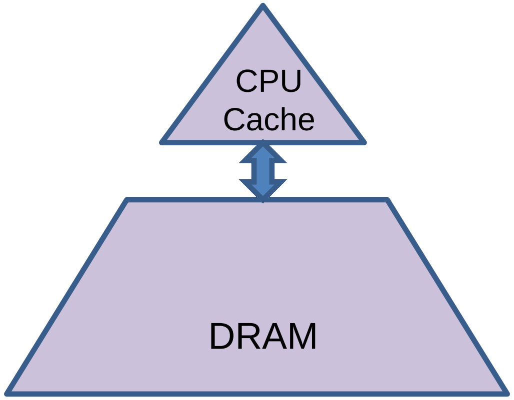
Exploiting Low-Rank Structure in Computing Matrix Powers with Applications to Preconditioning

Erin C. Carson
Nicholas Knight, James Demmel, Ming Gu
U.C. Berkeley

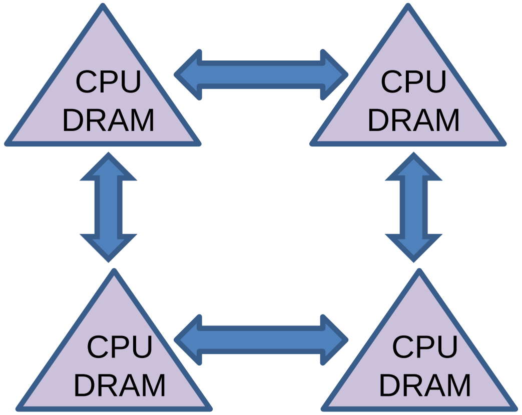
Motivation: The Cost of an Algorithm

- Algorithms have 2 costs: Arithmetic (flops) and movement of data (communication)
- Assume simple model with 3 parameters:
  - α – Latency, β – Reciprocal Bandwidth, γ – Flop Rate
  - Time to move n words of data is $\alpha + n\beta$



Sequential

- Problem: Communication is the bottleneck on modern architectures
  - α and β improving at much slower rate than γ



Parallel

- Solution: Reorganize algorithms to ***avoid communication***

Motivation: Krylov Subspace Methods

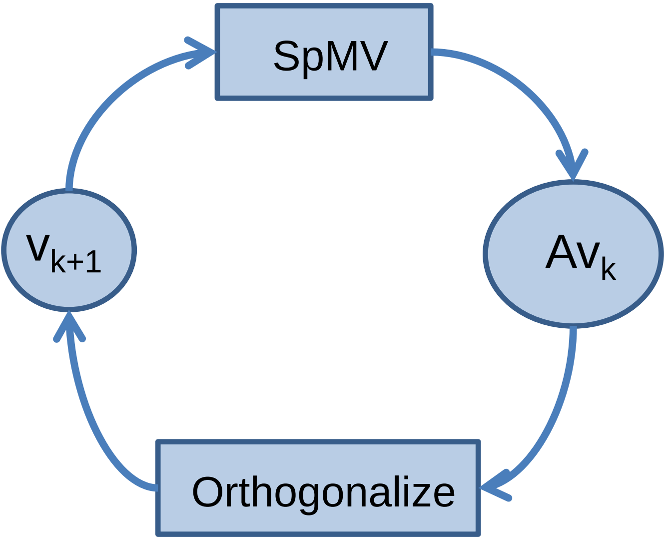
- Krylov Subspace Methods (KSMs) are iterative methods commonly used in solving large, sparse linear systems of equations
  - Krylov Subspace of dimension k with matrix A and vector v:

$$\mathcal{K}_k(A,\, v) = \mathrm{span}\{v,\, Av,\, A^2v,\, \ldots,\, A^{k-1}v\}$$

  - Work by iteratively adding a dimension to the expanding Krylov Subspace (SpMV) and then choosing the “best” solution from that subspace (vector operations)
- Problem: Krylov Subspace Methods are communication-bound
  - SpMV and global vector operations in every iteration

