Preconditioning Communication-Avoiding Krylov Methods

Sivasankaran Rajamanickam[†], Ichitaro Yamazaki[*], Erik G. Boman[†], Mark Hoemmen[†], Michael A. Heroux[†], Stanimire Tomov[*], Jack Dongarra[*]

[†]Sandia National Laboratories, Albuquerque, New Mexico, USA
[*]University of Tennessee, Knoxville, USA

SIAM Computational Science and Engineering
March 2015

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

We consider solving the linear system of equations,

$$Ax = b,$$

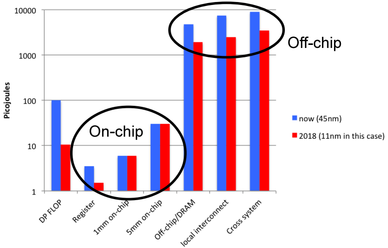
where A is large and non-symmetric.

- Many applications: e.g., scientific/engineering applications when solving PDEs
- Communication-avoiding Krylov method:
  - GMRES for solving large-scale problems
- Communication-Avoiding Preconditioners for CA methods
  - A domain decomposition framework for CA preconditioning
- Hybrid CPU/GPU cluster implementation

Communication-Avoiding Methods

- Communication:
  - Moving data between levels of memory
  - Moving data between processors in a network
- Communication-Avoiding: Reduce Communication (messages, volume)
  - Not Communication hiding

- Improves Time to solution and Reduces energy consumption
- More important in future architectures



(Image Courtesy: John Shalf, LBL)

Communication-Avoiding Iterative Methods

- Originally proposed 30 years ago for Conjugate Gradient (J. van Rosendale, 1983).
- Chronopoulos and Gear - "s-step iterative methods" (1989)
- R. Leland - The effectiveness of these methods (1989)
- Walker - Implementation of the GMRES method using Householder transformations (1988)
- E. de Sturler and H. A. van der Vorst - GMRES and CG, basis vectors (2005)
- M. Hoemmen (2010) - TSQR, "Communication-Avoiding" methods
- Two main problems:
  - "Good" basis vectors (works for practical 's')
  - Lack of preconditioners (This talk)

Preconditioners for Communication-Avoiding Iterative Methods

- Preconditioners that are like SpMV or that add no communication

  - Polynomial preconditioning, Sparse approximate inverse

  - CA-ILU(0) (L. Grigori and S. Moufawad, 2013)

  - Deflation based preconditioning (E. Carson 2014)

- Preconditioners that use low-rank like structures

  - Need changes to how the matrix is stored and no known evaluation with s-step methods

- Other related methods

  - s-step GMRES as bottom solver for multigrid (IPDPS 14)

  - Communication hiding pipelined Krylov methods do not have the preconditioning problem (P. Ghyssels et al., 2013)

  - Heirarchical Krylov Methods [L. McInnes et al.]

Restarted GMRES with GPUs

1 Generate Krylov Basis on GPUs: $O(m \cdot nnz(A) + m^2 n)$ flops

```
for j = 1, 2, ..., m do
  Sparse Matrix-Vector Multiply (SpMV (+ Precond)):
      q_{j+1} := A q_j
  Orthonormalization (Orth):
      q_{j+1} := q_{j+1} - Q_{1:j} Q_{1:j}^T q_{j+1}
end for
```

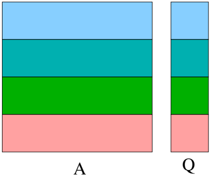
$\mathbf{q}_{j+1} := A\mathbf{q}_j$

$\mathbf{q}_{j+1} := \mathbf{q}_{j+1} - Q_{1:j}Q_{1:j}^T\mathbf{q}_{j+1}$

2 Solve Projected Subsystem on CPUs: $O(m^2)$ flops

small structured least-square problem

$\rightarrow$ restart with "best" initial vector $\mathbf{q}_1$ in $Q_{1:m}$



- generating basis vectors dominates computational cost.
  - distribute A and Q in a 1D block row among GPUs.
  - redundantly solve least-squares by each process.
- both *SpMV* and *Orth* require "expensive" communication:
  - point-to-point/neighborhood for *SpMV* (inter-GPU).
  - global all-reduces in *Orth* (inter-GPU).
  - data movements through local memory hierarchy (intra-GPU).

