Optimizing a Super-fast Eigensolver for Hierarchically Semiseparable Matrices

Abhishek Josyula   Pritesh Verma   Amar Gaonkar

Amlan Barua   Nikhil Hegde

Indian Institute of Technology Dharwad, Karnataka, India

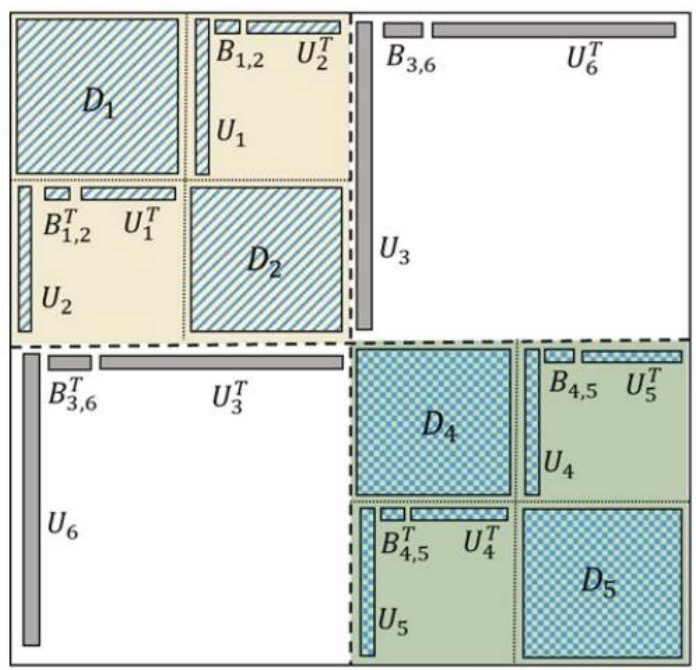
॥ सा विद्या या विमुक्तये ॥

भारतीय प्रौद्योगिकी संस्थान धारवाड

Indian Institute of Technology Dharwad

Hierarchically Semiseparable Matrices

- Off diagonal blocks have relatively small ranks w.r.t. size of the matrix. So, they can be represented product of low-ranked matrices
- the off-diagonal blocks have hierarchical bases



$U_6 = \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix}$, D_3, D_6

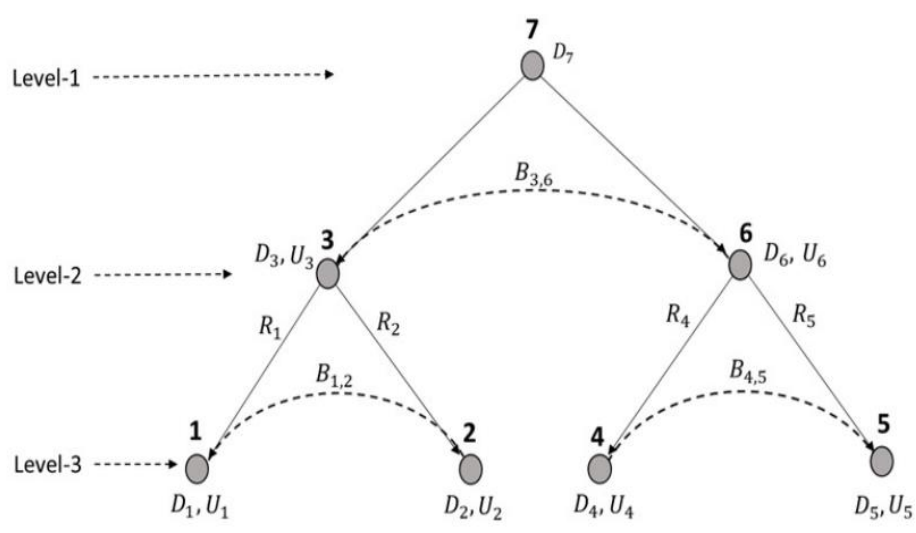
Hierarchically Semiseparable Matrices

- Off diagonal blocks have relatively small ranks w.r.t. size of the matrix. So, they can be represented product of low-ranked matrices
- the off-diagonal blocks have hierarchical bases

- **HSS** matrices appear naturally in different applications, e.g. in structural mechanics or fluid dynamics, through the spatial discretization of partial differential equations and in integral equations, ct scans.
- Standard eigen solvers in libraries like LAPACK are of $O(N^3)$ time complexity and fail to take advantage of the inherent structure of matrix.

SuperDC for Symmetric HSS Matrices

- SuperDC is a divide-conquer algorithm to compute the eigenvalues and eigenvectors of symmetric HSS Matrices
- Compute eigenvalues in $O(r^2N\log(N)) + O(rN\log^2 N)$, where r is rank of block matrix.
    - The 'Super' terminology is due to the sub-N^2 computational complexity.



Challenges and Opportunities

- The HSS matrices are often huge, of the order of few millions to few hundred millions.

- Previous implementation of `SuperDC`, `HSSEigen`, is a sequential MATLAB implementation.
  - `HSSEigen` uses a dense-matrix storage representation for all inputs. This is wasteful for some matrices like tridiagonal, banded etc.

Efficient parallel and distributed implementations are necessary.