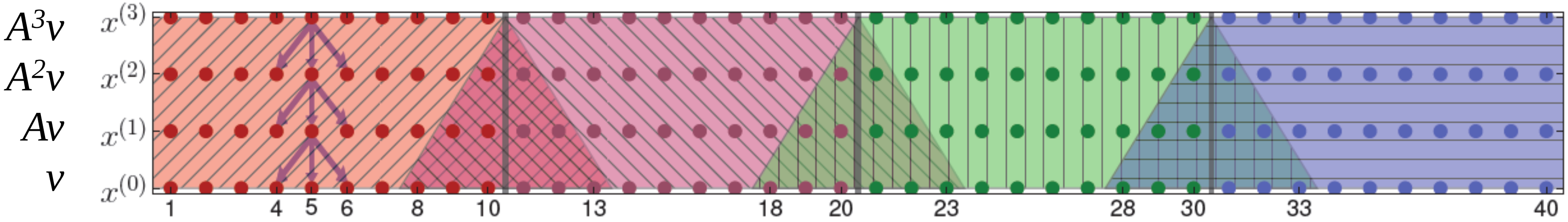
Avoiding Communication in Krylov Subspace Methods

- We need to break the dependency between communication bound kernels and KSM iterations
- Idea: Expand the subspace *s* dimensions (*s* SpMVs with *A*), then do *s* steps of refinement
- To do this we need two new Communication-Avoiding kernels
  - “Matrix Powers Kernel” replaces SpMV
  - “Tall Skinny QR” (TSQR) replaces orthogonalization operations



The Matrix Powers Kernel

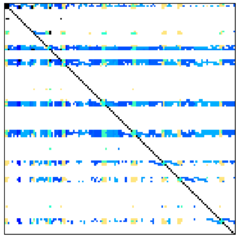
- Given A, v, s, and degree j polynomials $\rho_j, j = 0{:}\, s$ defined by a 3-term recurrence, the matrix powers kernel computes
$$\{\rho_0(A)v, \rho_1(A)v, \rho_2(A)v, \dots, \rho_s(A)v\}$$
- The matrix powers kernel computes these basis vectors only reading/communicating A $o(1)$ times!
  - Parallel case: Reduces latency by a factor of s at the cost of redundant computations



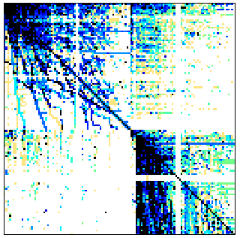
Parallel Matrix Powers algorithm for tridiagonal matrix example.
4 processors, $n = 40, s = 3$

Matrix Powers Kernel Limitations

- Asymptotic reduction in communication requires that *A* is well-partitioned
  - "Well-partitioned"- number of redundant entries required by each partition is small – the graph of our matrix has a good cover
- With this matrix powers algorithm, we can't handle matrices with dense components
  - Matrices with dense low-rank components appear in many linear systems (e.g., circuit simulations, power law graphs), as well as preconditioners (e.g., Hierarchical Semiseparable (HSS) matrices)
  - Can we exploit low-rank structure to avoid communication in the matrix powers algorithm?



ASIC_680k: circuit simulation matrix. Sandia.



webbase: web connectivity matrix. Williams et al.

Blocking Covers Approach to Increasing Temporal Locality

- Relevant work:
  - Leiserson, C.E. and Rao, S. and Toledo, S. Efficient out-of-core algorithms for linear relaxation using blocking covers. Journal of Computer and System Sciences, 1997.
- Blocking Covers Idea:
  - According to Hong and Kung’s Red-Blue Pebble game, we can’t avoid data movement if we can’t find a good graph cover
  - What if we could find a good cover by removing a subset of vertices from the graph? (i.e., connections are locally dense but globally sparse)
  - Relax the assumption that the DAG must be executed in order
  - Artificially restrict information from passing through removed vertices (“blockers”) by treating their state variables symbolically, maintain dependencies among symbolic variables as matrix

Blocking Covers Matrix Powers Algorithm

- Split A into sparse and low-rank dense parts, $A = D + UV^T$
- In our matrix powers algorithm, the application of V^T requires communication, so we want to limit the number these operations
- Then we want to compute (assume monomial basis for simplicity)

$$\{v, Av, \dots, A^s v\} = \{v, (D + UV^T)v, \dots, (D + UV^T)^s v\}$$

- We can write the jth basis vector as

$$c_j = (D + UV^T)^j v = Dc_{j-1} + UV^T c_{j-1} = D^j v + \sum_{k=1}^{j} D^{k-1} UV^T c_{j-k}$$

- Where the $V^T c_{j-k}$ quantities will be the values of the “blockers” at each step.
- We can premultiply the previous equation by V^T to write a recurrence:

$$V^T c_j = V^T D^j v + \sum_{k=1}^{j} \left(V^T D^{k-1} U\right)\left(V^T c_{j-k}\right)$$

Blocking Covers Matrix Powers Algorithm

Phase 0: Compute $\{U, DU, D^2U, \dots, D^{s-2}U\}$ using the matrix powers kernel. Premultiply by V^T.

