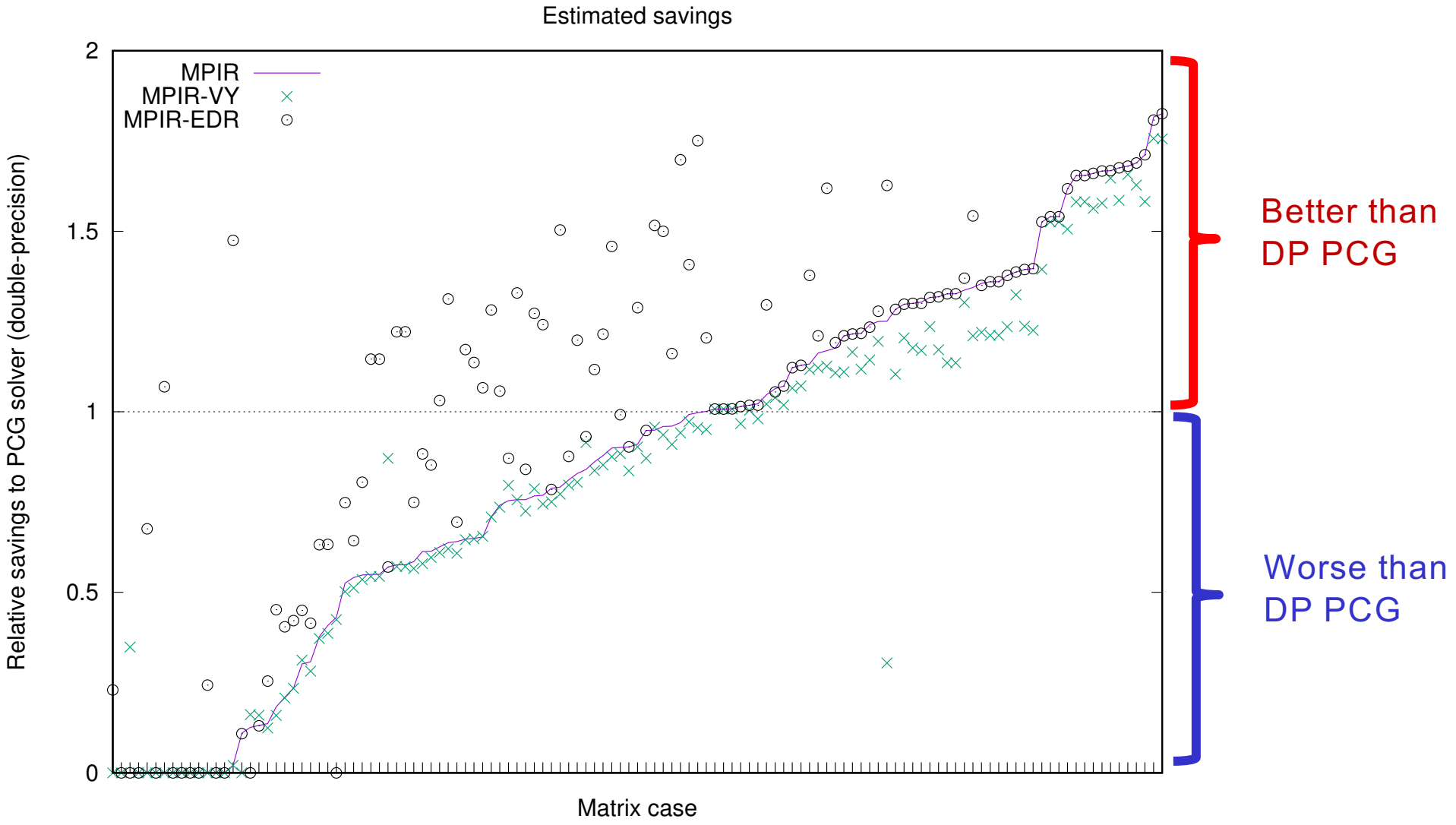
Cost Evaluation

- Setup:
  - 123 symmetric positive definite matrices from SuiteSparse Matrix Collection (formerly UFMC)
  - Baseline solver: PCG in double precision
  - All arithmetic done in double precision
  - For MPIR variants, all data used in the inner solver are stored in single precision: **reduced transfer cost!**
  - For EDR-RR, the test is performed every t = 100 iterations, and the maximum number of RR is set to 10
  - Cost take into account the actual number of iterations to obtain an absolute residual error below 10^{-7}