Communication-Avoiding Implementation of s-step GMRES

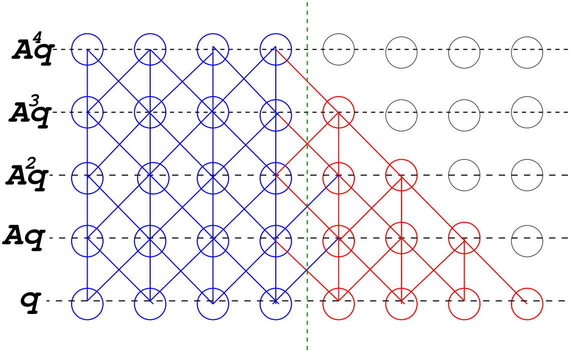
1. Generate Krylov Basis:
   for $j = 1, 1+s, \ldots, m$ **do**
   Matrix Powers Kernel (*MPK*):
   $\mathbf{q}_{k+1} := A\mathbf{q}_k$, **for** $k = j, \ldots, j+s-1$

   Block Orthogonalization (*BOrth*):
   orthogonalize $Q_{j+1:j+s}$ against $Q_{1:j}$
   Tall-skinny QR (*TSQR*):
   orthogonalize $Q_{j+1:j+s}$
   compute $H_{j:j+s-1,j+1:j+s}$
   end for
2. Solve Projected Subsystem on CPUs: $\sim O(m^2)$ flops.
   structured small least-square problem
   $\rightarrow$ restart with "best" initial vector $\mathbf{q}_1$ in $Q_{1:m}$.

- replace *SpMV* and *Ortho* with *MPK* and *BOrth*+*TSQR*.
- reduce comm by generating s vectors "at once"
  (e.g., replace BLAS-2 with BLAS-3).

Matrix Powers Kernel for a tridiagonal matrix

For a given starting vector $\mathbf{q}$, compute $A\mathbf{q}, A^2\mathbf{q}, \ldots, A^s\mathbf{q}$ (e.g., $s = 4$):

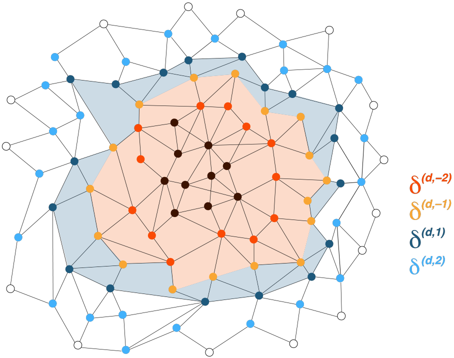


1. communicate required nonlocal elements for s-step between GPUs
2. apply s *SpMV*s with extra computation on shrinking ghost
   - local submatrix is expanded with s-level ghost

→ reduce inter-GPU latency by s (with redundant computation).

Matrix Powers Kernel for a general matrix ($s = 2$):

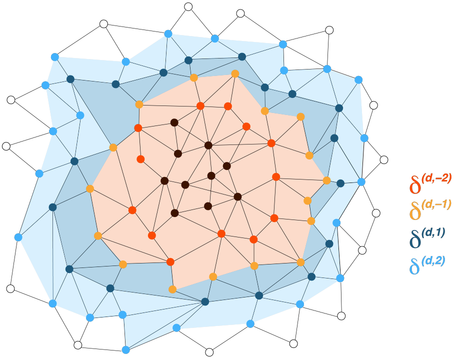
In adjacency graph of A,



- to compute local elements of $\mathbf{q}_{s+1}$,
  one *SpMV* requires local and nonlocal 1-level ghost elements

Matrix Powers Kernel for a general matrix ($s = 2$):

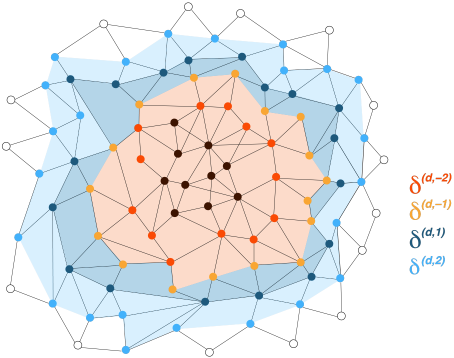
In adjacency graph of A,



- to compute local elements of $\mathbf{q}_{s+1}$,
  two *SpMV*s require local and nonlocal 2-level ghost elements.

