# Exploiting Data Sparsity in Parallel Matrix Powers Computations

Nicholas Knight(✉), Erin Carson, and James Demmel

University of California, Berkeley, USA
{knight,ecc2z,demmel}@cs.berkeley.edu

**Abstract.** We derive a new parallel communication-avoiding matrix powers algorithm for matrices of the form $A = D + USV^H$, where $D$ is sparse and $USV^H$ has low rank and is possibly dense. We demonstrate that, with respect to the cost of computing $k$ sparse matrix-vector multiplications, our algorithm asymptotically reduces the parallel latency by a factor of $O(k)$ for small additional bandwidth and computation costs. Using problems from real-world applications, our performance model predicts up to 13× speedups on petascale machines.

**Keywords:** Communication-avoiding · Matrix powers · Graph cover · Hierarchical matrices · Parallel algorithms

## 1 Introduction

The runtime of an algorithm can be modeled as a function of *computation* cost, proportional to the number of arithmetic operations, and *communication* cost, proportional to the amount of data movement. On modern computers, the time to move one word of data is much greater than the time to complete one arithmetic operation. Technology trends indicate that the performance gap between communication and computation will only widen in future computers, resulting in a paradigm shift in the design of high-performance algorithms: to achieve efficiency, one must focus on *communication-avoiding* approaches.

We consider a simplified machine model, where a parallel machine consists of $p$ processors, each able to perform arithmetic operations on their $M$ words of *local memory*. Processors communicate point-to-point *messages* of $n \leq M$ contiguous words, taking $\alpha + \beta n$ seconds on both sender and receiver, over a completely connected network (no contention), and each processor can send or receive at most one message at a time. For simplicity, we do not model overlapping communication and computation. Given an algorithm's *latency cost*, number of messages sent, *bandwidth cost*, number of words moved, and *arithmetic (flop) cost*, the number of arithmetic operations performed, we estimate the runtime $T$ (along the critical path) on a parallel machine with latency $\alpha$, reciprocal bandwidth $\beta$, and arithmetic (flop) rate $\gamma$ as

$$T = (\#\text{messages} \cdot \alpha) + (\#\text{words moved} \cdot \beta) + (\#\text{flops} \cdot \gamma). \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2013, Part I, LNCS 8384, pp. 15–25, 2014.
DOI: 10.1007/978-3-642-55224-3_2, 

Computing $k$ repeated sparse matrix-vector multiplications (SpMVs), or, a *matrix powers* computation, with $A \in \mathbb{C}^{n \times n}$ and $x \in \mathbb{C}^{n \times q}$, where typically $q \ll n$, can be written as

$$K_{k+1}(A, x, \{p_j\}_{j=0}^{k}) := [x^{(0)}, \ldots, x^{(k)}] := [p_0(A)x, p_1(A)x, \ldots, p_k(A)x], \quad (2)$$

where $p_j$ is a degree-$j$ polynomial. Due to a small ratio of arithmetic operations to data movement, the performance of this computation is bound by communication on modern computers. Matrix powers computations constitute a core kernel in a variety of applications, including steepest descent algorithms and Krylov subspace methods for linear systems and eigenvalue problems, including the power method to compute PageRank.

Previous efforts have produced parallel communication-avoiding matrix powers algorithms to compute (2) that achieve an $O(k)$ reduction in parallel latency cost versus computing $k$ repeated SpMVs for a set number of iterations [4, 11]. This improvement is only possible if $A$ is *well partitioned* (to be defined in Sect. 1.1). Although such advances show promising speedups for many problems, the requirement that $A$ is well partitioned often excludes matrices with dense components, even if those components have low rank (*data sparsity*). In this work, we derive a new parallel communication-avoiding matrix powers algorithm for matrices of the form $A = D + USV^H$, where $D$ is well partitioned and $USV^H$ may not be well partitioned but has low rank. (Recall $x^H = \overline{x}^T$ denotes the Hermitian transpose of $x$.) There are many practical situations where such structures arise, including power-law graph analysis and circuit simulation. Hierarchical ($\mathcal{H}$-) matrices (e.g., [1]), common preconditioners for Krylov subspace methods, also have this form. Our primary motivation is enabling preconditioned communication-avoiding Krylov subspace methods, where the preconditioned system has hierarchical semiseparable (HSS) structure. There is a wealth of literature related to communication-avoiding Krylov subspace methods; we direct the reader to the thesis of Hoemmen for an overview [5, Sects. 1.5 and 1.6].

With respect to the cost of computing $k$ SpMVs, our algorithm asymptotically reduces parallel latency by a factor of $O(k)$ with only small additional bandwidth and computational costs. Using a detailed complexity analysis for an example HSS matrix, our model predicts up to 13× speedups over the standard algorithm on petascale machines. Our approach is based on the application of a blocking covers technique [9] to communication-avoiding matrix powers algorithms [4,10]. We briefly review these works below.