<u>Our Focus:</u> Optimizing and Parallelizing SuperDC

Contributions

- Shared- and distributed-memory parallel algorithms for computing eigenvalues and eigenvectors of Symmetric HSS Matrices. Also, present a span and available parallelism analysis.

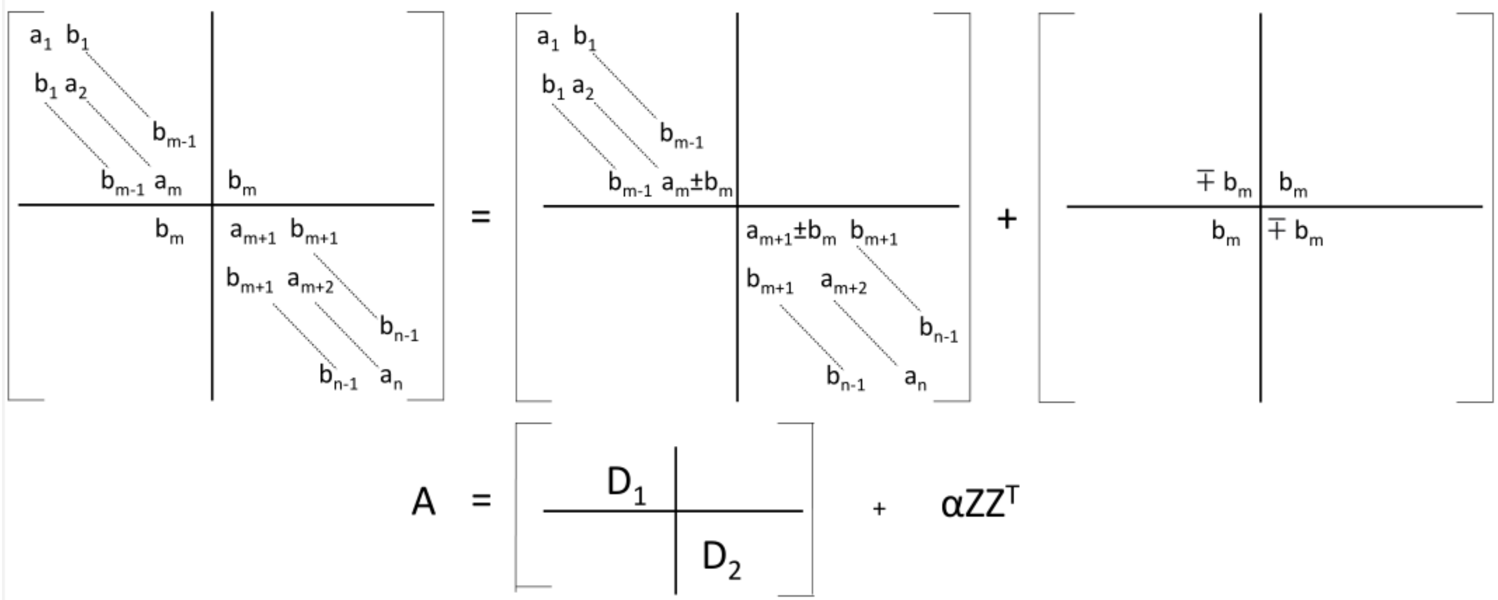
- Optimize for storage and present an efficient problem decomposition for distributed-memory parallel algorithm.

- Implement using multiple programming paradigms (OpenMP, OpenCilk, MPI) and evaluate with different scheduling policies, sparsity structures of input matrices, and program configurations.

Cuppen's DC Algorithm

Goal: Compute $A = Q\Lambda Q^T, \Lambda = \mathrm{diag}(\lambda_i), QQ^T = I$

1. Recursive decomposition



2. Solve for $D1 = Q_1\Lambda_1 Q_1^T$ and $D2 = Q_2\Lambda_2 Q_2^T$

3. $$A = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} + \alpha v v^T \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}$$

where $v = \mathrm{diag}(\Lambda_1, \Lambda_2) Z$

Cuppen's DC Algorithm

$$A = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} + \alpha v v^T \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}$$

4. Eigenvalues of the matrix $\tilde{\Lambda} = \mathrm{diag}(\Lambda_1, \Lambda_2) + \alpha v v^T$ are the roots of the secular equation:

$$1 - \alpha \sum_{i=1}^{n} \frac{v_i^2}{\lambda_i - \tilde{\lambda}} = 0$$

where v_i are elements of vector v, λ_i belong to either Λ_1 or Λ_2

5. Eigenvectors of $\tilde{\Lambda}$ obtained using $q_i = (\mathrm{diag}(\Lambda_1, \Lambda_2) - \lambda_i I)^{-1} v$

6. Eigenvectors of A are $\begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} Q$

HSS Matrix Definition

Notation:

- I_i is index set of a tree node numbered i. A m-level, complete binary tree is considered and nodes are numbered in post-order.
- Each node of an m-level tree represents a contiguous index set $I_i \subseteq \{1,2,..2^m - 1\}$. E.g. for root node, $I_{2^m-1} = \{1,2,..2^m - 1\}$
- for any non-leaf node i: $I_l \cap I_r = \phi$ and $I_l \cup I_r = I_i$ and $I_r, I_i \neq \phi$, l and r denote the left and right children resp.
- $A_{I\times J}$ indicates submatrix of A obtained from index sets I, J