Cost Evaluation



Cost Evaluation



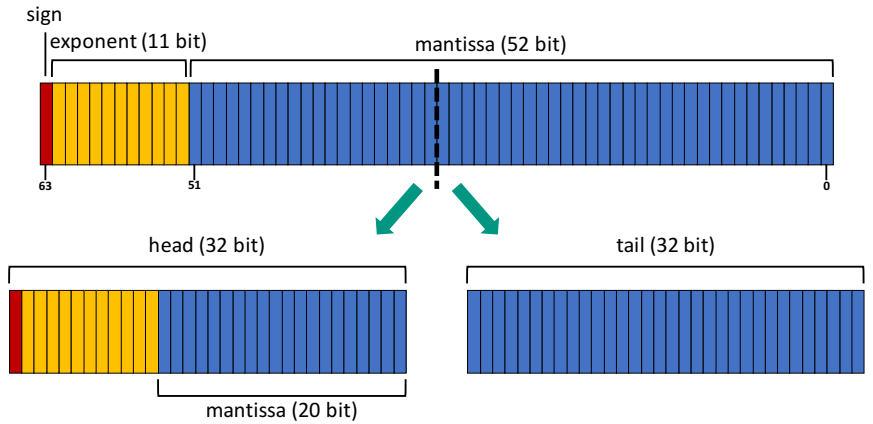
Customized Precision via Segmentation

- Decouple arithmetic from storage formats:
  - FPUs only support a limited number of IEEE 754 formats (single, double and, in some architectures, half)
  - … but we are free to store the data in memory in any **customized format**
- Remember: As a memory-bound algorithm, PCG is limited by memory bandwidth (i.e., how many bit are used to store the data)
  - Extended can be double
  - Reduced can be, e.g., 16, 24, 32, 40, 48, 56 bits
  - Maintain a single copy of the matrix with “multiple precisions” via **segments**

T. Grützmacher, H:. Anzt. “A modular precision format for decoupling arithmetic format and storage format”. Submitted to HeteroPar 2018

Customized Precision via Segmentation

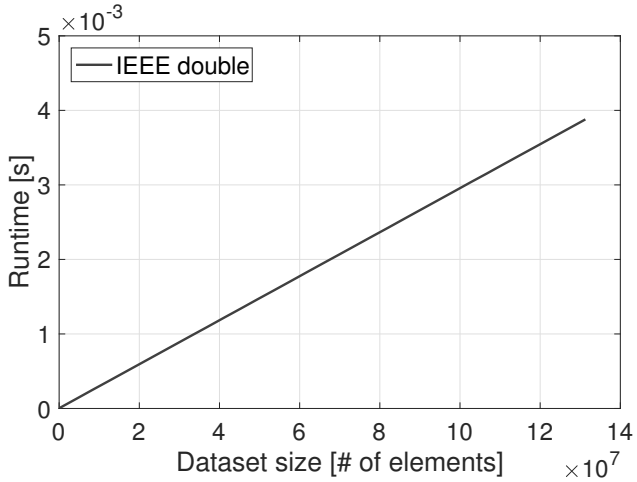
- Split the IEEE double precision format into segments.
  (2-segment modular precision, 4-segment modular precision…)

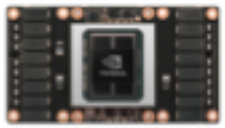


- *Special "conversion" routines to double precision.*
- *Mantissa much shorter than IEEE single/half precision.*
- *No under- / overflow.*

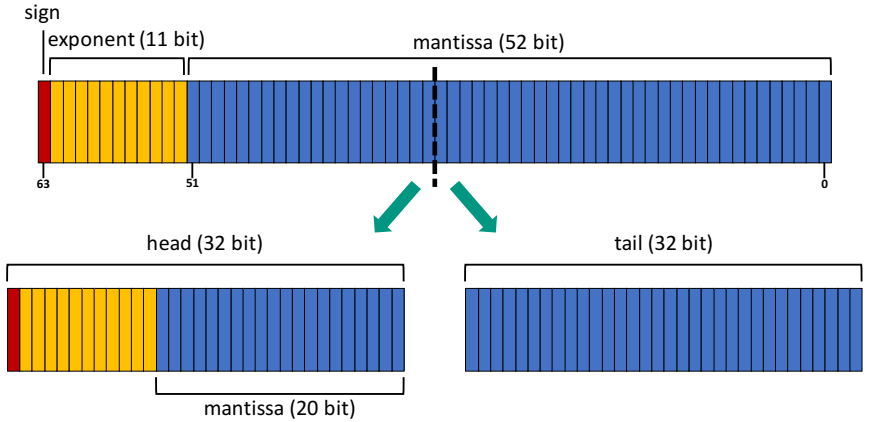
Customized Precision via Segmentation

- Split the IEEE double precision format into segments.
  (2-segment modular precision, 4-segment modular precision…)
- For efficient data access (coalesced reads), interleave data in memory.
- *Data can be accessed much faster if low precision is acceptable.*

