SAND2022-7865C

Mixed Precision *s*-step Conjugate Gradient with Residual Replacements on (NVIDIA) GPUs

Ichitaro Yamazaki*, Erin Carson ᵀ, Brian Kelley*

*Sandia National Laboratories, New Mexico, USA
ᵀCharles University, Prague, Czech Republic

36th IEEE International Parallel and Distributed Processing Symposium (IPDPS)
June 2, 2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Aims with "Mixed Precision *s*-step Conjugate Gradient with Residual Replacements on (NVIDIA) GPUs"

- improve the stability of the solver, using higher-precision arithmetic with two main aims:

1. obtain higher accuracy (converging to lower residual norm)
2. improve performance (converging with a fewer iterations, but without significant increase in the per-iteration time = faster time-to-solution)
   - Careful design and implementation
     - using higher precision only at the critical parts of the algorithms
     - optimizing the underlying kernels for particular properties

Conjugate Gradient (CG)

Require: SPD matrix A, RHS vector $\mathbf{b}$, and initial approximate solution vector $\mathbf{x}_1$

$\mathbf{r}_1 := \mathbf{b} - A\mathbf{x}_1,\ \delta_1 := \mathbf{r}_1^T\mathbf{r}_1,\ \mathbf{p}_1 := \mathbf{r}_1$
for $j = 1, 2, \ldots$ **do**
    // SpMV with P2P communication
    $\boxed{\mathbf{w}_j := A\mathbf{p}_j}$
    // dot-product with global-reduce
    $\boxed{\gamma_j := \mathbf{p}_j^T\mathbf{w}_j}$
    $\alpha_j := \delta_j / \gamma_j$
    // update solution and residual vectors
    $\boxed{\mathbf{x}_{j+1} := \mathbf{x}_j + \alpha_j\mathbf{p}_j}$
    $\boxed{\mathbf{r}_{j+1} := \mathbf{r}_j - \alpha_j\mathbf{w}_j}$
    // dot-product with global-reduce
    $\boxed{\delta_{j+1} := \mathbf{r}_{j+1}^T\mathbf{r}_{j+1}}$
    if Converged **then**
        break
    else
        // compute next search direction
        $\beta_{j+1} := \delta_{j+1}/\delta_j$
        $\boxed{\mathbf{p}_{j+1} := \mathbf{r}_{j+1} + \beta_j\mathbf{p}_j}$
    end if
end for

- CG is a popular iterative method for symmetric positive definite (SPD) linear system, $Ax = b$.
- It relies on two types of kernels
  - Matrix Vector multiply (SpMV)
    - for generating Krylov subspace = span(p, Ap, A^2p, …)
    - typically combined with precondiner to improve convergence
    - used as a black box, provided by users, for supporting a wide range of applications
  - BLAS-1 operations (focus of the paper)
    - for computing search direction, to update solution and residual vectors
    - two **`dot`**'s (with global all-reduce) and three **`axpy`**'s
- CG iteration relies on efficient short-term recurrence, but underlying BLAS-1 kernels are latency bound with low performance
  - could become significant in the iteration time (e.g., at large scale)

s-step Conjugate Gradient [Chronopoulos, Gear ’89]

- s-step CG generates a set of s basis vectors at a time
  - Potential reduction in communication cost by a factor of s
    - reducing latency cost (one synchronization per s steps)
    - exposing more parallelism and data reuse (BLAS-3 instead of BLAS-1)
    - Require $O(1)$ communication for generating $O(s)$ basis vectors.

for $j = 1,\ 1 \cdot s + 1,\ 2 \cdot s + 1, \dots$ **do**
    // Matrix Powers Kernel to generate Krylov space
    for $k = 1, 2, \dots, s$ **do**
        $[\mathbf{r}_{j+k}, \mathbf{p}_{j+k+1}] := A\,[\mathbf{r}_{j+k-1}, \mathbf{p}_{j+k}]$
    end for
    // Compute new solution and residual vector
    $\boxed{G := V^T V}$ with $V = [P_{j:j+s}, R_{j:j+s-1}]$
    $\dots$
    $\boxed{[\mathbf{r}_{j+s}, \mathbf{x}_{j+1}, \mathbf{p}_{j+2}] := V[\mathbf{y}, \mathbf{t}, \mathbf{c}]}$
end for

- Two challenges for practical use
  1. Computational overheads
  2. Numerical stability

Computational overhead (first challenge)

- In order to reduce communication, it requires additional computation
  - If underlying kernels are optimized for multiple vectors, performance may be improved

Require: SPD matrix A, RHS vector $\mathbf{b}$, and initial approximate solution vector $\mathbf{x}_1$

$\mathbf{r}_1 := \mathbf{b} - A\mathbf{x}_1$, $\delta_1 := \mathbf{r}_1^T\mathbf{r}_1$, $\mathbf{p}_1 := \mathbf{r}_1$
for $j = 1, 2, \ldots$ **do**
  // SpMV with P2P communication
  $\mathbf{w}_j := A\mathbf{p}_j$
  // dot-product with global-reduce
  $\gamma_j := \mathbf{p}_j^T\mathbf{w}_j$
  $\alpha_j := \delta_j/\gamma_j$
  // update solution and residual vectors
  $\mathbf{x}_{j+1} := \mathbf{x}_j + \alpha_j\mathbf{p}_j$
  $\mathbf{r}_{j+1} := \mathbf{r}_j - \alpha_j\mathbf{w}_j$
  // dot-product with global-reduce
  $\delta_{j+1} := \mathbf{r}_{j+1}^T\mathbf{r}_{j+1}$
  if Converged **then**
    break
  else
    // compute next search direction
    $\beta_{j+1} := \delta_{j+1}/\delta_j$
    $\mathbf{p}_{j+1} := \mathbf{r}_{j+1} + \beta_j\mathbf{p}_j$
  end if
end for