A matrix is in symmetric HSS form if there is a mapping of nodes $\{1,2,..2^m - 1\}$ *to matrices* D, U, R, B *– called generators as follows:*

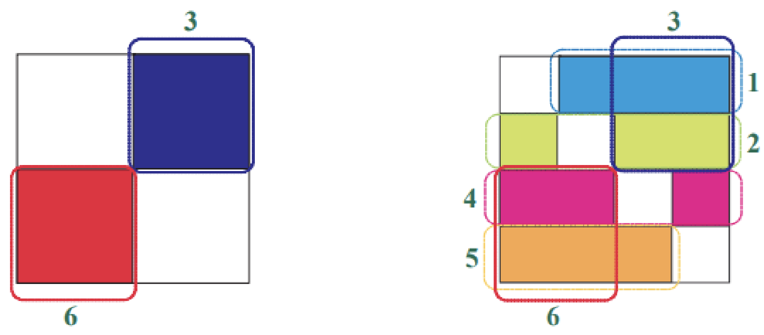
HSS Matrix Definition

- $$A_{I_i \times I_i} = D_{I_i} = \begin{bmatrix} D_{I_l} & U_{I_l} B_{I_l,I_r} U_{I_r}^T \\ U_{I_r} B_{I_l,I_r}^T U_{I_l}^T & D_{I_r} \end{bmatrix}$$

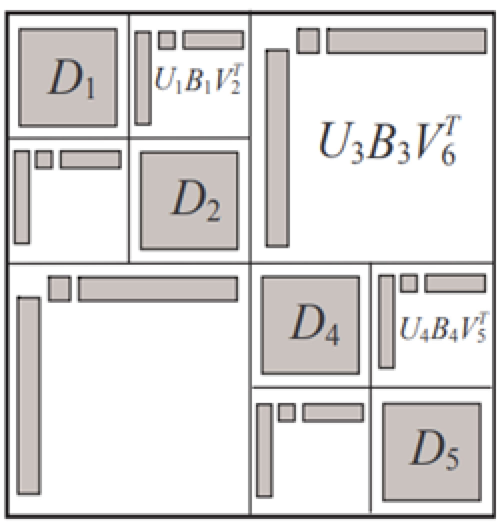
$$U_{I_i} = \begin{bmatrix} U_{I_l} & \\ & U_{I_r} \end{bmatrix} \begin{bmatrix} R_{I_l} \\ R_{I_r} \end{bmatrix}$$

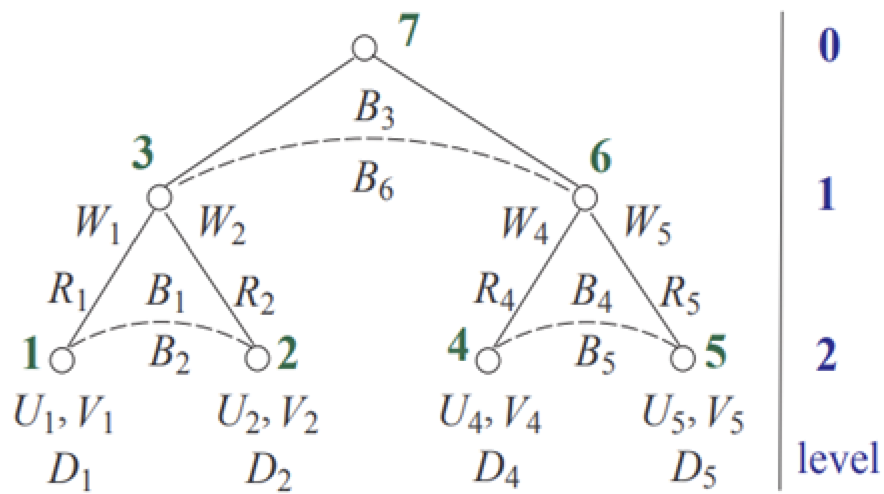
- Note that for all non-leaf descendants of root node i.e. nodes numbered j ∈ $\{2^{m-1}$ to $2^m - 2\}$, R_{I_j} is zero matrix and not needed.
- U_{I_l} and U_{I_r} can be combined to form the basis matrix for a larger matrix U_{I_i} (node i is the parent of nodes l and r here)

HSS Matrix Definition - Visualization



(i) One level of HSS blocks. (ii) Two levels of HSS blocks.





Note: for symmetric matrices, V and W matrices are not needed. Also, B3 = B6, B1=B2, and B4=B5 for symmetric matrices.

SuperDC for Symmetric HSS matrices

Goal: Compute $D_i = Q\Lambda Q^T, \Lambda = \mathrm{diag}(\lambda_i)$, note: D_i is in HSS form

1. Recursive decomposition - cast D_i as sum of a diagonal matrix and a rank-r update:

$$D_i = \mathrm{diag}\left(\tilde{D}_l, \tilde{D}_r\right) + Z_i Z_i^T, \text{ where } \widetilde{D_l} = D_l - U_l B_l B_l^T U_l^T$$

$$\widetilde{D_r} = D_r - U_l U_l^T$$

$$Z_i = \begin{pmatrix} U_l B_l \\ U_r \end{pmatrix}$$

$\widetilde{D_l}$ and $\widetilde{D_r}$ must be in HSS form and the rank of $Z_i Z_i^T$ remains at most r.