Matrix Powers Kernel for a general matrix ($s = 2$):

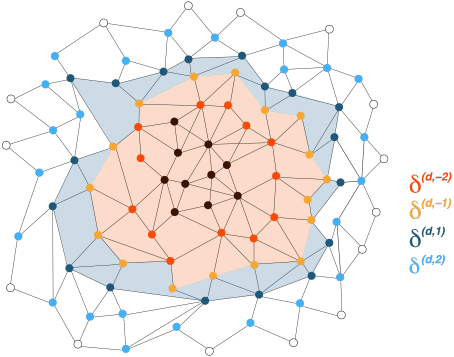
In adjacency graph of A,



- at 1st step of *MPK*,
  we perform *SpMV* with local and 2-level ghost elements of $\mathbf{q}_1$

Matrix Powers Kernel for a general matrix ($s = 2$):

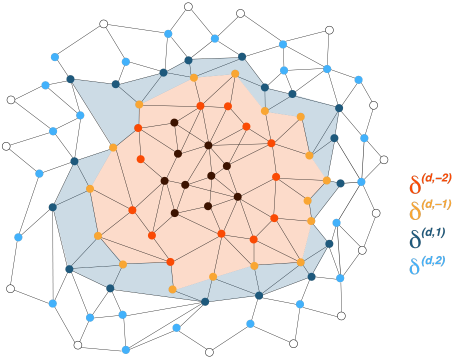
In adjacency graph of A,



- at 1st step of *MPK*,
  we perform *SpMV* with local and 2-level ghost elements of of $\mathbf{q}_1$
  → compute local and 1-level ghost elements of $\mathbf{q}_2$

Matrix Powers Kernel for a general matrix ($s = 2$):

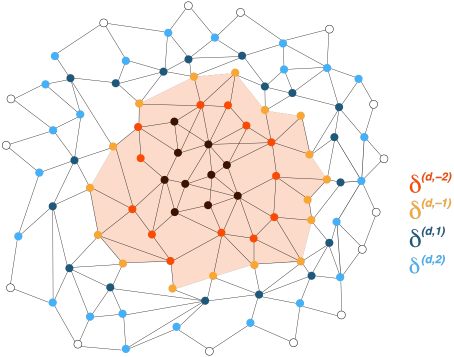
In adjacency graph of A,



- at 2nd step of *MPK*,
  we perform *SpMV* with local and 1-level ghost elements of $\mathbf{q}_2$

Matrix Powers Kernel for a general matrix ($s = 2$):

In adjacency graph of A,



- at 2nd step of *MPK*,
  we perform *SpMV* with local and 1-level ghost elements
  → compute local elements of $\mathbf{q}_3$

Our Matrix Powers Kernel Implementation with multiple GPUs

Initialize *MPK*:
  set up communication pattern.
  expand local submatrix with ghost elements, etc.

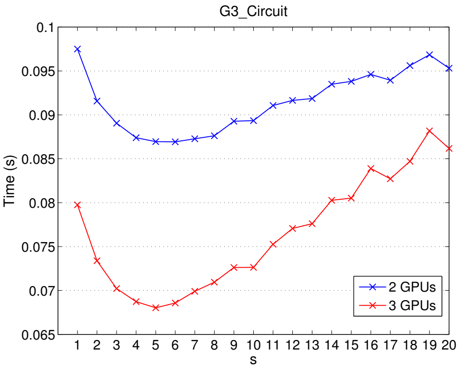
CA-GMRES with GPUs.
1. Generate Krylov Basis:
   for $j = 1, 1+s, \ldots, m$ **do**
      MPK:
        Inter-GPU Communication: each MPI process
          1. CPU ← GPUs using CUDA
          2. CPUs ⟷ CPUs using MPI
          3. CPU → GPUs using CUDA
        GPU Kernel:
          for $k = 1, 2, \ldots s$ do
            SpMV with local and k-level ghost elements
          end for
      BOrth and *TSQR*.
   end for
2. Solve projected system.