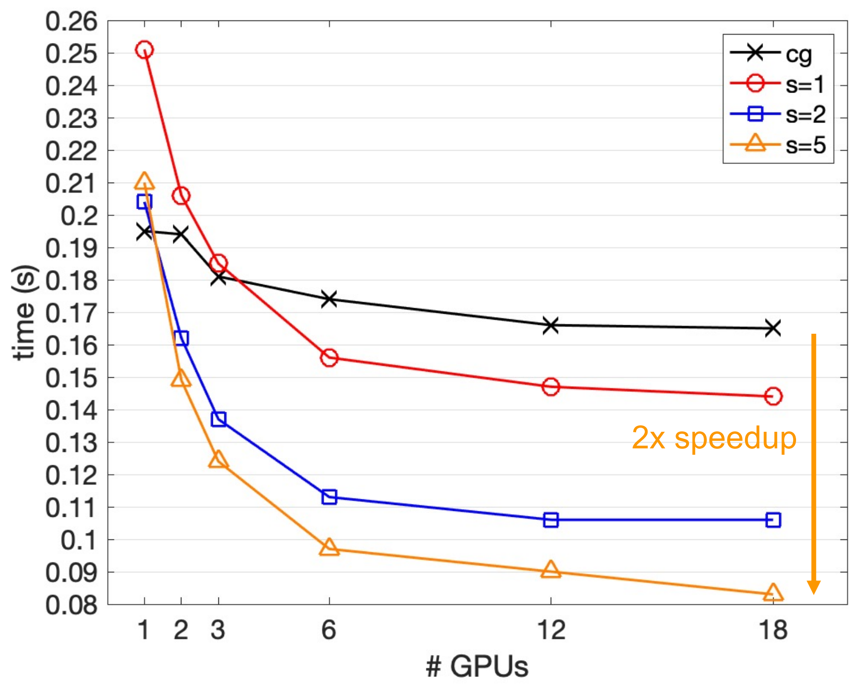
for $j = 1,\ 1\cdot s+1,\ 2\cdot s+1, \ldots$ **do**
  // Matrix Powers Kernel to generate Krylov space
  for $k = 1, 2, \ldots, s$ **do**
    $[\mathbf{r}_{j+k}, \mathbf{p}_{j+k+1}] := A\,[\mathbf{r}_{j+k-1}, \mathbf{p}_{j+k}]$
  end for
  // Compute new solution and residual vector
  $G := V^TV$ with $V = [P_{j:j+s}, R_{j:j+s-1}]$
  $\ldots$
  $[\mathbf{r}_{j+s}, \mathbf{x}_{j+1}, \mathbf{p}_{j+2}] := V[\mathbf{y}, \mathbf{t}, \mathbf{c}]$
end for

"s" **SpMM**s with two vectors = **2× flops**, but two vectors at a time

One dot-products with "2s+1" vectors = **2s× flops**, but with one **SYRK**

3 **GEMV**s with "2s+1" columns = **2× flops**, but with one **GEMM**

Strong-scaling results on Summit (V100 GPUs)

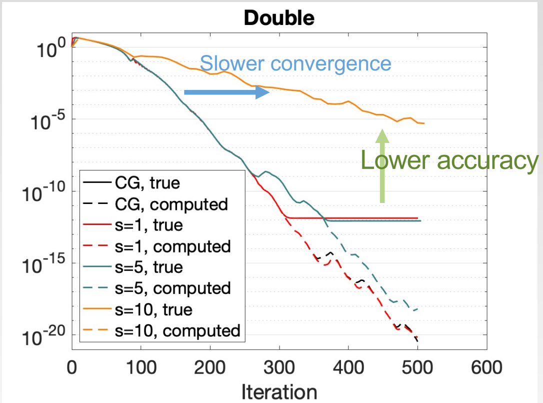


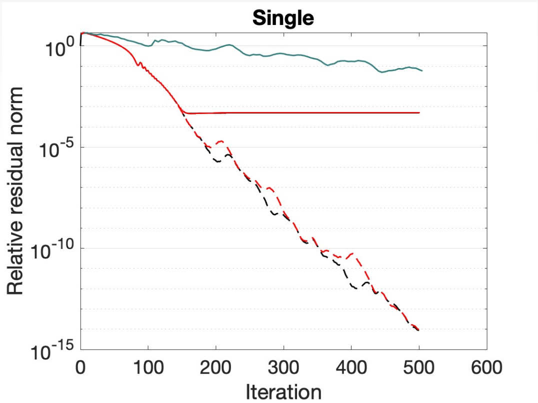
- 500 CG iteration time with 7-pts Laplace 3D (n_x=100)
- When communication (e.g., **latency**) becomes significant, *s*-step CG may reduce iteration time, even with the computational overhead