2. Solve for $\widetilde{D_l} = Q_l \widetilde{\Lambda_l} \quad Q_l^T$ and $\widetilde{D_r} = Q_r \widetilde{\Lambda_r} \quad Q_r^T$

3. $D_i = \mathrm{diag}\left(Q_l, Q_r\right)\left[\mathrm{diag}\left(\tilde{\Lambda}_l, \tilde{\Lambda}_r\right) + \tilde{Z}_i \tilde{Z}_i^T\right]\mathrm{diag}\left(Q_l^T, Q_r^T\right)$

where: $\tilde{Z}_i = \mathrm{diag}\left(Q_l^T, Q_r^T\right) Z_i$

SuperDC for Symmetric HSS matrices

4. Eigendecomposition of the matrix $\left[\text{diag}\left(\tilde{\Lambda}_l, \tilde{\Lambda}_r\right) + \tilde{Z}_i \tilde{Z}_i^T\right]$ needs to be computed.

$$
\begin{aligned}
\text{diag}\left(\tilde{\Lambda}_l, \tilde{\Lambda}_r\right) + v^{(1)}\left(v^{(1)}\right)^T &= Q^{(1)}\tilde{\Lambda}^{(1)}\left(Q^{(1)}\right)^T \\
\tilde{\Lambda}^{(1)} + v^{(2)}\left(v^{(2)}\right)^T &= Q^{(2)}\tilde{\Lambda}^{(2)}\left(Q^{(2)}\right)^T \\
&\vdots \\
\tilde{\Lambda}^{(k-1)} + v^{(k)}\left(v^{(k)}\right)^T &= Q^{(k)}\tilde{\Lambda}^{(k)}\left(Q^{(k)}\right)^T
\end{aligned}
$$

With k eigenvalue problems need to be solved (rank-k updates)

Let $\tilde{Z}_i = \left(\tilde{z}^1, \tilde{z}^2, \ldots, \tilde{z}^k\right)$, where $\tilde{z}^j$ are columns of the matrix

$$v^{(i)} = (Q_i^{(i-1)})^T \tilde{z}^{(i)}$$

$$Q^{(0)} = diag\left(\tilde{Q}_l, \tilde{Q}_r\right)$$

SuperDC for Symmetric HSS matrices

5. Eigendecomposition of the matrix

$$D_i = \left(Q_i^{(0)} Q_i\right) diag\left(\lambda_j^{(r)}\Big|_{j=1}^{n}\right) \left(Q_i^{(0)} Q_i\right)^T,$$

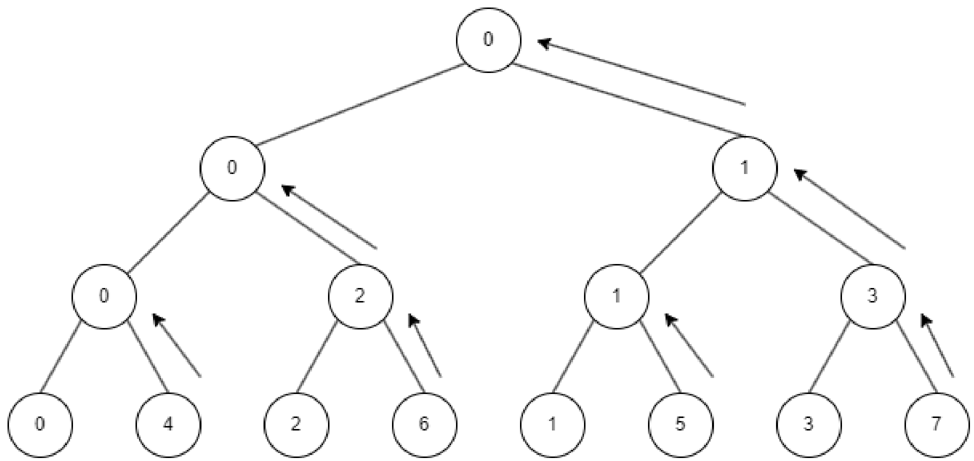
$$where\ Q_i = Q_i^{(1)} \dots Q_i^{(r)}\ \ and\ \tilde{\Lambda}^{(i)} = diag\left(\lambda_j^{(i)}\Big|_{j=1}^{n}\right)$$

$\mathrm{diag}\left(\lambda_j^{(r)}\Big|_{j=1}^{n}\right)$ are the eigenvalues of D_i

$\left(Q_i^{(0)} Q_i\right)$ is the eigenmatrix of D_i

Parallel SuperDC

- Focus: parallelize the conquer stage only
- Map the tree nodes to processes as per the following:



- Precludes block-cyclic distribution of matrix blocks
- Necessary to minimize communication and avoid fragmentation of generators.
- Results in O(p) communication, p = number of processes.