- currently optimized only for inter-GPU communication, and not for intra-GPU communication

Matrix Powers Kernel Performance on a node

Our *MPK* requires overheads, but reduces inter-GPU latency:

- additional memory to store "ghost" elements
- addition computation for *SpMV* with "ghost" elements
- potentially, increasing total inter-GPU communication volume.



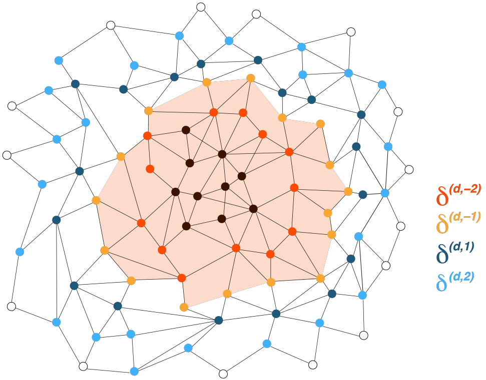
Integrating preconditioner into *MPK*

Apply *Preco* followed by *SpMV* at each step of *MPK*

for $k = j, j+1, \ldots, j+s-1$ **do**
    Preco: $\mathbf{q}_{k+1} := M^{-1}\mathbf{q}_k$
    SpMV: $\mathbf{q}_{k+1} := A\mathbf{q}_{k+1}$
end for

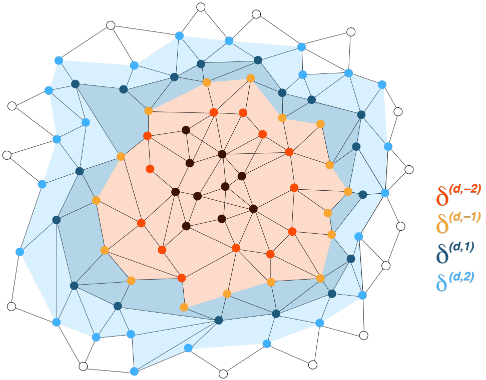
- focus on right-preconditioning, generating $\kappa(AM^{-1}, \mathbf{q}_1)$
  - can be easily extended to left-preconditioning
- not increase inter-GPU comm from what is already needed by *MPK*

Challenge: block Jacobi preconditioner increases communication



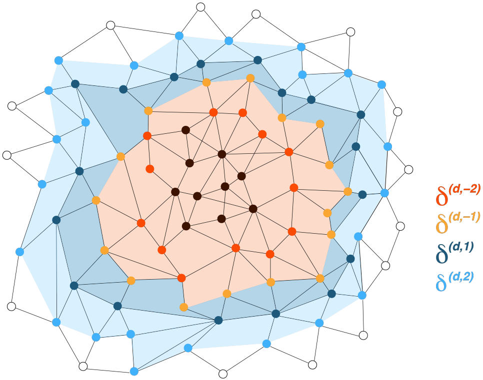
- each GPU *Precon* local elements of $\mathbf{q}_1$, solving its local sub-problem.
- *SpMV* requires "preconditioned" s-level ghost elements of $\mathbf{q}_1$
  $\rightarrow$ additional communication

Challenge: block Jacobi preconditioner increases communication



- each GPU *Precon* local elements of $\mathbf{q}_1$, solving its local sub-problem.
- *SpMV* requires "preconditioned" s-level ghost elements of $\mathbf{q}_1$ $\rightarrow$ additional communication

Challenge: block Jacobi preconditioner increases communication

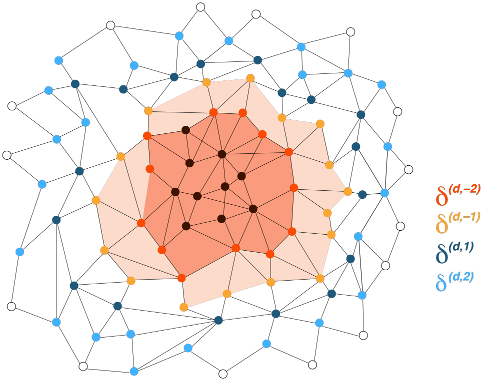


- Solution 1: consider $2 \times s$ levels of ghost (*Preco* then *SpMV*)