EXASCALE COMPUTING PROJECT

Potential numerical instability with *s*-step CG (second challenge)

- Both convergence rate and attainable accuracy can deteriorate
- Potentially very ill-conditioned *s*-step basis vectors V_{2s+1}
  - Condition number $\kappa(V_{2s+1})$ can grow exponentially with s
  - Orthogonality errors can grow quadratically to the condition number
    - The Gram matrix G has the squared condition number





Mixed-precision s-step CG with **residual replacement** on GPUs

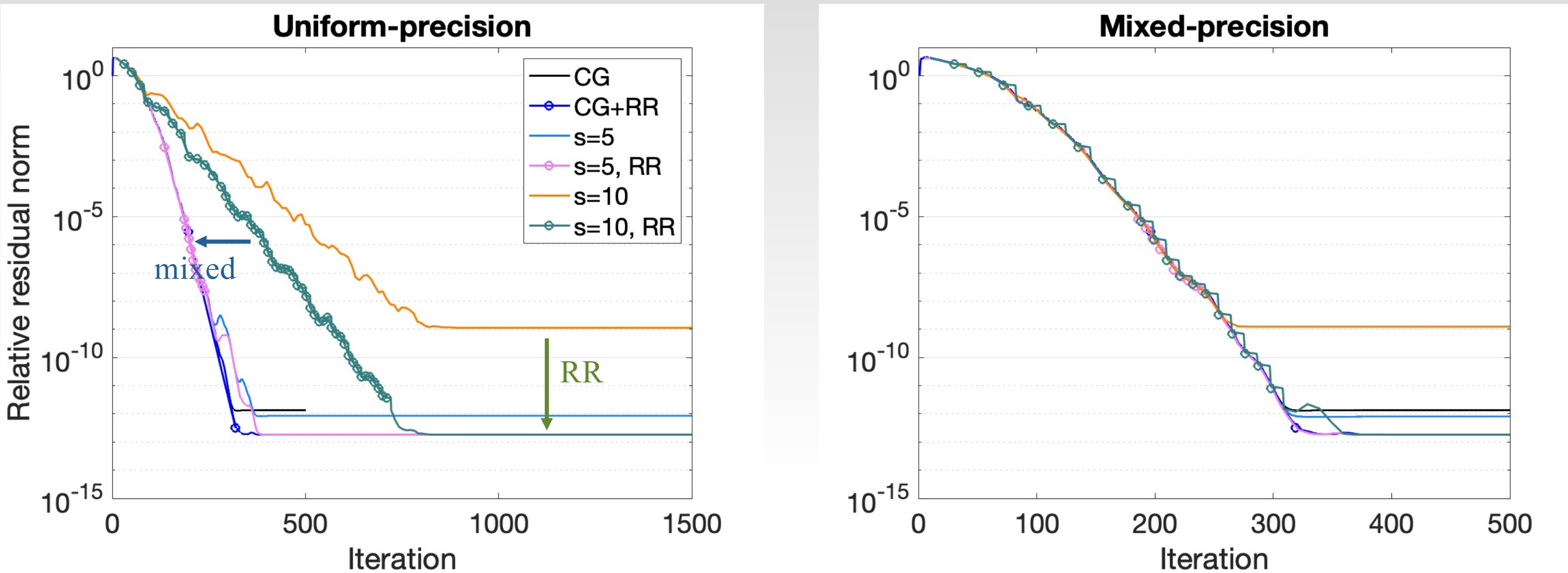
for improving convergence and accuracy

- Higher-precision to improve convergence behavior [Carson, Gergelits, '21]
  - form the Gram matrix G in **double** the working precision
  - orthogonality error depends linearly, instead of quadratically, to condition number of s-step basis vectors
- Residual replacement to improve solution accuracy [Carson, Demmel '14]
  - replace computed residual vector with true residual vector at "selected" iterations
  - the selection requires the computation of $\bar{G} = |V_{(j-1)/s}|^T |V_{(j-1)/s}|$, in the working precision
  - s-step CG obtains the same residual norm bound as standard CG, $O(\epsilon)\|A\|\|x\|$.

```
r_1 := b − A x_1, δ_1 := r_1^T r_1, p_1 := r_1
z = 0
for j += ŝ do
    // MPK (on GPUs) with P2P communication
    P_j := p_j, R_j := r_j
    P_{j+1} := A P_j
    for k = 1, 2, ..., s − 1 do
        [R_{j+k}, P_{j+k+1}] := A[R_{j+k−1}, P_{j+k}]
    end for
    // dot-products (on GPUs) with global-reduce
    G := V_ℓ^T V_ℓ,
      where V_ℓ := [P_{j:j+s}, R_{j:j+s−1}] and ℓ := (j−1)/s
    if Converged then
        break
    end if
    // update coefficients (redundantly on each host)
    c_1 := e_1, t_1 := e_{s+1}, δ_k := t_k G t_k
    for k = 1, 2, ..., s do
        d_k := B c_k, γ_k := c_k G d_k, α_k := δ_k / γ_k
        y_{k+1} := y_k + α_k c_k
        t_{k+1} := t_k − α_k d_k
        δ_{k+1} := t_{k+1}^T G t_{k+1}, β_k := δ_{k+1} / δ_k
        c_{k+1} := t_{k+1} + β_k c_k
        if time to replace residual vector then
            x_{j+k} := x_j + [P_{j:j+s} R_{j:j+s−1}] y_{k+1}
            p_{j+k} := [P_{j:j+s} R_{j:j+s−1}] c_{k+1}
            z = z + x_{j+k}
            r_{j+k} := b − A z_{j+k}
            x_{j+k} := 0
            ŝ := k
            break
        end if
    end for
    if not replaced then
        // update vectors (on GPUs)
        x_{j+s} := x_j + [P_{j:j+s} R_{j:j+s−1}] y_{s+1}
        r_{j+s} := [P_{j:j+s} R_{j:j+s−1}] t_{s+1}
        p_{j+s} := [P_{j:j+s} R_{j:j+s−1}] c_{s+1}
        ŝ := s
    end if
end for
x := x_end + z
```