### 1.1 The Blocking Covers Technique

Hong and Kung [6] prove a lower bound on data movement for a sequential matrix powers computation on a regular mesh. Given directed graph $G = (V, E)$ representing nonzeros of $A$, vertex $v \in V$, and constant $\tau \geq 0$, let the *$\tau$-neighborhood* of $v$, $N^{(\tau)}(v)$, be the set of vertices in $V$ such that $u \in N^{(\tau)}(v)$ implies there is a path of length at most $\tau$ from $u$ to $v$; a *$\tau$-neighborhood-cover* of $G$ is a sequence of subgraphs, $\mathcal{G} = \{G_i = (V_i, E_i)\}_{i=1}^{k}$, such that $\forall v \in V$,

$\exists G_i \in \mathcal{G}$ for which $N^{(\tau)}(v) \subseteq V_i$ [9]. If $G$ has a $\tau$-neighborhood cover with $O(|E|/M)$ subgraphs, each with $O(M)$ edges where $M$ is the size of the primary memory, Hong and Kung's method reduces data movement by a factor of $\tau$ over computing (2) column-wise. A matrix that meets these constraints is also frequently called *well partitioned* [4] (we use this terminology for the parallel case as well).

Certain graphs with low diameter (e.g., multigrid graphs) may not have $\tau$-neighborhood covers that satisfy these memory constraints. Leiserson et al. overcome this restriction by "removing" a set $B \subseteq V$ of *blocker* vertices, chosen such that the remaining graph $V - B$ is well partitioned [9]. Let the $\tau$-neighborhood with respect to $B$ be defined as $N_B^{(\tau)}(v) = \{u \in V : \exists \text{ path } u \to u_1 \to \cdots \to u_t \to v, \text{ where } u_i \in V - B \text{ for } i \in \{1, \ldots, t < \tau\}\}$. Then a $(\tau, r, M)$-blocking cover of $G$ is a pair $(\mathcal{G}, \mathcal{B})$, where $\mathcal{B} = \{B_i\}_{i=1}^k$ is a sequence of subsets of $V$ such that: (1) $\forall i \in \{1, \ldots, k\}, M/2 \leq |E_i| \leq M$, (2) $\forall i \in \{1, \ldots, k\}, |B_i| \leq r$, (3) $\sum_{i=1}^k |E_i| = O(|E|)$, and (4) $\forall v \in V, \exists G_i \in \mathcal{G}$ such that $N_{B_i}^{(\tau)}(v) \subseteq V_i$ [9]. Leiserson et al. present a 4 phase sequential matrix powers algorithm that reduces the data movement by a factor of $\tau$ over the standard method if the graph of $A$ has a $(\tau, r, M)$-blocking cover that meets certain criteria. Our parallel algorithm is based on a similar approach. Our work generalizes the blocking covers approach [9], both to the parallel case and to a larger class of data-sparse matrix representations.

### 1.2 Parallel Matrix Powers Algorithms

Parallel variants of matrix powers, for both structured and general sparse matrices, are presented in the thesis of Mohiyuddin [10], which summarizes and elaborates upon previous work and implementations [4,11]. We review two of these parallel matrix powers algorithms, referred to as PA0, the naïve algorithm for computing (2) via $k$ SpMV operations, and PA1, a communication-avoiding variant. We assume the polynomials $\{p_l\}_{l=0}^k$ in (2) satisfy a recurrence,

$$p_0(z) := 1, \qquad p_{j+1}(z) = \left( z p_j(z) - \sum\nolimits_{i=0}^{j} h_{i,j} p_i(z) \right) / h_{j+1,j}, \tag{3}$$

whose coefficients we store in an upper Hessenberg matrix

$$H_k := \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,k-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,k-1} \\ 0 & h_{2,1} & \ddots & h_{2,k-1} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{k,k-1} \end{bmatrix}. \tag{4}$$

Let $\text{nz}(A) = \{(i,j) : A_{ij} \text{ treated as nonzero}\}$ represent the edges in the directed graph of $A$, and let $A_{\mathcal{I}}$ indicate the submatrix of $A$ consisting of rows $i \in \mathcal{I}$. For simplicity, we ignore cancellation, i.e., we assume $\text{nz}(p_j(A)) \subseteq \text{nz}(p_{j+1}(A))$ and every entry of $x^{(j)}$ is treated as nonzero for all $j \geq 0$.

We construct a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the dependencies in computing $x^{(j)} := p_j(A)x$ for every $0 \leq j \leq k$. First, denoting row $i$ of $x^{(j)}$ by $x_i^{(j)}$, we define the $n(k+1)$ vertices $\mathcal{V} := \{x_i^{(j)} : 1 \leq i \leq n, 0 \leq j \leq k\}$. The edge set $\mathcal{E}$ consists of $k$ copies of $\text{nz}(A)$, between each adjacent pair of the $k+1$ *levels* $\mathcal{V}^{(j)} := \{x_i^{(j)} : 1 \leq i \leq n\}$, unioned with the edges due to the polynomial recurrence, i.e.,

$$\mathcal{E} := \left\{ \left( x_{i_1}^{(j+1)}, x_{i_2}^{(j)} \right) : \substack{0 \leq j < k, \\ (i_1, i_2) \in \text{nz}(A)} \right\} \cup \left\{ \left( x_i^{(j+d')}, x_i^{(j)} \right) : \substack{1 \leq d' \leq d, \\ 0 \leq j \leq k-d', \\ 1 \leq i \leq n} \right\} \tag{5}$$

where $H_k$ has $d$ nonzero superdiagonals (including main diagonal).