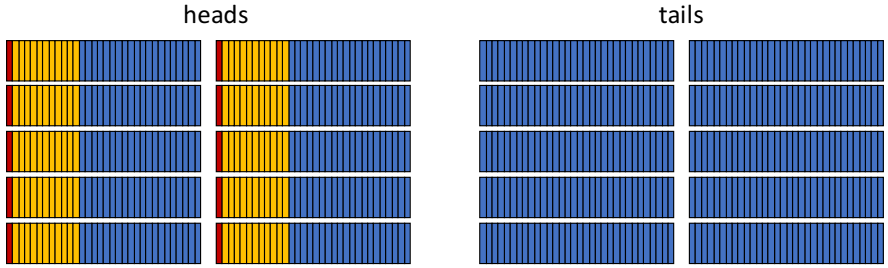




NVIDIA P100 "Pascal"
5.3 TFLOP/s DP
16GB RAM @720 GB/s

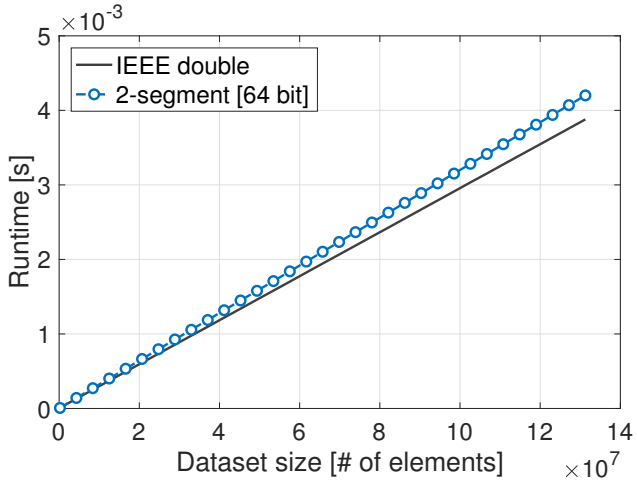


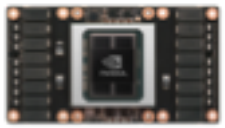
- *Special "conversion" routines to double precision.*
- *Mantissa much shorter than IEEE single/half precision.*
- *No under- / overflow.*



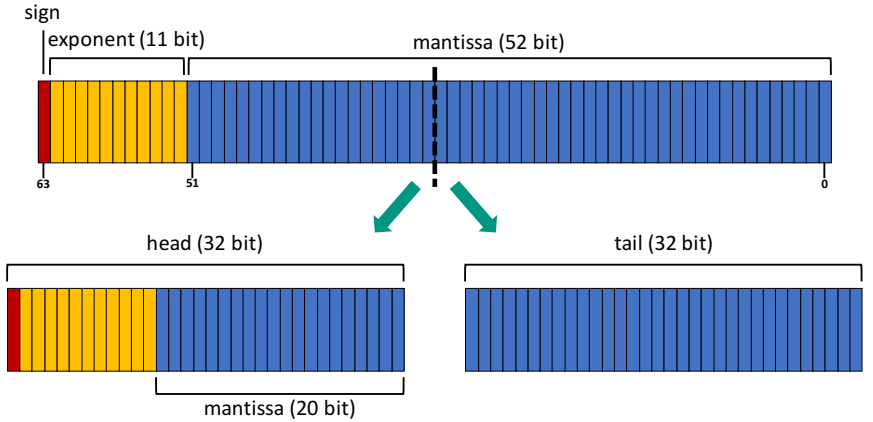
Customized Precision via Segmentation

- Split the IEEE double precision format into segments.
  (2-segment modular precision, 4-segment modular precision…)
- For efficient data access (coalesced reads), interleave data in memory.
- *Data can be accessed much faster if low precision is acceptable.*