EXASCALE COMPUTING PROJECT

Numerical results with mixed-precision *s*-step CG with RR
using 3D Laplace ($n=100^3$)



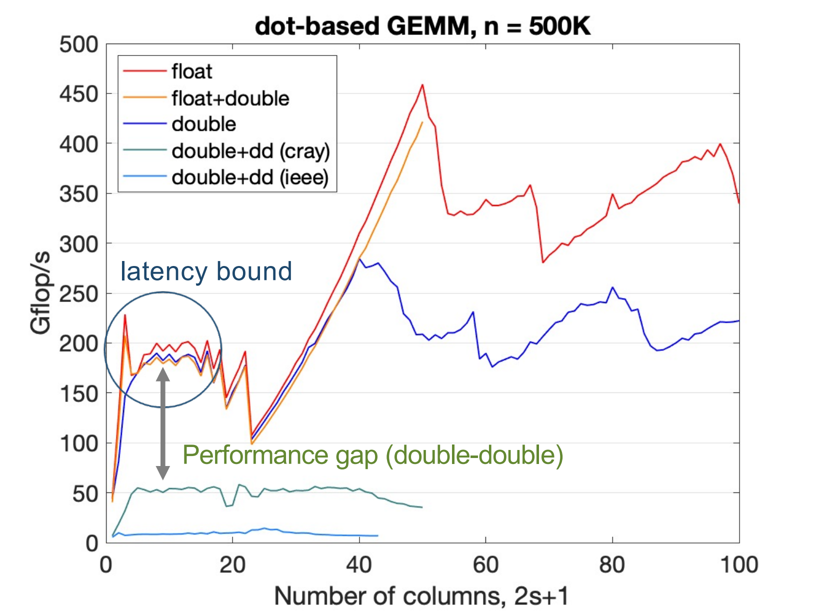
Both mixed-precision and RR are needed to obtain convergence similar to standard CG

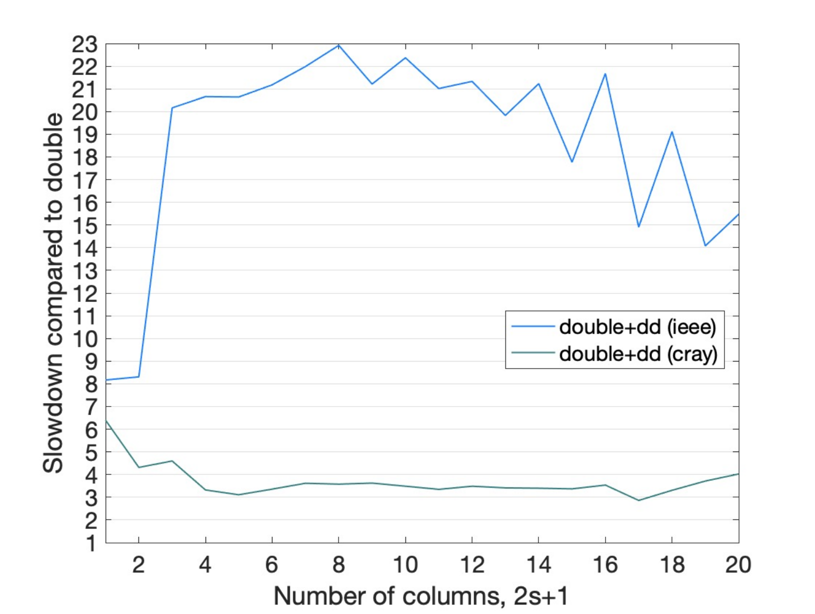
- Higher-precision improves the convergence
- RR improves the accuracy

Implementation for performance study on a GPU cluster (Summit)

- MPI (cuda-aware) for data exchange between GPUs
- Kokkos for portable performance on different manycore architectures
  - we only show performance on NVIDIA GPUs,
- Mixed-precision dot-products to compute G
  - It reads “big” tall-skinny V in working precision, but internally use higher precision to compute “small” G
  - It is latency bound, hopefully with a small overhead (of computing and writing G in higher precision)
- **double** or **single** precision as our working precision
  - double working precision
    - typical for scientific and engineering application
    - may require software-emulated higher precision (double-double in our experiments on V100 GPUs)
  - single working precision
    - experiments where higher precision is implemented by hardware,
    - practical use of single-precision CG exists, e.g., mixed-precision reliable updates and iterative refinements