Now we partition $\mathcal{V}$ 'rowwise,' that is, each $x_i^{(j)}$ is assigned a processor *affinity* $m \in \{0, \ldots, p-1\}$, for $0 \leq j \leq k$. Let $\mathcal{V}_m$ and $\mathcal{V}_m^{(j)}$ restrict $\mathcal{V}$ and $\mathcal{V}^{(j)}$ to their elements with affinity $m$. Let $\mathcal{R}(\mathcal{S})$ denote the *reachability set* of $\mathcal{S} \subseteq \mathcal{V}$, i.e., the set $\mathcal{S}$ and vertices reachable from $\mathcal{S}$ via paths in $\mathcal{G}$; then, as with $\mathcal{V}$, we define the subsets $\mathcal{R}^{(j)}$, $\mathcal{R}_m$, and $\mathcal{R}_m^{(j)}$ of $\mathcal{R}$.

At the end of the computation, processor $m$ has computed/stored the entries $\mathcal{V}_m$. Thus, for PA0, processor $m$ must own $A_{\{i : x_i^{(j)} \in \mathcal{V}_m\}}$ and $\mathcal{V}_m^{(0)}$, and for PA1, processor $m$ must own $A_{\{i : x_i^{(1)} \in \mathcal{R}(\mathcal{V}_m)\}}$ and $\mathcal{R}^{(0)}(\mathcal{V}_m)$. We assume that the rows of $A$ are distributed to processors offline, while the source vector $x^{(0)}$ must be distributed at runtime (online).

With this notation, we present the parallel matrix powers algorithms PA0 (Algorithm 1) and PA1 (Algorithm 2), as pseudocode for processor $m$. The advantage of PA1 over PA0 is that it may send fewer messages between processors: whereas PA0 requires $k$ rounds of messages, PA1 requires only one. If the number of other processors with whom processor $m$ must communicate is within a constant factor for both algorithms, PA1 obtains a $\Theta(k)$-fold latency savings. In general, however, PA1 incurs greater bandwidth, arithmetic, and storage costs, as processors may perform redundant computations to avoid communication. Furthermore, in practice, PA1 requires additional data structures to encode the reachability sets; we assume these data structures are populated offline in a preprocessing phase.

We refer the reader to the complexity analysis in Tables 2.3 and 2.4, performance modeling in Sect. 2.6, and performance results in Sects. 2.10.3 and 2.11.3 of Mohiyuddin's thesis [10], which demonstrate that this optimization can lead to speedups in practice. For example, for a 9-point stencil on a $n^{1/2}$-by-$n^{1/2}$ mesh with $p$ processors, assuming $k \ll (n/p)^{1/2}$ and the monomial basis ($p_j(z) = z^j$), the number of arithmetic operations grows by a factor $1 + 2k(p/n)^{1/2}$, the number of messages decreases by a factor of $k$, and the number of words moved grows by a factor of $1 + k(p/n)^{1/2}$ [10]. Therefore, since the additional costs are lower order terms, we expect PA1 to give $\Theta(k)$ speedup when performance is latency-bound. Our performance modeling has confirmed this results [7].

**Algorithm 1.** PA0. Code for proc. $m$.

1: **for** $j = 1, \ldots, k$ **do**
2: **for** all procs. $\ell \neq m$ **do**
3: Send $x_i^{(j-1)} \in \mathcal{R}_m^{(j-1)}(\mathcal{V}_\ell^{(j)})$ to proc. $\ell$.
4: Recv. $x_i^{(j-1)} \in \mathcal{R}_\ell^{(j-1)}(\mathcal{V}_m^{(j)})$ from proc. $\ell$.
5: **end for**
6: Compute $x_i^{(j)} \in \mathcal{V}_m^{(j)}$ via (3).
7: **end for**

**Algorithm 2.** PA1. Code for proc. $m$.

1: **for** all procs. $\ell \neq m$ **do**
2: Send $x_i^{(0)} \in \mathcal{R}_m^{(0)}(\mathcal{V}_\ell^{(k)})$ to proc. $\ell$.
3: Recv. $x_i^{(0)} \in \mathcal{R}_\ell^{(0)}(\mathcal{V}_m^{(k)})$ from proc. $\ell$.
4: **end for**
5: **for** $j = 1, \ldots, k$ **do**
6: Compute $x_i^{(j)} \in \mathcal{R}^{(j)}(\mathcal{V}_m)$ via (3).
7: **end for**

## 2 Derivation of Parallel Blocking Covers

Recall that, given matrices $A \in \mathbb{C}^{n\times n}$ and $x \in \mathbb{C}^{n\times q}$, and $k \in \mathbb{N}$, our task is to compute (2). If $A$ is not well partitioned, PA0 must communicate at every step, but now the cost of PA1 may be much worse: when $k > 1$, every processor needs all rows of $A$ and $x^{(0)}$; there is no parallelism in computing all but the last SpMV. (Note when $k = 1$, PA1 degenerates to PA0.)