Shared-memory parallel implementations

- Create two OMP tasks / Cilk threads repeatedly for every level of recursive decomposition.
  - OMP Tasks are mapped to worker threads. Untied tasks allow for resumption of a task by any idle thread.
  - OpenCilk uses work-stealing scheduler
- Stop creating new tasks / Cilk threads based on program input

```
if node is leaf then
    computeLeafEig()
else if node is non-leaf then
    left,right = hsstree->GetChildren(node)
    cilk_spawn cilkSuperDC(left, ++level)
    cilk_spawn cilkSuperDC(right, ++level)
    cilk_sync
    QtMulZ()
    r_RankOneUpdate()
end if
```

Shared-memory parallel implementations

- Available parallelism analysis

$$\mathcal{T}_\infty(n) = 2 * \mathcal{T}_\infty(n/2) + \mathcal{O}(\texttt{QtMulZ}) + \mathcal{O}(\texttt{r\_RankOneUpdate})$$

$$\Rightarrow \mathcal{T}_\infty(n) = 2 * \mathcal{T}_\infty(n/2) + \mathcal{O}(rn\log n) + r * \mathcal{O}\left(r^2 n \log n\right)$$

$$\Rightarrow \mathcal{T}_\infty(n) = \mathcal{O}\left(r^3 n \log n\right)$$

$\mathcal{T}_1(n) \,/\, \mathcal{T}_\infty(n) = \mathcal{O}(\frac{\log n}{r})$ known that: $\mathcal{T}_1(n)$, is $\mathcal{O}\left(r^2 n \log^2 n\right)$

- Bulk-synchronous / level-wise synchronization not suitable
  - When the eigenvalue computation at *all* nodes at a level are complete, proceed to the next lower level (i.e. up the tree).
  - Stragglers take long time to execute

Experimental Setup

- Single node experiments:
  - 36-core dual-socket, Intel Xeon Gold 6240C@2.60GHz processor
  - CPU has 64 KB shared data and instruction caches, 1 MB unified L2 and 36 MB L3 unified caches
  - 128GB DDR4 memory
  - Ubuntu 20.04, Clang 14.0.6 for OpenCilk, GCC 12.0.0, LAPACK 3.9, Matlab 2020
- Multi node experiments:
  - Each node has: Xeon 8268, 2.9GHz processor, 48 cores, 192GB RAM.
- Data Sets
  - Tridiagonal, Banded, and Discretized kernel matrix

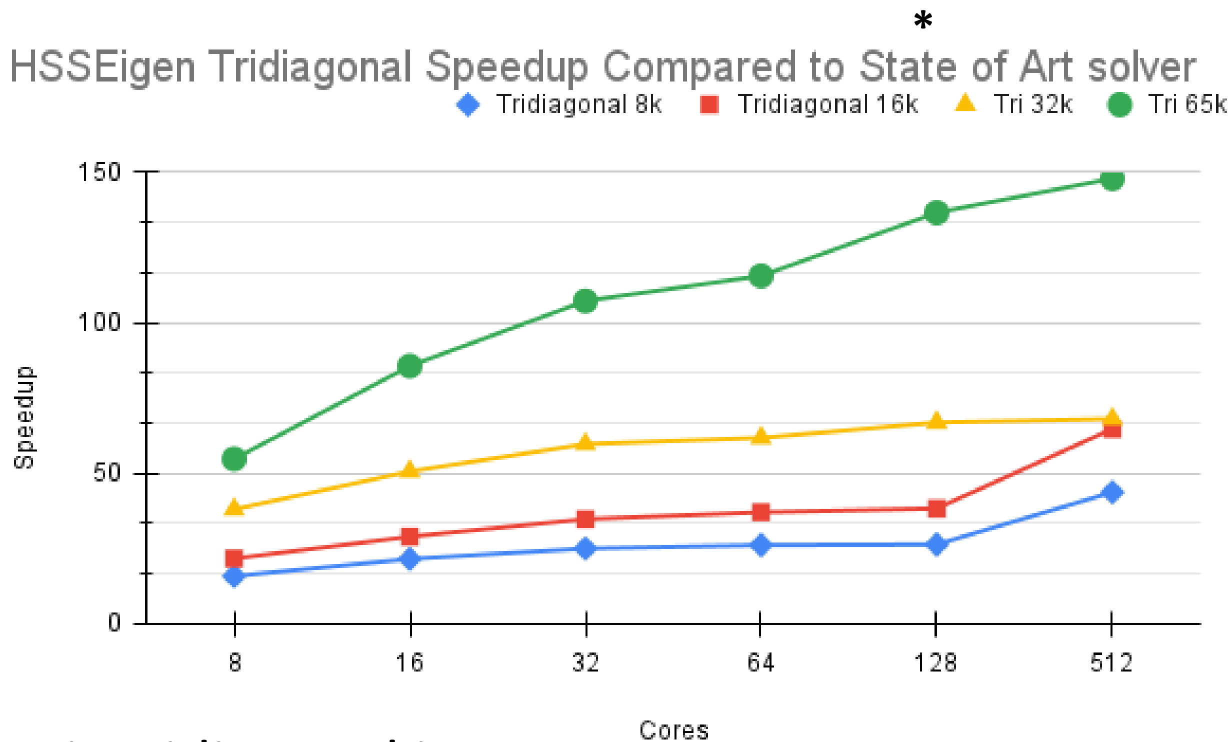
Results - Summary

Input	Implementations *			
	eig_lapack	hsseigen	hssedc_dist	speedup
Tri (8192)	0.9436	1.087978	0.02487	43.745
Tri (16384)	3.5630	2.871914	0.04432	64.799
Tri (32768)	13.4933	8.120748	0.11925	68.097
Tri (65536)	53.4653	33.25809	0.22488	147.891
Tri (262144)	776.1905	ME	0.6421	-
Band5 (8192)	23.7676	13.697263	4.4736	3.141
Band5 (16384)	207.1346	29.362002	6.8811	4.267
Band5 (32768)	1897.354	58.43701	10.9187	5.351
Band5 (65536)	TLE	135.456462	22.5023	6.019
Band5 (262144)	ME	ME	46.1816	-
DKM (8192)	49.1961	99.914571	56.7133	1.761
DKM (16384)	217.2924	364.477255	231.924	1.571