Performance of mixed-precision dot-products using Kokkos on one NVIDIA V100 GPU





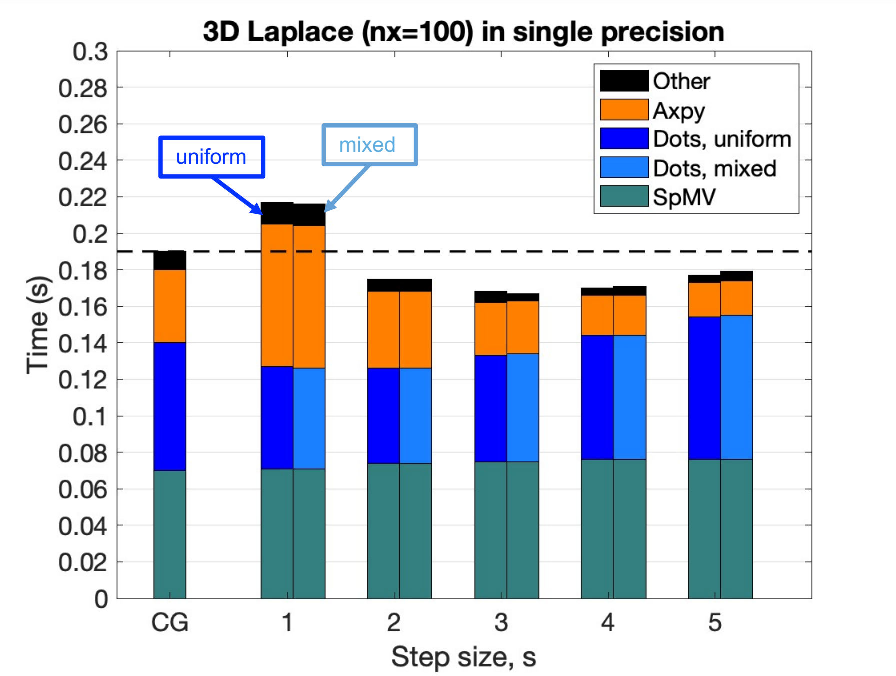
- Dot-products is often latency bound, and virtually no overhead using higher precision, when implemented by hardware
  - For mixed single+double or uniform double, vs uniform single
- Mixed-precision dot-products with Cray-style double-double requires 17× more flops, while IEEE variant requires 21× more flops
- The overhead tends to become smaller on multiple GPUs.

Iteration time breakdown for single working precision on one NVIDIA V100 GPU

hardware-implemented higher precision

- Uniform precision
  - SpMM with two vectors is as fast as SpMV with one vector
  - Vector-updates with multiple vectors continue to improve the performance with a larger *s*
  - Dot-products improves the performance for a small *s*, but the computational overhead (2*s*×) becomes significant for a large *s*

- Mixed-precision, float + double, dot-product
  - Input vectors are read in single precision
  - No overhead in performing multiply-add and write back *G*, in double precision
  - Any reduction in the iteration count has direct impact to time-to-solution, no overhead even if not reduction in the iteration count