If, however, $A$ can be split in the form $D+USV^H$, where $D$ is well partitioned and $USV^H$ has low rank, we can use a generalization of the blocking covers approach [9] to recover parallelism. In this case, $D$ has a good cover and $US$ can be applied locally, but the application of $V^H$ incurs global communication. Thus, the application of $V^H$ will correspond to the blocker vertices in our algorithm, PA1-BC, which we now derive.

First, we recursively partition $H_k := \begin{bmatrix} H_{k-1} & h^{(k-1)} \\ 0_{1,k-1} & h_{k,k-1} \end{bmatrix}$ with $H_1 := [h_{0,0}, h_{1,0}]^T$, so $h^{(0)}, \ldots, h^{(k-1)}$ forms the upper triangle of $H_k$; substituting $z := A =: D + USV^H$, the recurrence for $x^{(j)} = p_j(A)x^{(0)}$ is

$$x^{(j+1)} = \left(Dx^{(j)} - [x^{(0)}, \ldots, x^{(j)}](h^{(j)} \otimes I_{q,q}) + USV^H x^{(j)}\right)/h_{j+1,j}. \quad (6)$$

We exploit the following identity, established by induction [7], to avoid performing $V^H \cdot x^{(j)}$ explicitly.

**Lemma 1.** *Given the additive splitting $z = z_1 + z_2$, (3) can be rewritten as*

$$p_j(z) = p_j(z_1) + \sum_{i=1}^{j} p_{i-1}^{j-i+1}(z_1) z_2 p_{j-i}(z)/h_{j-i+1,j-i} \quad (7)$$

*for $j \geq 0$, where $p_j^i(z)$ is a degree-$j$ polynomial related to $p_j(z)$ by reindexing the coefficients $h_{l,j} := h_{l+i,j+i}$ in (3).*

Now substitute $z := A = D + USV^H =: z_1 + z_2$ in (7), premultiply by $SV^H$, and postmultiply by $x^{(0)}$, to obtain

$$SV^H x^{(j)} = S\left(V^H p_j(D)x^{(0)} + \sum_{i=1}^{j} V^H p_{i-1}^{j-i+1}(D) U \frac{SV^H x^{(j-i)}}{h_{j-i+1,j-i}}\right). \quad (8)$$

Let $W_i := V^H p_i(D)U$ for $0 \le i \le k-2$, $y_i := V^H p_i(D)x$ for $0 \le i \le k-1$, and $b_j := SV^H x^{(j)}$ for $0 \le i \le k-1$. We can write $p_i^j$ in terms of $p_i = p_i^0$ via the following result, established by induction [7].

**Lemma 2.** *There exist coefficient vectors $w_i^j \in \mathbb{C}^{i+1}$ satisfying*

$$[W_0, \ldots, W_i](w_i^j \otimes I_{r,r}) = V^H p_i^j(D)U \tag{9}$$

*for $0 \le i \le k-2, 1 \le j \le k-i-1$, that can be computed by $w_0^j := 1$ and*

$$w_{l+1}^j := \left(H_{l+1} w_l^j - \left[\begin{bmatrix} w_0^j \\ 0_{l,1} \end{bmatrix}, \begin{bmatrix} w_1^j \\ 0_{l-1,1} \end{bmatrix}, \ldots, \begin{bmatrix} w_l^j \\ 0 \end{bmatrix}\right] h^{(j)}_{\{j,\ldots,j+l\}}\right) / h_{j+l+1,j+l}. \tag{10}$$

Using this result, we write (8) as

$$b_j = S\left(y_j + [W_0, \ldots, W_{j-1}] \cdot \sum\nolimits_{i=1}^{j} \left(\begin{bmatrix} w_{i-1}^{j-i+1} \\ 0_{j-i,1} \end{bmatrix} \otimes I_{r,r}\right) \frac{b_{j-i}}{h_{j-i+1,j-i}}\right); \tag{11}$$

however, in case $H_k$ is Toeplitz, the summation simplifies to $\left[\frac{b_{j-1}^T}{h_{j,j-1}}, \ldots, \frac{b_0^T}{h_{1,0}}\right]^T$, so we need not compute $\{w_i^j\}$.

Ultimately we must evaluate (6), substituting $b_j$ for $SV^H x^{(j)}$. This can be accomplished by applying PA1 to the following recurrence for $p_j(z, c)$, where $c := \{c_0, \ldots, c_{j-1}, \ldots\} := \{Ub_0, \ldots, Ub_{j-1}, \ldots\}$:

$$p_0(z, c) := 1, \quad p_{j+1}(z, c) := \left(z p_j(z) - \sum\nolimits_{i=0}^{j} h_{i,j} p_i(z) + c_j\right) / h_{j+1,j}. \tag{12}$$