Phase 1: Compute $\{v, Dv, D^2v, \dots, D^{s-1}v\}$ using the matrix powers kernel. Premultiply by V^T.

Phase 2: Using the computed quantities, each processor backsolves for $V^T c_j$ for $j = 1{:}\, s - 1$

Phase 3: Compute the c_j vectors, interleaving the matrix powers kernel with local $UV^T c_{j-1}$ multiplications

$$V^T c_j = V^T D^j v + \sum_{k=1}^{j} \left(V^T D^{k-1} U\right)\left(V^T c_{j-k}\right)$$

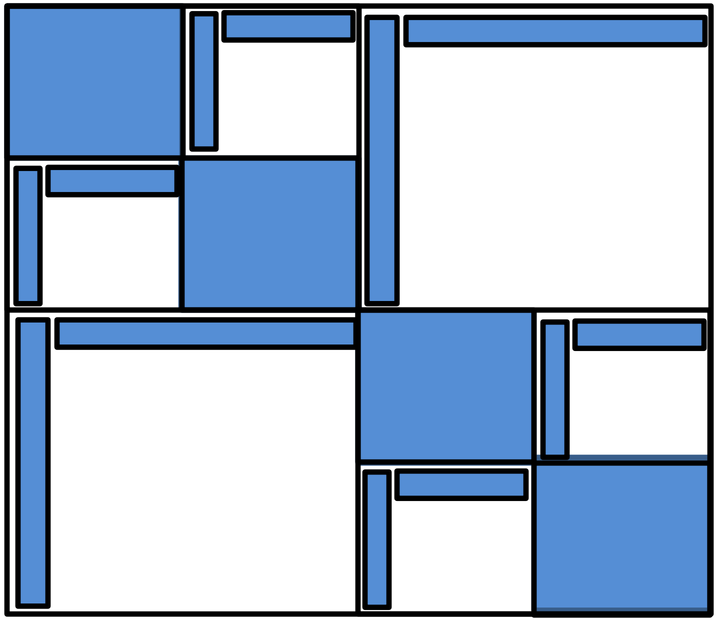
$$c_j = D c_{j-1} + UV^T c_{j-1}$$

Asymptotic Costs

Phase	Flops	Words Moved	Messages
0	$Akx(D, U, s-2) \; + \; O(\frac{sr^2n}{p})$	$O(sr^2 \log p)$ + r(ghost zones, D^{s-2})	$O(\log p)$
1	$Akx(D, v, s-1) \; + \; O(\frac{srn}{p})$	$O(sr \log p)$ + (ghost zones, D^{s-1})	$O(\log p)$
2	$O(s^2r^2)$	0	0
3	$Akx(D, v, s) + O(\frac{srn}{p})$	0	0

	Flops	Words Moved	Messages
Total Online (CA)	$2 \times Akx(D, v, s) + O(\frac{srn}{p})$	$O(sr \log p)$ + (ghost zones, D^{s-1})	$O(\log p)$
Standard Alg.	$s \times Akx(D, v, 1) \; + \; O(\frac{srn}{p})$	$O(sr \log p)$ + s(ghost zones, D)	$O(s \log p)$

Extending the Blocking Covers Matrix Powers Algorithm to HSS Matrices



HSS Structure:

- l-level binary tree
- Off-diagonal blocks have rank r
- Can write A hierarchically:

$$D_{0;1} = A$$

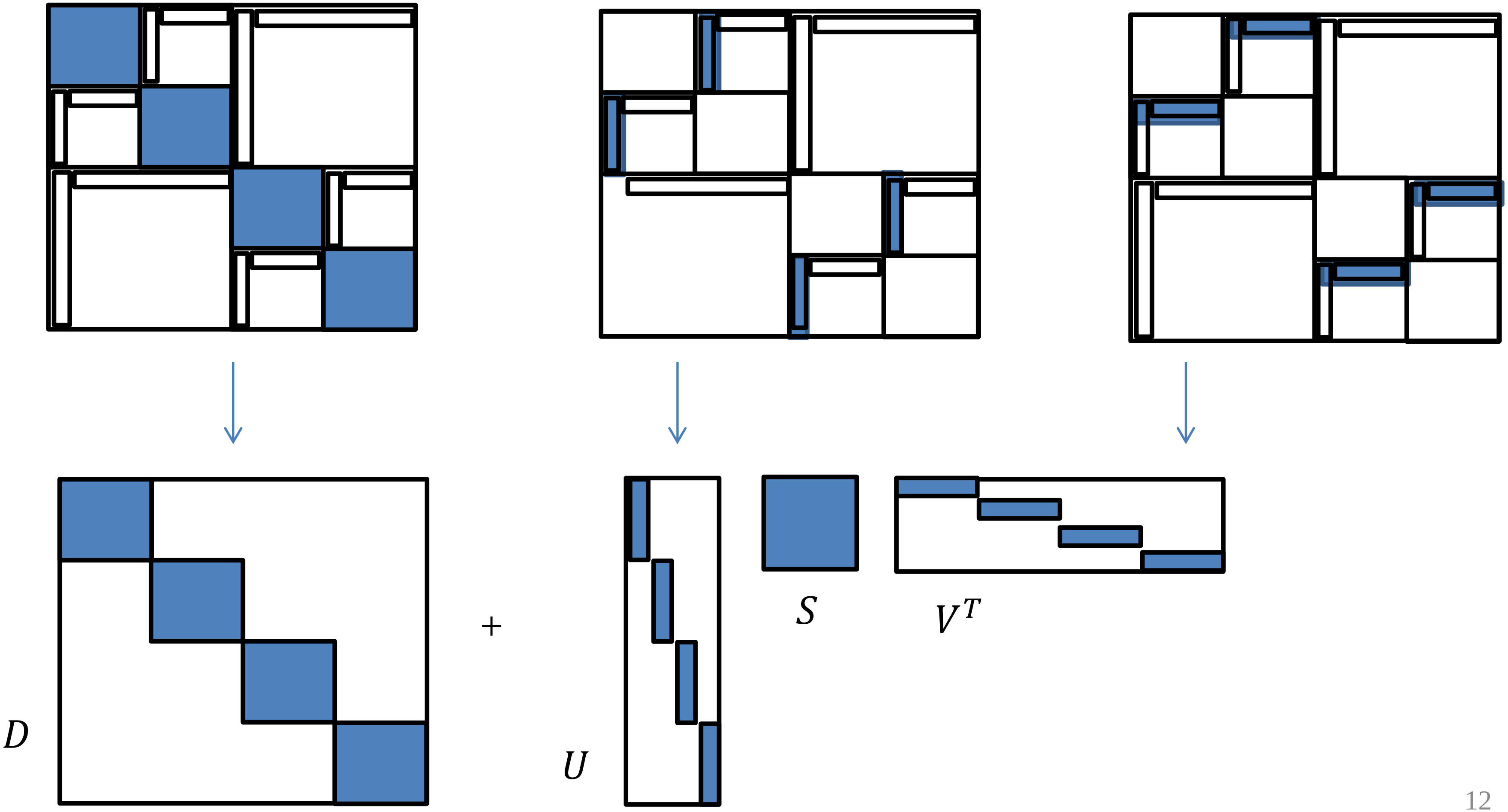
$$D_{k;i} = \begin{pmatrix} D_{k+1;2i-1} & U_{k+1;2i-1} B_{k+1;2i-1,2i} V^T{}_{k+1;2i} \\ U_{k+1;2i} B_{k+1;2i,2i-1} V^T{}_{k+1;2i-1} & D_{k+1;2i} \end{pmatrix}$$

- Can define translations for row and column bases, i.e:

$$U_{k;i} = \begin{pmatrix} U_{k+1;2i-1} R_{k+1;2i-1} \\ U_{k+1;2i} R_{k+1;2i} \end{pmatrix} \qquad V_{k;i} = \begin{pmatrix} V_{k+1;2i-1} W_{k+1;2i-1} \\ V_{k+1;2i} W_{k+1;2i} \end{pmatrix}$$