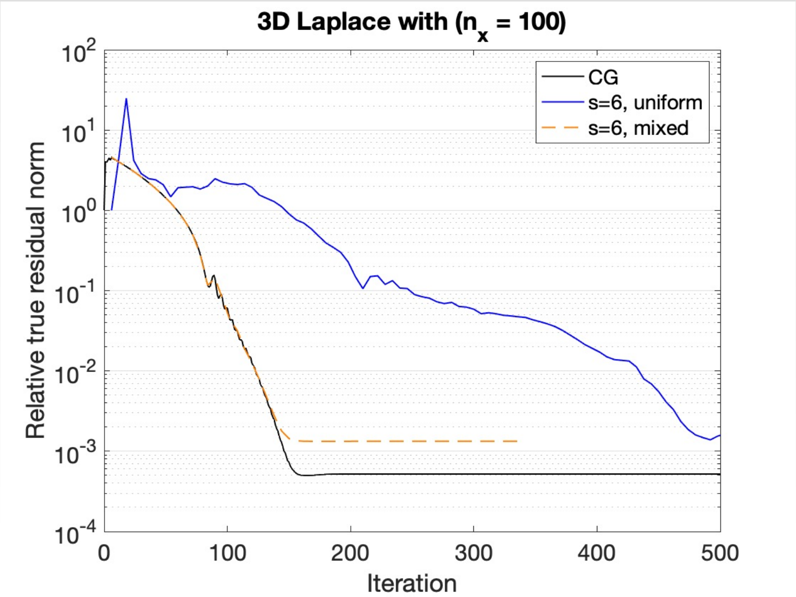


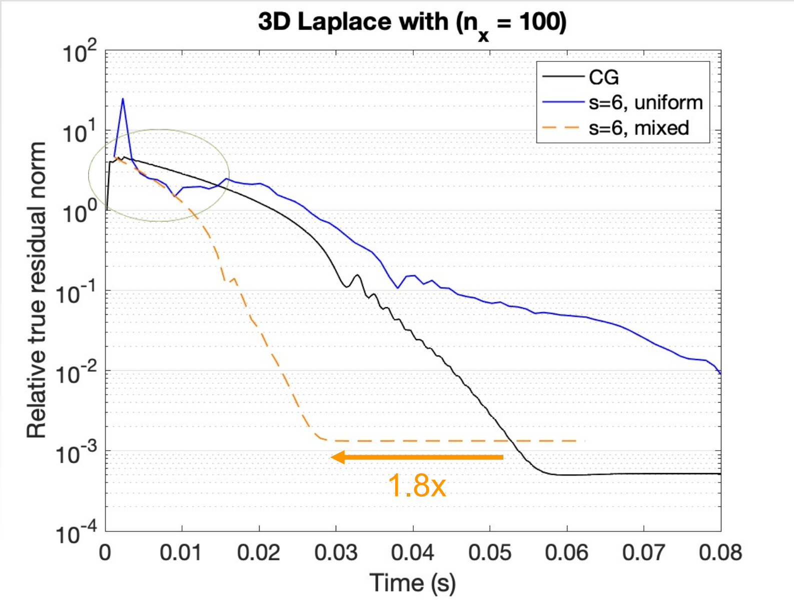
Kernel performance study on one GPU (up to 1.1x), while larger speedups on multiple GPUs

EXASCALE COMPUTING PROJECT

Time-to-solution with mixed-precision single+double *s*-step CG on six NVIDIA V100 GPUs

hardware-implemented higher precision



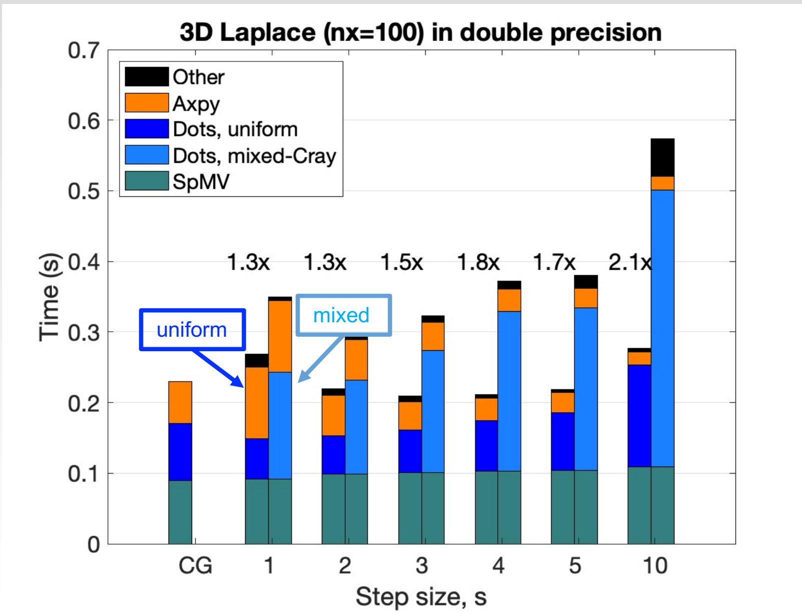


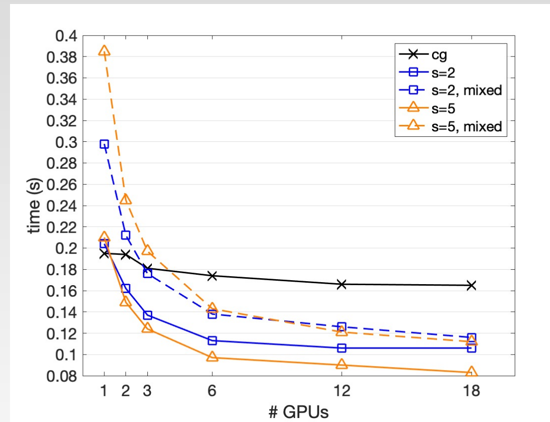
- *s*-step reduces the time-per-iteration, but can suffer from numerical instability
- mixed-precision improves the stability with virtually no overhead, leading to faster to solution