Given the notation established, we construct PA1-BC (Algorithm 3). In terms of the graph of $D$, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, processor $m$ must own

$$D_{\{i:x_i^{(1)} \in \mathcal{R}(\mathcal{V}_m)\}},\ U_{\{i:x_i^{(1)} \in \mathcal{R}(\mathcal{V}_m)\}},\ V_{\{i:x_i^{(j)} \in \mathcal{V}_m\}},\ \text{and } \mathcal{R}^{(0)}(\mathcal{V}_m), \tag{13}$$

in order to compute the entries $x_i^{(j)} \in \mathcal{V}_m$. In exact arithmetic, PA1-BC returns the same output as PA0 and PA1. However, by exploiting the splitting $A = D + USV^H$, PA1-BC may avoid communication when $A$ is not well partitioned. Communication occurs in calls to PA1 (Lines 1 and 4), as well as in Allreduce collectives (Lines 2 and 5). As computations in Lines 1, 2, and 3 do not depend on the input $x^{(0)}$, they need only be computed once per matrix $A = D + USV^H$, thus we assume their cost is incurred offline.

For the familiar reader, the sequential blocking covers algorithm [9] is a special case of a sequential execution of Algorithm 3, using the monomial basis, where $U = [e_i : i \in \mathcal{I}]$ and $SV^H = A_{\mathcal{I}}$, where $e_i$ is the $i$-th column of the identity and $\mathcal{I} \subseteq \{1, \ldots, n\}$ are the indices of the blocker vertices. In Algorithm 3, Lines $\{1, 2, 3\}$, $\{4, 5\}$, 6, and 7 correspond to the 4 phases of the sequential blocking covers algorithm, respectively [9]. In the next section, we demonstrate the benefit of our approach on a motivating example, matrix powers with HSS matrix $A$.

**Algorithm 3.** PA1-BC. Code for proc. $m$.

1: Compute local rows of $K_{k-1}(D, U, H_{k-1})$ with PA1, premultiply by local columns of $V^H$.
2: Compute $[W_0, \ldots, W_{k-2}]$ by an Allreduce.
3: Compute $w_i^j$ for $0 \le i \le k-2$ and $1 \le j \le k-i-1$, via (10).
4: Compute local rows of $K_k(D, x^{(0)}, H_k)$ with PA1, premultiply by local columns of $V^H$.
5: Compute $[y_0, \ldots, y_{k-1}]$ by an Allreduce.
6: Compute $[b_0, \ldots, b_{k-1}]$ by (11).
7: Compute local rows of $[x^{(0)}, \ldots, x^{(k)}]$ with PA1, modified for (12).

## 3 Hierarchical Semiseparable Matrix Example

Hierarchical ($\mathcal{H}$-) matrices are amenable to the splitting $A = D + UV^H$, where $D$ is block diagonal and $UV^H$ represents the off-diagonal blocks. Naturally, $U$ and $V$ are quite sparse and it is important to exploit this sparsity in practice. In the special case of HSS matrices, many columns of $U$ and $V$ are linearly dependent, and we can exploit the matrix $S$ in the splitting $USV^H$ to write $U$ and $V$ as block diagonal matrices. We review the HSS notation and the algorithm for computing $v = Ax$ given by Chandrasekaran et al. [3, Sects. 2 and 3]. For any $0 \le L \le \lfloor \lg n \rfloor$, where $\lg = \log_2$, we can write $A$ hierarchically as a perfect binary tree of depth $L$ by recursively defining its diagonal blocks as $A =: D_{0;1}$ and

$$D_{\ell-1;i} =: \begin{bmatrix} D_{\ell;2i-1} & U_{\ell;2i-1}B_{\ell;2i-1,2i}V_{\ell;2i}^H \\ U_{\ell;2i}B_{\ell;2i,2i-1}V_{\ell;2i-1}^H & D_{\ell;2i} \end{bmatrix} \tag{14}$$

for $1 \le \ell \le L$, $1 \le i \le 2^{\ell-1}$, where $U_{0;1}, V_{0;1} := []$, and for $\ell \ge 2$,

$$U_{\ell-1;i} =: \begin{bmatrix} U_{\ell;2i-1}R_{\ell;2i-1} \\ U_{\ell;2i}R_{\ell;2i} \end{bmatrix}, \qquad V_{\ell-1;i} =: \begin{bmatrix} V_{\ell;2i-1}W_{\ell;2i-1} \\ V_{\ell;2i}W_{\ell;2i} \end{bmatrix}; \tag{15}$$

the subscript expression $\ell; i$ denotes vertex $i$ of the $2^\ell$ vertices at level $\ell$.