  "global" preconditioner, potentially large overhead e.g., CA-ILU(0) [Grigori et.al'14]

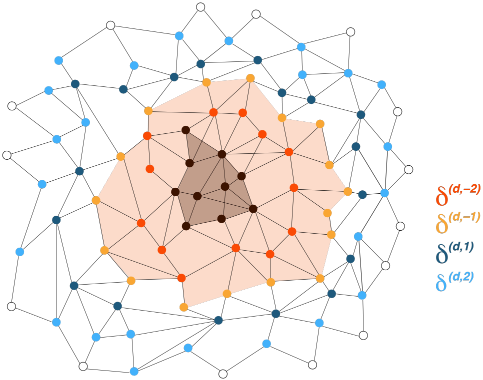
- Solution 2: consider what we can do without additional comm

Domain Decomposition Preconditioner for CA-Krylov



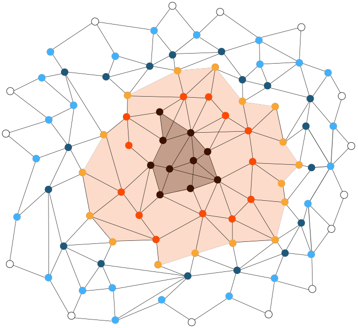
- for 1st *SpMV*, neighboring GPUs require elements on 1-level underlap
  - local elements reachable from other subdomains by one edge

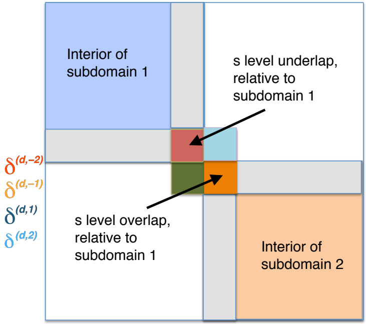
Domain Decomposition Preconditioner for CA-Krylov



- for 2nd *SpMV*, neighboring GPUs require elements on 2-level underlap
  - local elements reachable from other subdomains by two edges

Domain Decomposition Preconditioner for CA-Krylov

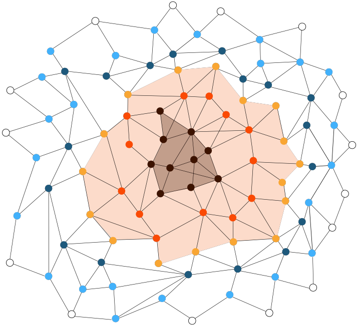


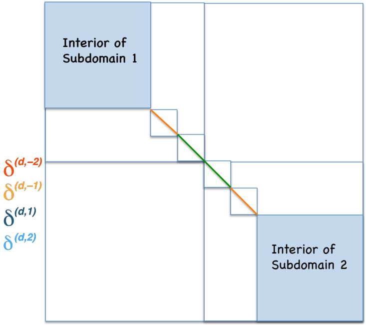


In order to "localize" effects of preconditioner,

- form "interior" by removing s-level "underlap"
- apply "local" preconditioner on "interior" and "underlap/ghost," separately
  - ILU(k or τ), SAI(k), Jacobi, GaussSeidel, etc. on "interior"
  - diagonal Jacobi on "underlap" and "ghost"

Domain Decomposition Preconditioner for CA-Krylov



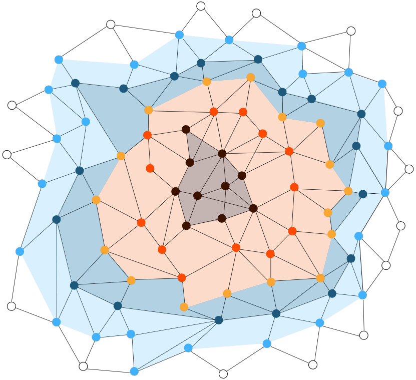


In order to "localize" effects of preconditioner,

- form "interior" by removing s-level "underlap"
- apply "local" preconditioner on "interior" and "underlap/ghost," separately
  - ILU(k or τ), SAI(k), Jacobi, GaussSeidel, etc. on "interior"
  - diagonal Jacobi on "underlap" and "ghost"