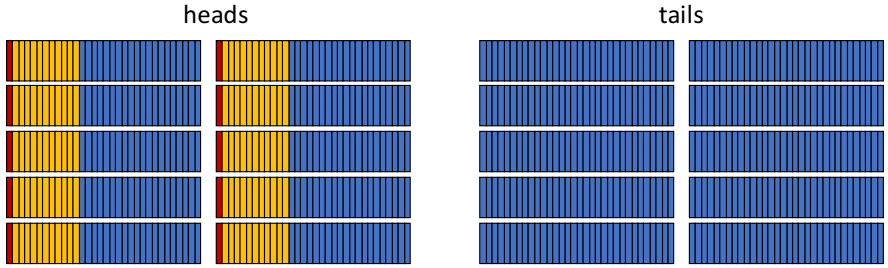




NVIDIA P100 "Pascal"
5.3 TFLOP/s DP
16GB RAM @720 GB/s

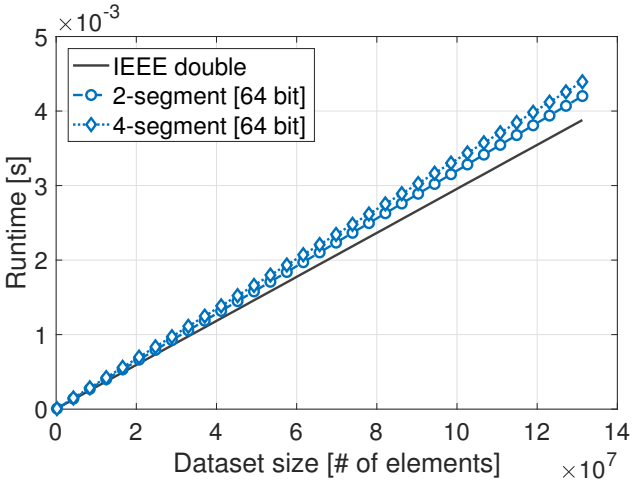


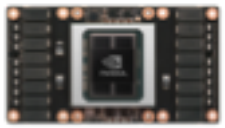
- *Special "conversion" routines to double precision.*
- *Mantissa much shorter than IEEE single/half precision.*
- *No under- / overflow.*



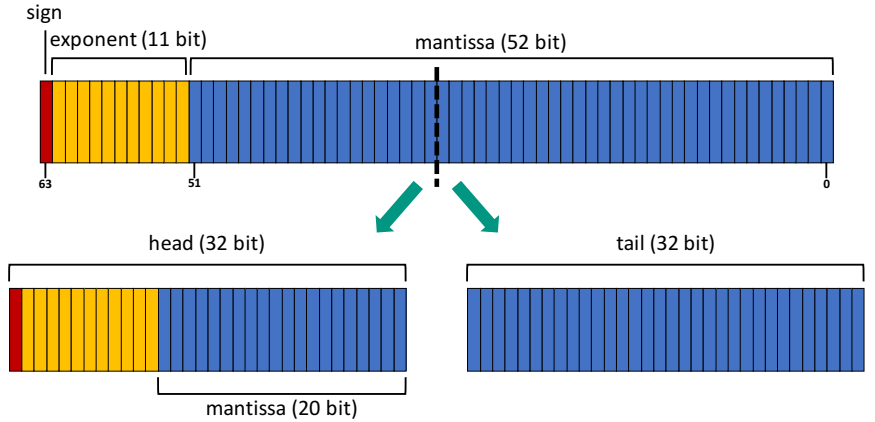
Customized Precision via Segmentation

- Split the IEEE double precision format into segments.
  (2-segment modular precision, 4-segment modular precision…)
- For efficient data access (coalesced reads), interleave data in memory.
- *Data can be accessed much faster if low precision is acceptable.*