The action of $A$ on a matrix $x$, i.e., $v := Ax$, satisfies $v_{0;1} = D_{0;1}x_{0;1}$, and for $1 \le \ell \le L$, $1 \le i \le 2^\ell$, satisfies $v_{\ell;i} = D_{\ell;i}x_{\ell;i} + U_{\ell;i}f_{\ell;i}$, with $f_{1;1} = B_{1;1;2}g_{1;2}$, $f_{1;2} = B_{1;2;1}g_{1;1}$, and, for $1 \le \ell \le L-1$, $1 \le i \le 2^\ell$,

$$f_{\ell+1;2i-1} = \begin{bmatrix} R_{\ell+1;2i-1}^T \\ B_{\ell+1;2i-1,2i}^T \end{bmatrix}^T \begin{bmatrix} f_{\ell;i} \\ g_{\ell+1;2i} \end{bmatrix}, \; f_{\ell+1;2i} = \begin{bmatrix} R_{\ell+1;2i}^T \\ B_{\ell+1;2i,2i-1}^T \end{bmatrix}^T \begin{bmatrix} f_{\ell;i} \\ g_{\ell+1;2i-1} \end{bmatrix}, \tag{16}$$

where, for $1 \le \ell \le L-1$, $1 \le i \le 2^\ell$, $g_{\ell;i} = \begin{bmatrix} W_{\ell+1;2i-1} \\ W_{\ell+1;2i} \end{bmatrix}^H \begin{bmatrix} g_{\ell+1;2i-1} \\ g_{\ell+1;2i} \end{bmatrix}$, and $g_{L;i} = V_{L;i}^H x_{L;i}$ for $1 \le i \le 2^L$. For any HSS level $\ell$, we assemble the block diagonal matrices

$$U_\ell := \bigoplus_{i=1}^{2^\ell} U_{\ell;i}, \qquad V := \bigoplus_{i=1}^{2^\ell} V_{\ell;i}, \qquad D_\ell := \bigoplus_{i=1}^{2^\ell} D_{\ell;i}, \tag{17}$$

denoted here as direct sums of their diagonal blocks. We also define matrices $S_\ell$, representing the recurrences for $f_{\ell;i}$ and $g_{\ell;i}$, satisfying $v = Ax =: D_\ell x + U_\ell S_\ell V_\ell^H x$. We now discuss parallelizing the computation $v = Ax$, to generalize PA0 and PA1 to HSS matrices.

### 3.1 PA0 for HSS Matrices

We first discuss how to modify PA0 for HSS $A$, exploiting the $v = Ax$ recurrences for each $1 \le j \le k$; we call the resulting algorithm PA0-HSS. For brevity, we defer a detailed description [7]. PA0-HSS can be seen as an HSS specialization of known approaches for distributed-memory $\mathcal{H}$-matrix-vector multiplication [8].

We assume the HSS representation of $A$ has perfect binary tree structure to some level $L > 2$, and there are $p \ge 4$ processors with $p$ a power of 2. For each processor $m \in \{0, 1, \ldots, p-1\}$, let $L_m$ denote the smallest level $\ell \ge 1$ such that $p/2^\ell$ divides $m$. We also define the intermediate level $1 < L_p := \lg(p) \le L$ of the HSS tree; each $L_m \ge L_p$, and equality is attained when $m$ is odd.

First, on the upsweep, each processor locally computes $V_{L_p}^H x$ (its subtree, rooted at level $L_p = \lg(p)$) and then performs $L_p$ steps of parallel reduction, until there are two processors active, and then a downsweep until level $L_p$, at which point each processor is active, owns $S_{L_p} V_{L_p}^H x$, and recurses into its local subtree to finally compute its rows of $v = D_L x + U_L S_L V_L x$. More precisely, we assign processor $m$ the computations $f_{\ell;i}$ and $g_{\ell;i}$ for

$$\left\{ \ell, i : \substack{L \ge \ell \ge L_p \\ 2^\ell m/p+1 \le i \le 2^\ell (m+1)/p} \right\} \text{ and for } \left\{ \ell, i : \substack{L_p - 1 \ge \ell \ge L_m \\ i = 2^\ell m/p+1} \right\} \tag{18}$$

and $D_L$, $U_L$, and $V_L$ are distributed contiguously block rowwise, so processor $m$ stores blocks $D_{L_p;m+1}$, $U_{L_p;m+1}$, and $V_{L_p;m+1}$. The $R_{\ell;i}$, $W_{\ell;i}$, and $B_{\ell;i}$ matrices are distributed so that they are available for the computations in the upsweep/downsweep; memory requirements are listed in Table 1.

### 3.2 PA1 for HSS Matrices

The block-diagonal structure of $D_\ell$, $U_\ell$, and $V_\ell$ in (17) suggests an efficient parallel implementation of PA1-BC, which we present as PA1-HSS (Algorithm 4). However, now each processor must perform the *entire* upsweep/downsweep between levels 1 and $L_p$ locally. The additional cost shows up in our complexity analysis (see Table 1) as a factor of $p$, compared to a factor of $\lg(p)$ in PA0-HSS; we also illustrate this tradeoff in Sect. 4.