Domain Decomposition Preconditioner for CA-Krylov

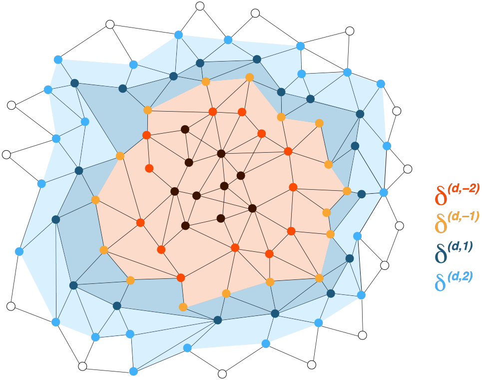
For *Precon* at 1st step of *MPK*,



- local preconditioning on interior and 2-level underlap/ghost of $\mathbf{q}_1$
  - ILU(k or τ), SAI(k), Jacobi, GaussSeidel, etc. on interior
  - diagonal Jacobi on underlap and 2-level ghost

Domain Decomposition Preconditioner for CA-Krylov

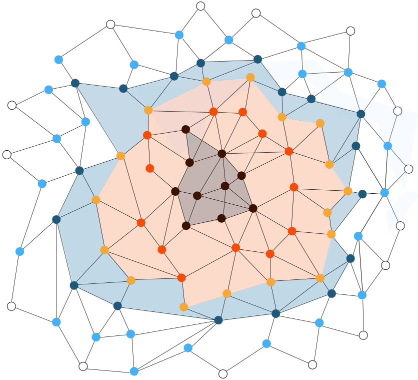
For *SpMV* at 1st step of *MPK*,



- *SpMV* with local subdomain and 2-level ghost of $\mathbf{q}_1$

Domain Decomposition Preconditioner for CA-Krylov

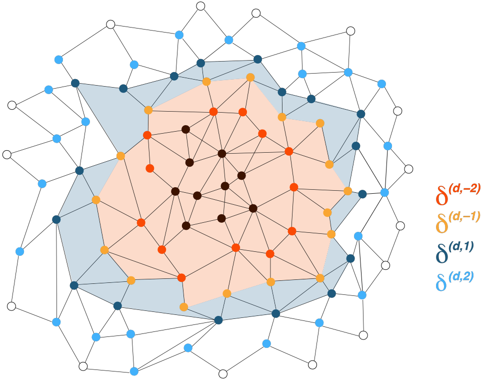
For *Precon* at 2nd step of *MPK*,



- local preconditioning on interior and 1-level underlap/ghost of $\mathbf{q}_2$
  - ILU(k or τ), SAI(k), Jacobi, GaussSeidel, etc. on interior
  - diagonal Jacobi on underlap and 1-level ghost

Domain Decomposition Preconditioner for CA-Krylov

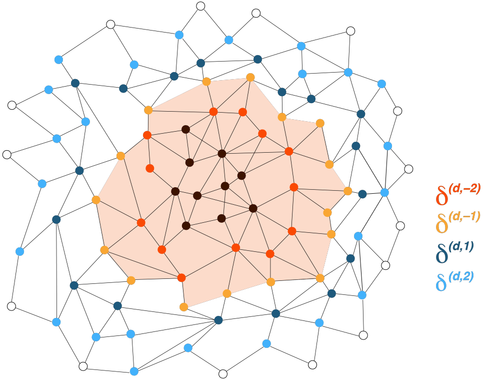
For *SpMV* at 2nd step of *MPK*,



- *SpMV* with local subdomain and 1-level ghost of $\mathbf{q}_2$

Domain Decomposition Preconditioner for CA-Krylov

For *SpMV* at 2nd step of *MPK*,

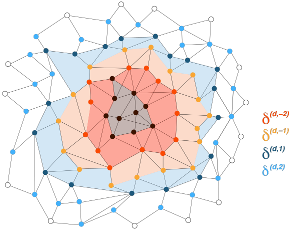


- apply *SpMV* with local subdomain and 1-level ghost
  $\rightarrow$ compute local elements of $\mathbf{q}_3$

Domain Decomposition Preconditioner for CA-Krylov

Summary: at jth step of *MPK*,

- effects of interior precond grows (i.e., sth to $(s-j+2)$th levels of underlap)
- underlap required by neighbors shrinks (i.e., $(s-j+1)$th to 1st levels)