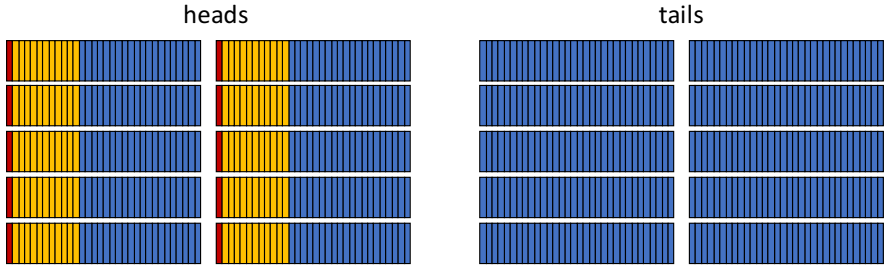




NVIDIA P100 "Pascal"
5.3 TFLOP/s DP
16GB RAM @720 GB/s

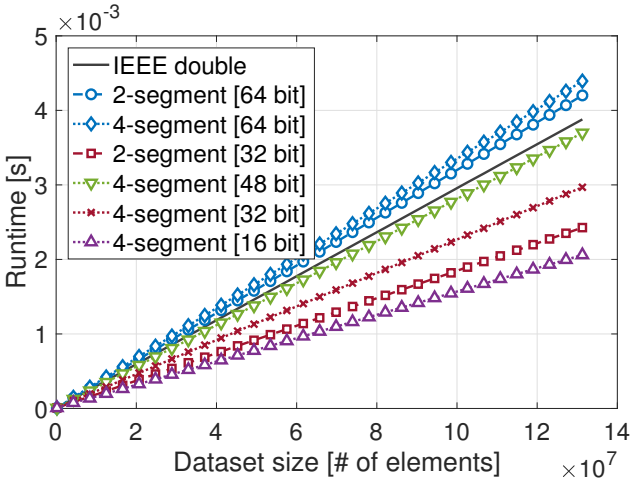


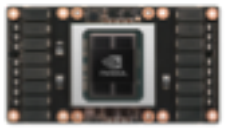
- *Special "conversion" routines to double precision.*
- *Mantissa much shorter than IEEE single/half precision.*
- *No under- / overflow.*



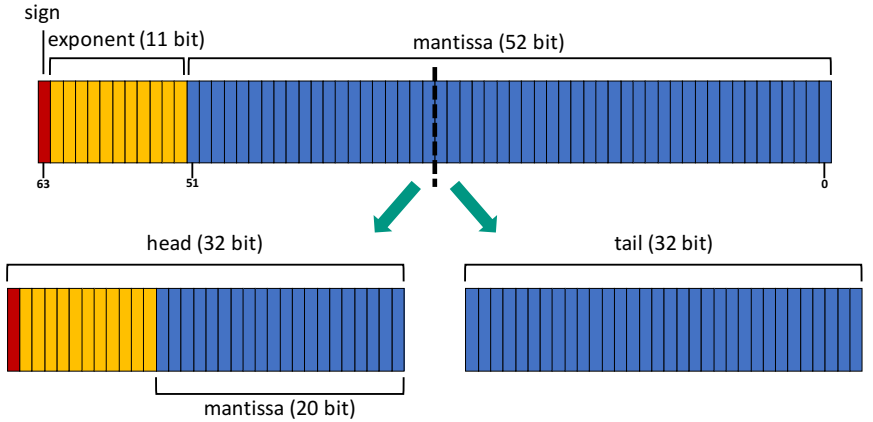
Customized Precision via Segmentation

- Split the IEEE double precision format into segments.
  (2-segment modular precision, 4-segment modular precision…)
- For efficient data access (coalesced reads), interleave data in memory.
- *Data can be accessed much faster if low precision is acceptable.*

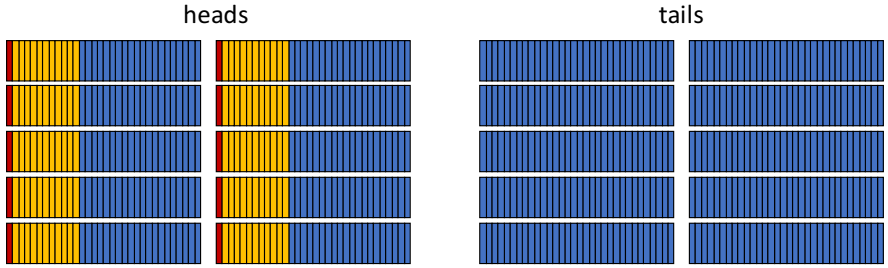




NVIDIA P100 "Pascal"
5.3 TFLOP/s DP
16GB RAM @720 GB/s



- *Special "conversion" routines to double precision.*
- *Mantissa much shorter than IEEE single/half precision.*
- *No under- / overflow.*



Concluding Remarks

- MPIR can be efficiently combined with any inner Krylov solver provided it is enhanced with an appropriate RR technique
- Theoretical savings can be significant in terms of reduced memory bandwidth: lower energy and computational costs
- Reduced precision storage in the inner solver can be realized via modular precision formats