* execution times in seconds

- `eig_lapack` – LAPACKE API based C++ implementation. Sequential.
- `hsseigen` – MATLAB based sequential implementation. Sequential.
- `hssedc_dist` – distributed-memory parallel implementation. Speedup is w.r.t. the best baseline i.e. `hssedc_seq`, our sequential C++ implementation. shows execution with highest core count (also the best one).

Results – strong-scaling



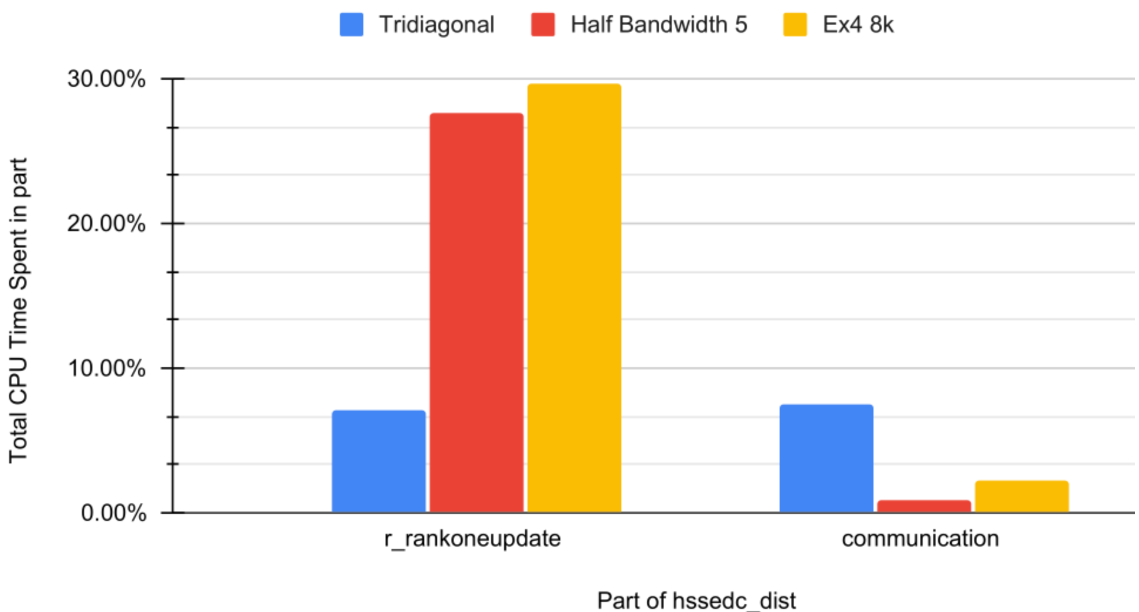
Symmetric tridiagonal inputs:

- larger input sizes yield better speedups. 147.8x speedup with 512-core execution of 64K sized input. Larger inputs also evaluated.
- Rank-1 updated involved. Finer HSS matrix decomposition makes more parallelism available.

Other (banded and DKM) inputs:

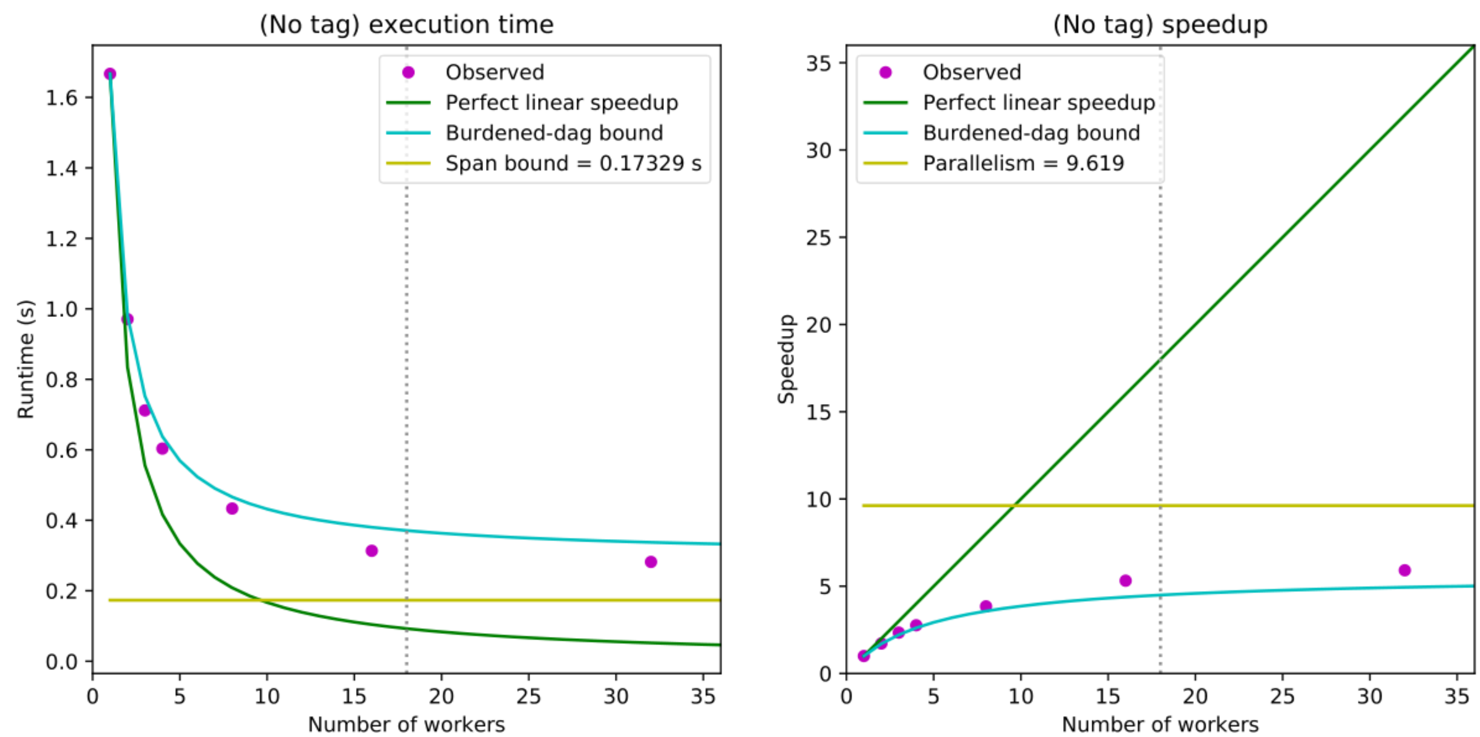
- Up to rank-r updates involved. This is inherently sequential.

Results – serial bottleneck



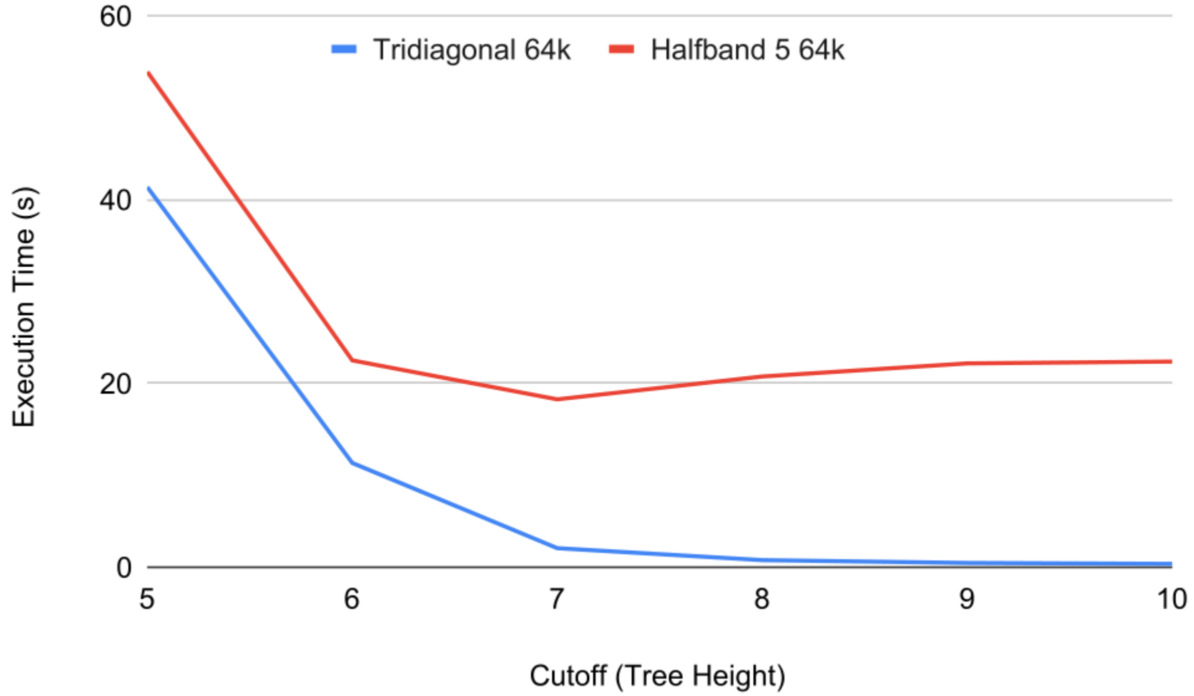
- Obtained from HPCToolkit.
- Percentage time spent in `r_rankoneupdate` increases for matrices having higher ranks in their off-diagonal blocks. This module is the serial part of the computation.
- Communication overhead is not the cause of smaller speedups in DKM and Banded matrices

Results – implementation overheads



- Data collected using CilkScale, scalability analyzer for OpenCilk programs.
- Shows that “observed” is in between “burdened dag bound” and “span bound”. This indicates that the implementation overheads, if any, do not significantly affect the performance.