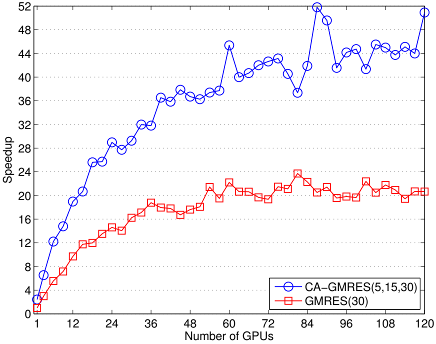


- no increase in inter-GPU communication
- any local preconditioner/solver on interior
  - ILU(k or τ), SAI(k), Jacobi, Gauss-Seidel, etc.
- preconditioner on underlap/ghost
  - diagonal Jacobi: interior precond propagates only within subdomain
  - extension in current work

Experimental Setup

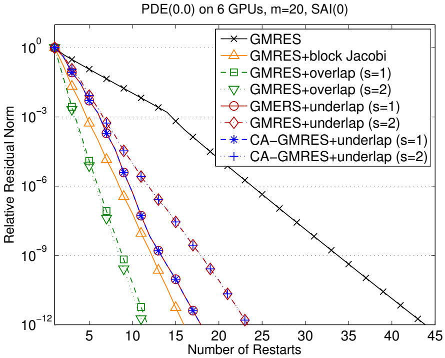
- graph partitioning (e.g., METIS) for load balance and small communication
- local matrix reordering (e.g., METIS, RCM) for performance
(e.g., nested dissection for triangular solves on GPU)
- matrix equilibration for numerical stability
- Newton basis to enhance *MPK* stability, $\mathbf{v}_{k+1} = \Pi_{i=1}^{k}(A - \theta_i)\mathbf{q}_1$.
shifts θ_i are Ritz values from first restart loop (GMRES)
- *Precon* computed on CPU (e.g., ITSOL, ParaSail), and copied and apply it on GPU (e.g., trsv/spmv of CuSPARSE)
- Keeneland at Georgia Tech
each node has 2×6 Intel Xeon + 3 NDIVIA M2090.
- Test matrix: PDE(α): $n \approx 10^6$, symmetric but can be indefinite
  - larger α makes it more ill-conditioned
  - $\alpha > 1$ makes it indefinite

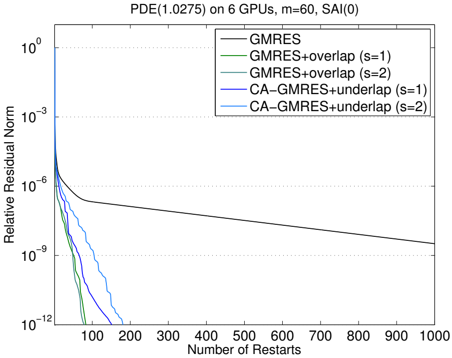
CA-GMRES Performance (speedups vs. GMRES on one GPU)



- obtained speedups of up to 2.5
  - more details in our IPDPS/SC'14 papers.

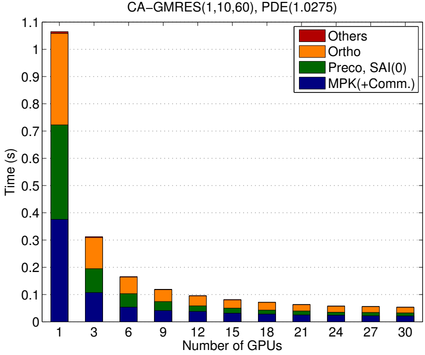
Convergence Results

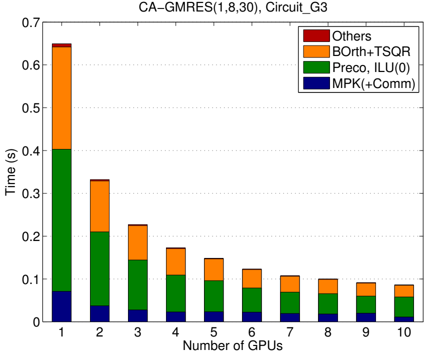




- DD preconditioner improves the convergence
  - faster convergence with a larger overlap
  - slower convergence with a larger underlap