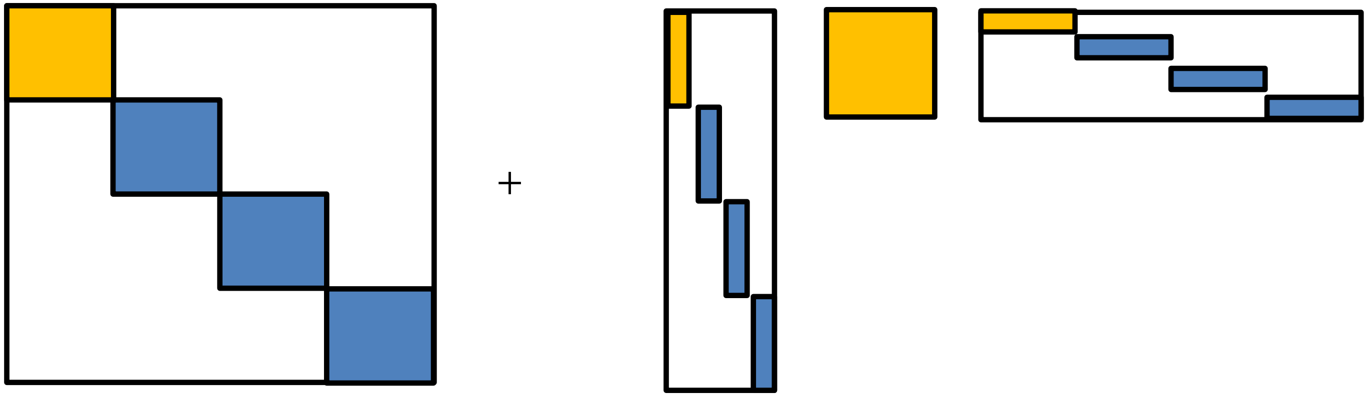
Exploiting Low-Rank Structure

- Matrix can be written as $D + USV^T$
- S composed of R, W, B's translation operations (S is not formed explicitly)



Parallel HSS Akx Algorithm

- Data Structures:
  - Assume $p = 2^l$ processors
  - Each processor i owns
    - D_i , dense diagonal block, dimension $(n/p \times n/p)$
    - V_i, dimension $(r \times n/p)$
    - U_i, dimension $(r \times n/p)$
    - x_i, $(n/p \times 1)$ piece of source vector
    - All matrices $R, W, B,$
      - These are all small $O(2^l r^2)$ sized matrices, assumed they fit on each proc.



Extending the Algorithm

- Only change needed is in Phase 2 (backsolving for $V^T c_j$)
  - Before, we computed, for $j = 1 : s - 1$

$$V^T c_j = V^T D^j v + \sum_{k=1}^{j} \left(V^T D^{k-1} U\right)\left(V^T c_{j-k}\right)$$

  - Now, we can exploit hierarchical semiseparability:
  - For $j = 1 : s - 1$, first compute

$$g_l = V^T D^j v + \sum_{k=1}^{j} \left(V^T D^{k-1} U\right)\left(V^T c_{j-k+1}\right)$$

Extending the Algorithm

- Then each processor locally performs upsweep and downsweep:

```
for $y = l - 1 : 1$
    $g_y = \begin{bmatrix} \begin{bmatrix} {W^T}_{y+1;1} & {W^T}_{y+1;2} \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} {W^T}_{y+1;2(2^y)-1} & {W^T}_{y+1;2(2^y)} \end{bmatrix} \end{bmatrix} g_{y+1}$
$f_0 = (0)$
for $y = 0 : l - 1$
    $f_{y+1} = \begin{bmatrix} B_{y+1;1,2} & & \\ & \ddots & \\ & & B_{y+1;2^{y+1},2^{y+1}-1} \end{bmatrix} g_y + \begin{bmatrix} \begin{bmatrix} R_{y+1;1} \\ R_{y+1;2} \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} R_{y+1;2^{y+1}-1} \\ R_{y+1;2^{y+1}} \end{bmatrix} \end{bmatrix} f_y$
$V^T c_j = f_l$
```

- At the end, each processor has locally computed the $V^T c_j$ recurrence for the j^{th} iteration (additional sr^2p flops in Phase 2)

HSS Matrix Powers Communication and Computation Cost

- Offline (Phase 0)
  - Flops: $Akx(D, U, s) + O(\frac{sr^2n}{p})$
  - Words Moved: $O(r^2 s \log p)$
  - Messages: $O(\log p)$
- Online (Phases 1, 2, 3)
  - Flops: $2 \times Akx(D, x, s) + O(\frac{srn}{p})$
  - Words Moved: $O(rs \log p)$
  - Messages: $O(\log p)$

- **Same flops (asymptotically) as *s* iterations of standard HSS Matrix-Vector Multiply algorithm**

- **Asymptotically reduces messages by factor of *s*!**

Future Work

- Auto-tuning: Can we automate the decision of which matrix powers kernel variant to use?
  - What should be the criteria for choosing blockers?
- Stability
  - How good is the resulting (preconditioned) Krylov basis?
- Performance studies
  - How does actual performance of HSS matrix powers compare to *s* HSS matrix-vector multiplies?
- Extension to other classes of preconditioners
- Can we apply the blocking covers approach to other algorithms with similar computational patterns?