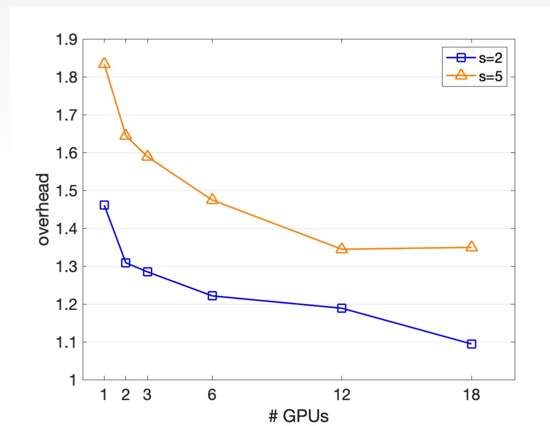
EXASCALE COMPUTING PROJECT

Iteration time breakdown with double-precision on an NVIDIA V100 GPU

software-emulated higher precision



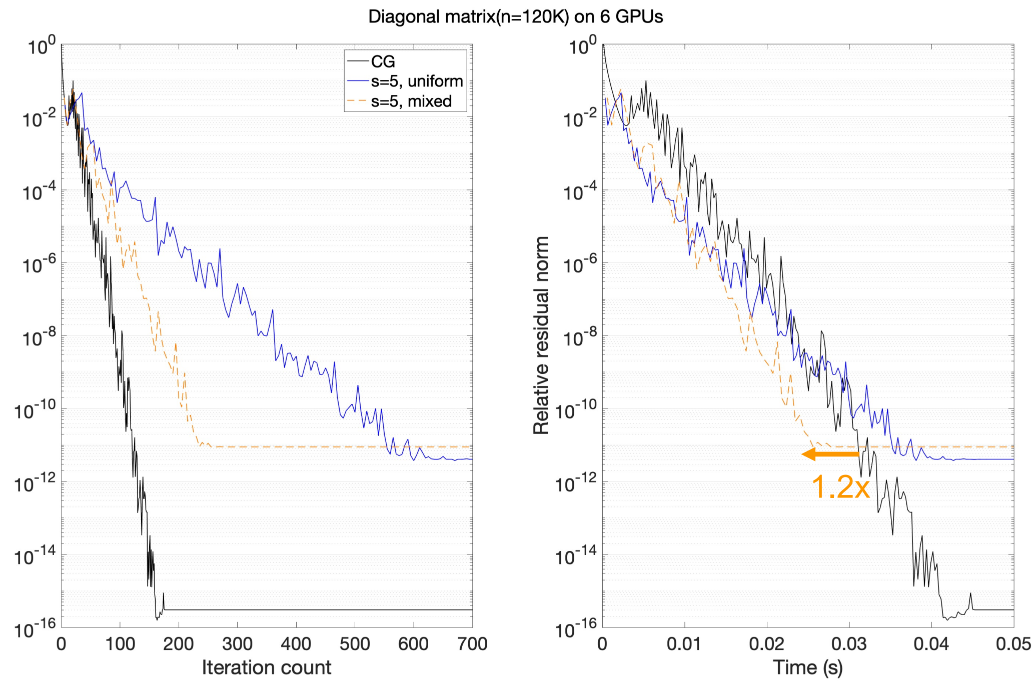




- mixed-precision dot-products has significant overhead especially with a large *s*
  - Overhead becomes smaller as latency become more significant on multiple GPUs

Time-to-solution with mixed double + double-double *s*-step CG

software-emulated higher precision



- Using diagonal matrix,

$$a_{i,i} = \lambda_1 + (i-1)/(n-1)(\lambda_n - \lambda_1)\rho^{n-i}.$$

  - Overhead of dot-products is more significant
  - Allows controlling the conditioning of the matrix
- Even with software-emulated higher precision, there are cases where mixed-precision variant reduces the time-to-solution
  - when *s*-step CG suffers from instability with a small step size, *s*

EXASCALE COMPUTING PROJECT

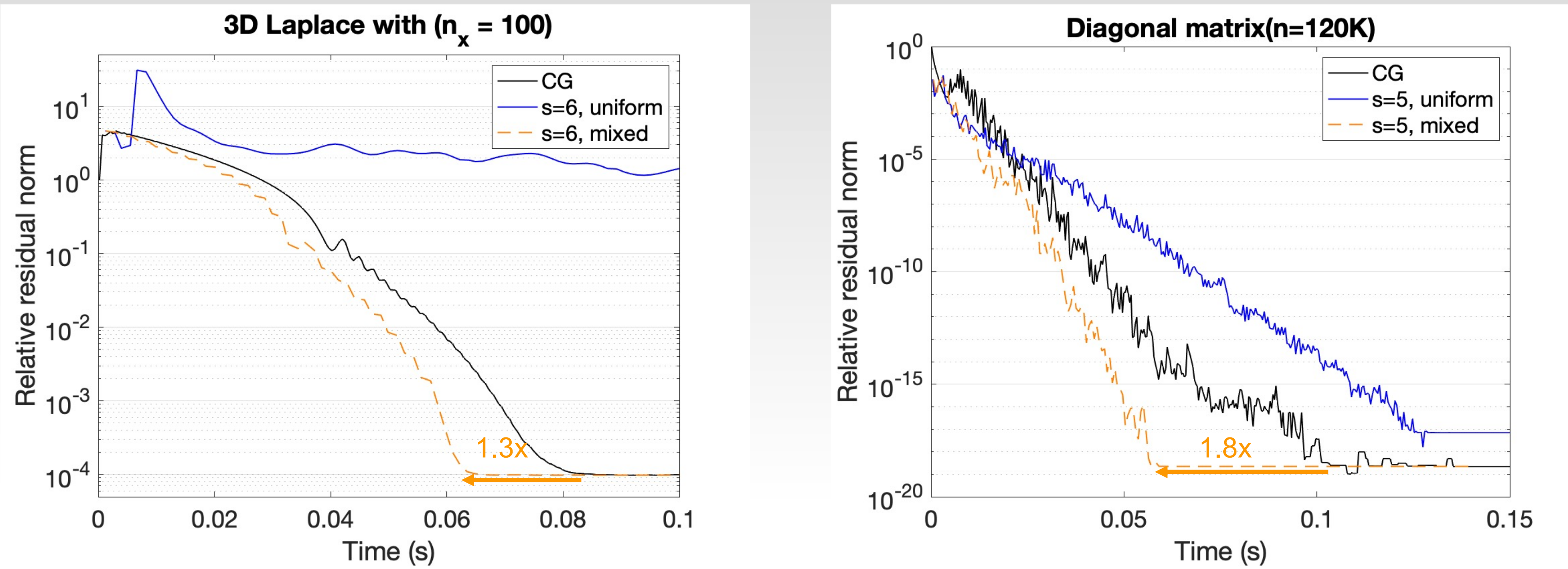
Mixed-precision *s*-step CG with **residual replacement**