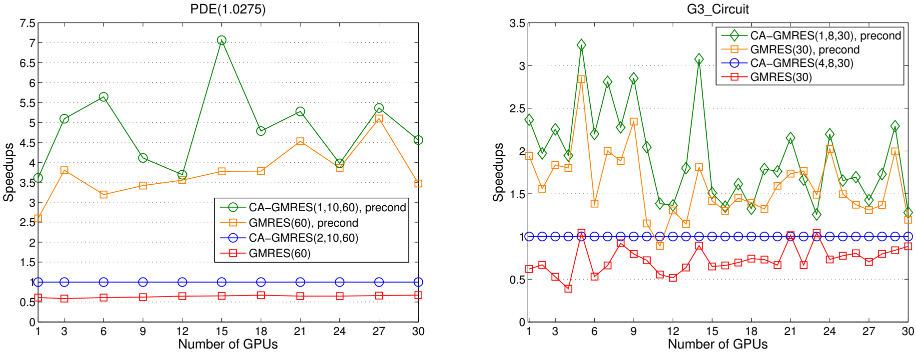
Restart Cycle Time Breakdown





- SAI(0) is used for `PDE(1.0275)`
- ILU(0) is required for `Circuit_G3`

Time to Solution Speedups vs. CA-GMRES



- speedups of up to 7.5× over CA-GMRES without preconditioner
- speedups of up to 1.7× over GMRES with preconditioner
  - Our *MPK* is not optimized on a GPU
  - On GPUs, *Ortho* performs great, and *SpMV*/*Preco* can dominate easily.

Summary

- proposed domain decomposition preconditioners for CA-Krylov
  - do not increase inter-process communication
  - can use any solver on interior problem
- presented results of a block Jacobi like implementation
  - diagonal Jacobi on underlap/ghost
  - potential to improve convergence/performance
    - over precond GMRES or standard CA-GMRES

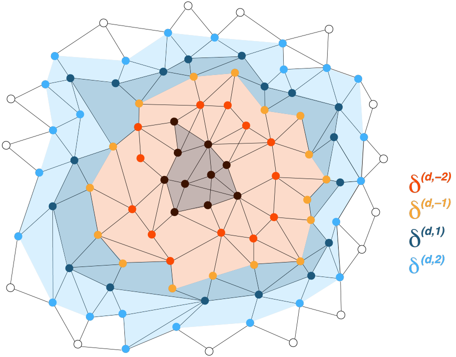
Future work

- improving performance
  - utilizing CPU, partitioning, etc.
- underlap/ghost preconditioning
- more extensions (e.g., “flexible” preconditioner)

Thank you!!

Domain Decomposition Preconditioner for CA-Krylov

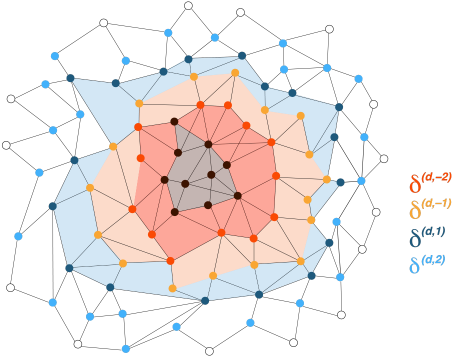
For SpMV at 1st step of MPK,



- perform SpMV with local subdomain and 2nd-level ghost

Domain Decomposition Preconditioner for CA-Krylov

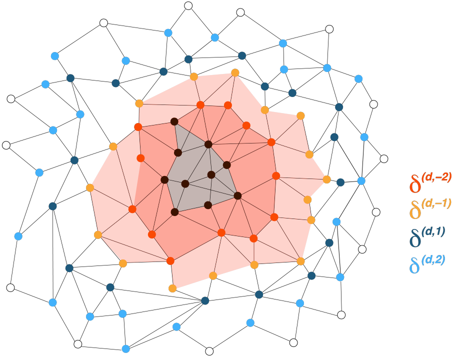
After SpMV at 1st step of MPK,



- effects of interior precond propagates into 2nd-level underlap

Domain Decomposition Preconditioner for CA-Krylov

After SpMV at 2nd step of MPK,



- effects of interior precond pro pages into 1st-level ghost

Matrix Powers Kernel Performance on a node