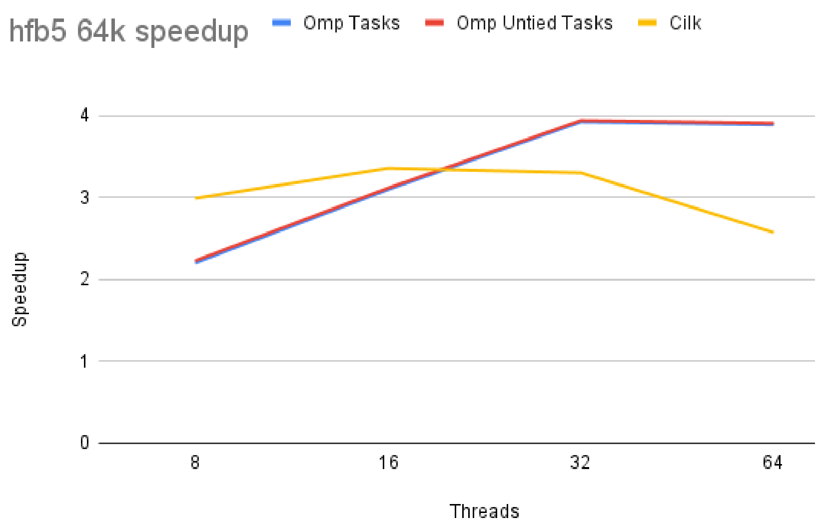
Results – tree decomposition

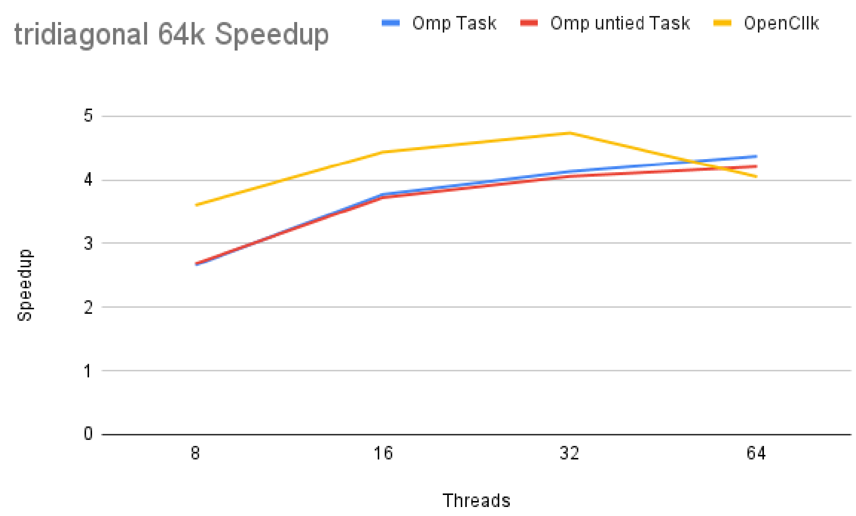


- Suitable height / level of the tree up to which parallel tasks can be spawned: = log(p) , p = number of processes / worker threads.

- Suitable partitioning scheme: split the tree horizontally at height / level = log(p). Let each subtree (arising out of split) be handled independently by processes.

Results – others





- Work stealing offers no benefit.
- OMP implementation is better than that of OpenCilk and work-stealing offers no major advantage.

Conclusions

SuperDC is a state-of-the art Divide-Conquer algorithm for computing eigenvalues and eigenvectors of Symmetric HSS matrices.

We optimize SuperDC to:

- allow for parallel execution of the Conquer stage.
- allow large HSS matrices to be input.
- reduce storage requirements for banded matrices from $O(N^2)$ to $O(N)$

Results show:

- Parallel implementations show scalable performance with tridiagonal inputs. For other inputs, the serial bottleneck causes slowdown.
- Overall, a significant improvement over the state-of-the-art implementation of SuperDC

References

* 2021. *hsseigen* https://github.com/fastsolvers
* James Vogel, Jialin Xia, Stephen Cauley, and Venkataramanan Balakrishnan. 2016. Superfast divide-and-conquer method and perturbation analysis for structured eigenvalue solutions. *SIAM Journal on Scientific Computing* 38, 3 (2016), A1358–A1382.
* Xiaofeng Ou and Jianlin Xia. 2022. SuperDC: Superfast Divide-And-Conquer Eigenvalue Decomposition With Improved Stability for Rank-Structured Matrices. *SIAM Journal on Scientific Computing* 44, 5 (2022), A3041–A3066. https://doi.org/10.1137/21M1438633
* Jan JM Cuppen. 1980. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.* 36, 2 (1980), 177–195.
* Ming Gu and Stanley C Eisenstat. 1994. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM journal on Matrix Analysis and Applications* 15, 4 (1994), 1266–1276.
* Jack J Dongarra and Danny C Sorensen. 1987. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Statist. Comput.* 8, 2 (1987), s139–s154.
* Nathan Tallent, John Mellor-Crummey, Laksono Adhianto, Michael Fagan, and Mark Krentel. 2008. HPCToolkit: performance tools for scientific computing. In *Journal of Physics: Conference Series*, Vol. 125. IOP Publishing, 012088.
* Tao B Schardl and I-Ting Angelina Lee. 2023. OpenCilk: A Modular and Extensible Software Infrastructure for Fast Task-Parallel Code. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming.* 189–203.