We assume the same data layout as PA0-HSS: each processor owns a diagonal block of $D_{L_p}$, $U_{L_p}$, and $V_{L_p}$, but only stores the smaller blocks of level $L$. We assume each processor is able to apply $S_{L_p}$. We rewrite (6) for the local rows, and exploit the block diagonal structure of $D_{L_p}$ and $U_{L_p}$, to write

$$\begin{aligned} x_{L_p;m+1}^{(j+1)} = \Big( D_{L_p;m+1} x_{L_p;m+1}^{(j)} &- [x_{L_p;m+1}^{(0)}, \ldots, x_{L_p;m+1}^{(j)}](h^{(j)} \otimes I_{q,q}) \\ &+ U_{L_p;m+1} (b_j)_{\{mr+1,\ldots,(m+1)r\}} \Big) / h_{j+1,j}. \end{aligned} \tag{19}$$

Each processor locally computes all rows of $b_j = S_{L_p} V_{L_p}^H x^{(j)} = S_{L_p} \cdot z$, where $z$ is the maximal parenthesized term in (11), using the HSS recurrences:

$$V_{L_p}^H x^{(j)} = z =: \left[g_{L_p,1}^T \cdots g_{L_p,p}^T\right]^T \mapsto \left[f_{L_p,1}^T \cdots f_{L_p,p}^T\right]^T := b_j = S_{L_p} V_{L_p}^H x^{(j)}. \quad (20)$$

The rest of PA1-HSS is similar to PA1-BC, except Allgather operations replace Allreduce operations in Lines 1 and 5 to exploit block structure of $V^H$.

**Algorithm 4.** PA1-HSS (Blocking Covers). Code for proc. $m$.

1: Compute $K_{k-1}(D_{L_p;m+1}, U_{L_p;m+1}, H_{k-1})$, premultiply by $V_{L_p;m+1}^H$.
2: Compute $[W_0, \ldots, W_{k-2}]$ by an Allgather.
3: Compute $w_i^j$ for $0 \le i \le k-2$, and $1 \le j \le k-i-1$, via (10).
4: Compute $K_k(D_{L_p;m+1}, x_{L_p;m+1}^{(0)}, H_k)$, premultiply by $V_{L_p;m+1}^H$.
5: Compute $[y_0, \ldots, y_{k-1}]$ by an Allgather.
6: Compute $[b_0, \ldots, b_{k-1}]$ by (11), where $S = S_{L_p}$ is applied by (20).
7: Compute local rows of $[x^{(0)}, \ldots, x^{(k)}]$ according to (19).

### 3.3 Complexity Analysis

We gave a detailed complexity analysis of PA0-HSS and PA1-HSS in [7]; we summarize the asymptotics (i.e., ignoring constant factors) in Table 1. We assume $A$ is given in HSS form, as described above, where all block matrices are dense. For simplicity, we assume $n$ and *HSS-rank* $r$ are powers of 2 and leaf level $L = \lg(n/r)$. Note that one could use faster Allgather algorithms (e.g., [2]) for PA1-HSS to eliminate the factor of $\lg p$ in the number of words moved.

## 4 Performance Model

We model speedups of PA1-HSS over PA0-HSS on two machine models used by Mohiyuddin [10] – 'Peta,' an 8100 processor petascale machine, and 'Grid,' 125 terascale machines connected via the Internet. Peta has a flop rate $\gamma = 2 \cdot 10^{-11}$ s/flop, latency $\alpha = 10^{-5}$ s/message, and bandwidth $\beta = 2 \cdot 10^{-9}$ s/word, and Grid has flop rate $\gamma = 10^{-12}$ s/flop, latency $\alpha = 10^{-1}$ s/message, and bandwidth $\beta = 25 \cdot 10^{-9}$ s/word. Complexity counts used can be found in [7].

Speedups of PA1-HSS over $k$ invocations of PA0-HSS, for both Peta and Grid, are shown in Fig. 1. We used parameters from the parallel HSS performance tests of Wang et al. [13], where $p = (4, 16, 64, 256, 1024, 4096)$, $n = (2.5, 5, 10, 20, 40, 80) \cdot 10^3$, $r = (5, 5, 5, 5, 6, 7)$. Note that for Grid we only use the first 3 triples $(p_i, n_i, r_i)$ since $p_{\max} = 125$. We assume a three-term recurrence ($H_k$ is tridiagonal), as these suffice in practice to obtain well-conditioned polynomial bases, even for large $k$ [12].

On Peta, we see $O(k)$ speedups for smaller $p$ and $k$, but as these quantities increase, the expected speedup drops. This is due to the extra multiplicative factor of $p$ in the bandwidth cost and the extra additive factor of $k^3qp$ in the

**Table 1.** Asymptotic complexity of PA0-HSS and PA1-HSS, ignoring constant factors. 'Offline' refers to Lines 1–3 and 'Online' refers to Lines 4–7 of PA1-HSS.

| Algorithm | | Flops | Words moved | Messages | Memory |
|---|---|---|---|---|---|
| PA0-HSS | | $kqrn/p + kqr^2 \lg p$ | $kqr \lg p$ | $k \lg p$ | $(kq+r)n/p + r^2 \lg p$ |
| PA1-HSS | (offline) | $kr^2n/p + k^3$ | $kr^2p \lg p$ | $\lg p$ | $(kq+r)n/p + k(q+r)rp$ |
| | (online) | $kqrn/p + k(k+r)^2qp$ | $kqrp \lg p$ | $\lg p$ | |