Algorithm:

$\mathbf{r}_1 := \mathbf{b} - A\mathbf{x}_1$, $\delta_1 := \mathbf{r}_1^T\mathbf{r}_1$, $\mathbf{p}_1 := \mathbf{r}_1$
$\mathbf{z} = 0$
for $j += \widehat{s}$ **do**
  // MPK (on GPUs) with P2P communication
  $P_j := \mathbf{p}_j$, $R_j := \mathbf{r}_j$
  $P_{j+1} := AP_j$
  for $k = 1, 2, \ldots, s-1$ **do**
    $[R_{j+k}, P_{j+k+1}] := A[R_{j+k-1}, P_{j+k}]$
  end for
  // dot-products (on GPUs) with global-reduce
  $G := V_\ell^T V_\ell$,
   where $V_\ell := [P_{j:j+s}, R_{j:j+s-1}]$ and $\ell := (j-1)/s$
  if Converged **then**
    break
  end if
  // update coefficients (redundantly on each host)
  $\mathbf{c}_1 := \mathbf{e}_1$, $\mathbf{t}_1 := \mathbf{e}_{s+1}$, $\delta_k := \mathbf{t}_k G \mathbf{t}_k$
  for $k = 1, 2, \ldots, s$ **do**
    $\mathbf{d}_k := B\mathbf{c}_k$, $\gamma_k := \mathbf{c}_k G \mathbf{d}_k$, $\alpha_k := \delta_k/\gamma_k$
    $\mathbf{y}_{k+1} := \mathbf{y}_k + \alpha_k \mathbf{c}_k$
    $\mathbf{t}_{k+1} := \mathbf{t}_k - \alpha_k \mathbf{d}_k$
    $\delta_{k+1} := \mathbf{t}_{k+1}^T G \mathbf{t}_{k+1}$, $\beta_k := \delta_{k+1}/\delta_k$
    $\mathbf{c}_{k+1} := \mathbf{t}_{k+1} + \beta_k \mathbf{c}_k$
    if time to replace residual vector **then**
      $\mathbf{x}_{j+k} := \mathbf{x}_j + [P_{j:j+s} R_{j:j+s-1}]\mathbf{y}_{k+1}$
      $\mathbf{p}_{j+k} := [P_{j:j+s} R_{j:j+s-1}]\mathbf{c}_{k+1}$
      $\mathbf{z} = \mathbf{z} + \mathbf{x}_{j+k}$
      $\mathbf{r}_{j+k} := \mathbf{b} - A\mathbf{z}_{j+k}$
      $\mathbf{x}_{j+k} := 0$
      $\widehat{s} := k$
      break
    end if
  end for
  if not replaced **then**
    // update vectors (on GPUs)
    $\mathbf{x}_{j+s} := \mathbf{x}_j + [P_{j:j+s} R_{j:j+s-1}]\mathbf{y}_{s+1}$
    $\mathbf{r}_{j+s} := [P_{j:j+s} R_{j:j+s-1}]\mathbf{t}_{s+1}$
    $\mathbf{p}_{j+s} := [P_{j:j+s} R_{j:j+s-1}]\mathbf{c}_{s+1}$
    $\widehat{s} := s$
  end if
end for
$\mathbf{x} := \mathbf{x}_{\text{end}} + \mathbf{z}$

- residual replacement to improve the attainable accuracy
  - With residual replacement, *s*-step CG obtains the same residual norm bound as standard CG, $O(\epsilon)\|A\|\|x\|$. [Carson, Demmel '14]
  - The detection require the computation of $\bar{G} = |V_{(j-1)/s}|^T |V_{(j-1)/s}|$, in the working precision
  - If the residual needs to be replaced before *s*-th step, we waste some computation needed to form G, and also take a step smaller than *s*
- dot-products is latency or bandwidth limited
  improve convergence (a fewer iterations) without significant increase in iteration time

Time-to-solution with mixed *s*-step CG with RR



- RR adds overhead, but improves the attainable solution accuracy
- More results in paper

EXASCALE COMPUTING PROJECT

Final remarks

- We studied mixed-precision *s*-step CG with residual replacements on GPUs
  - When the higher-precision is supported by hardware, it improves the stability with virtually no overhead, and hence reduces time-to-solution, or if not, no overhead.
  - If the higher-precision requires software-emulation, the overhead becomes significant. It may still help when *s*-step CG becomes unstable with a small step size, and the latency becomes significant in the iteration time.

- We are planning on some extensions/variations of the algorithm
  - We have only looked at monomial basis. Combining these techniques with more stable basis (e.g., Newton, Chebychev) may further improve the stability, and practicability, of *s*-step CG
  - We only looked at two-term recurrence variant of s-step CG. There is also three-term recurrence variant, where relative cost of dot-products is smaller in iteration cost, and hence the mixed-precision may be more attractive.

Thank you!!

Acknowledgements

This research was, in part, supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

*We thank the members of **xSDK multiprecision project**, which is part of the Exascale Computing Project.*