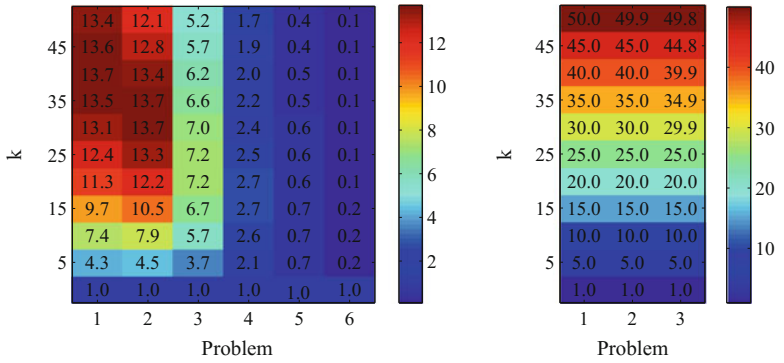


**Fig. 1.** Predicted PA1-HSS speedups on Peta (left) and Grid (right). Note that $p$ and $n$ increase with problem number on x-axis.

flop cost of PA1-HSS. Since the relative latency cost is lower on Peta, the effect of the extra terms becomes apparent for large $k$ and $p$. On Grid, PA0-HSS is extremely latency bound, so a $\Theta(k)$-fold reduction in latency results in a $\Theta(k)\times$ faster algorithm. This is the best we can expect. Note that many details are abstracted in these models, which are meant to give a rough idea of asymptotic behavior. Realizing such speedups in practice remains future work.

## 5 Future Work and Conclusions

In this work, we derive a new parallel communication-avoiding matrix powers algorithm for $A = D + USV^H$, where $D$ is well partitioned and $USV^H$ has low rank but $A$ may not be well partitioned. This allows speedups for a larger class of problems than previous algorithms [4,10], which require well-partitioned $A$. Our approach exploits low-rank properties of dense blocks, asymptotically reducing parallel latency cost. We demonstrate the generality of our parallel blocking covers technique by applying it to matrices with hierarchical structure. Performance models predict up to 13× speedups on petascale machines and up to $3k$ speedups on extremely latency-bound machines, despite tradeoffs in arithmetic and bandwidth cost. Future work includes a high-performance parallel implementation of our algorithm to verify predicted speedups, as well as integration into preconditioned communication-avoiding Krylov solvers.

**Acknowledgments.** We acknowledge support from the US DOE (grants DE-SC000 4938, DE-SC0005136, DE-SC0010200, DE-SC0008700 and AC02-05CH11231) and DARPA (grant HR0011-12-2-0016), as well as contributions from Intel, Oracle, and MathWorks.

## References

1. Bebendorf, M.: A means to efficiently solve elliptic boundary value problems. In: Bart, T., Griebel, M., Keyes, D., Nieminen, R., Roose, D., Schlick, T. (eds.) Hierarchical Matrices. LNCS, vol. 63, pp. 49–98. Springer, Heidelberg (2008)
2. Chan, E., Heimlich, M., Purkayastha, A., Van De Geijn, R.: Collective communication: theory, practice, and experience. Concurrency Comput.: Pract. Exper. **19**, 1749–1783 (2007)
3. Chandrasekaran, S., Dewilde, P., Gu, M., Lyons, W., Pals, T.: A fast solver for HSS representations via sparse matrices. SIAM J. Matrix Anal. Appl. **29**, 67–81 (2006)
4. Demmel, J., Hoemmen, M., Mohiyuddin, M., Yelick, K.: Avoiding communication in computing Krylov subspaces. Technical report UCB/EECS-2007-123, University of California-Berkeley (2007)
5. Hoemmen, M.: Communication-avoiding Krylov subspace methods. Ph.D. thesis, University of California-Berkeley (2010)
6. Hong, J., Kung, H.: I/O complexity: the red-blue pebble game. In: Proceedings of the 13th ACM Symposium on Theory of Computing, pp. 326–333. ACM, New York (1981)
7. Knight, N., Carson, E., Demmel, J.: Exploiting data sparsity in parallel matrix powers computations. Technical report UCB/EECS-2013-47, University of California-Berkeley (2013)
8. Kriemann, R.: Parallele Algorithmen für $\mathcal{H}$-Matrizen. Ph.D. thesis, Christian-Albrechts-Universität zu Kiel (2005)
9. Leiserson, C., Rao, S., Toledo, S.: Efficient out-of-core algorithms for linear relaxation using blocking covers. J. Comput. Syst. Sci. Int. **54**, 332–344 (1997)
10. Mohiyuddin, M.: Tuning hardware and software for multiprocessors. Ph.D. thesis, University of California-Berkeley (2012)
11. Mohiyuddin, M., Hoemmen, M., Demmel, J., Yelick, K.: Minimizing communication in sparse matrix solvers. In: Proceedings of the Conference on High Performance Computing Networking, Storage, and Analysis, pp. 36:1–36:12. ACM, New York (2009)
12. Philippe, B., Reichel, L.: On the generation of Krylov subspace bases. Appl. Numer. Math. **62**, 1171–1186 (2012)
13. Wang, S., Li, X., Xia, J., Situ, Y., de Hoop, M.: Efficient scalable algorithms for hierarchically semiseparable matrices. SIAM J. Sci. Comput. (2012, under review)