

Nearest Neighbour Search Algorithms for Spatial Data Mapping in Numerical Simulations

Abhishek Dhiman

Linköping University

Raghu Chaitanya Munjulury

`raghu.munjulury@liu.se`

Linköping University

Jordi Loureiro

Linköping University

Research Article

Keywords: Search Algorithm, Computational Fluid Dynamics, Data Mapping, Data Visualization, Subroutine, User-Defined Data Structures.

Posted Date: October 16th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11202539/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Nearest Neighbour Search Algorithms for Spatial Data Mapping in Numerical Simulations

Abhishek Dhiman¹, Raghu Chaitanya Munjulury^{1,2,*†},
Jordi Loureiro^{1,3,†}

¹Department of Management and Engineering, Linköping University,
Olaus Magnus väg, Linköping, 581 83, Östergötland, Sweden.

²SAAB Aeronautics, SAAB AB, Bröderna Ugglas gata, Linköping, 581
88, Östergötland, Sweden.

³Mechanical Integrity, Research and Development, Siemens Energy AB,
Slottsvägen, Finspång, 612 31, Östergötland, Sweden.

*Corresponding author(s). E-mail(s): raghu.munjulury@liu.se;
Contributing authors: sigmasix09@gmail.com; jordi.loureiro@liu.se;

[†]These authors contributed equally to this work.

Abstract

Improved data utilization and high-performance computing (HPC) resource management has greatly benefited the high-fidelity computational fluid dynamics (CFD) and multiphysics simulations, e.g., conjugate heat transfer (CHT). The work presents multiple implementations of the Nearest-Neighbour search method for efficient mapping of CFD-derived boundary conditions with data-driven interpolation to enable efficient steady-state thermal analysis. Consequently enabling custom spatial indexing of the high-resolution CFD data using user-defined data structures (UDDS) as FORTRAN90 subroutines for reduced memory overhead. Results of the implemented methods were compared with the Brute Force search approach, while ensuring compatibility with HPC, which demonstrated a significant reduction in simulation time on single and multiple cores of HPC.

Keywords: Search Algorithm, Computational Fluid Dynamics, Data Mapping, Data Visualization, Subroutine, User-Defined Data Structures.

1 Introduction

Continuous research and rapid development in the fields of data science, advanced data structures, and machine learning (ML) have increased opportunities for advancements in the CFD domain and improvements in data handling techniques. The underlying coupling in multiphysics simulations must respect the mathematical formulations and account for different mesh sizes, discretisation schemes, and interpolations used in each domain to maintain the conservation of properties and the order of accuracy. A fully coupled solver is the ideal and most accurate approach, but it leads to a large system of non-linear equations, resulting in high computational costs [1].

Subsequently, the multiphysics coupling in space and time is linked to interpolation, and the *Closest Pair of Points Problem* [2], which is a well-known fundamental problem associated with complex computational geometries. The existing literature for FEA data mapping provides four interpolation techniques like the nearest point method, distance method using the field of points, distance method using elements and method using element shape function [3]. The work focuses mainly on the nearest point method to map data from a fine CFD mesh onto a coarse structural mesh with computationally efficient [3] methods. Data structures (e.g., KD tree, Ball tree, Octree, slice representation) [4] solve this problem and are associated with time complexity and space dimensionality. The base of the present study is the master thesis work presented at Linköping University in 2024 [5] and can be referred for theoretical knowledge.

The scope of the work presented is limited by ABAQUS CAE 2019-2021 and FORTRAN90 (analysis subroutine development). The Python (version >3.7) at pre-processing offers great flexibility for algorithm development and data handling with built-in libraries and dedicated modules unavailable in Python 2.7. The speed of the FORTRAN programming language and subroutine availability are favourable for building/implementing a search algorithm. However, being old and simple promotes the Brute implementation of code and the lack of built-in libraries/modules makes it difficult to work with.

2 Methodology

The mapping is performed using the nearest point (neighbour) search method in several ways that are briefly explained below. Two methodologies are initially implemented as base for the data mapping using FORTRAN90 fixed form [6], namely, the Brute-Force and MATLAB subroutine file approaches. Then improved methodologies, namely, Bin and L1-Octree, are developed keeping in mind the functionalities offered by FORTRAN90 and ABAQUS 2019. The four approaches are tested on Case 1 (hollow cylinder) for benchmarking, and Case 2 (exhaust valve) case is used for validation of the methodologies. In the previous methodology, the data file is read using a MATLAB script, pre-processed using MATLAB functions, and segmented (bins) along one axis (Z). The FORTRAN90 subroutines and data bins are hard-coded in FORTRAN90 fixed form [6] and the subroutine file is created with (.f) extension. Henceforth, this method will be referred to as MATLAB approach.

Brute-Force approach (linear search) to find the nearest-neighbour based on the distance between all pairs of points in the dataset using Cartesian coordinates of the points. To speed up the algorithm runtime, the square root operation is omitted during distance calculation, which still yields identical results [7], and the point with the minimum or zero distance is identified as the closest neighbour.

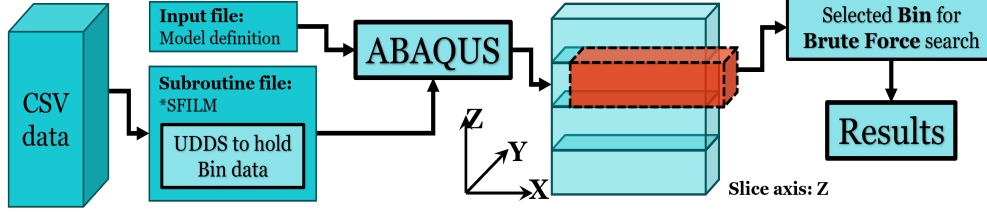


Fig. 1: Discretization or Bin approach flow chart

The Bin approach is implemented in FORTRAN90, based on data categorization and the "Discretise" function available in MATLAB. The Bin approach is a combination of Discretise and Brute-Force, which divides the data into a specified Bins along an axis Fig. 1. A UDDS is used to store the data with FORTRAN90 *SAVE* attribute, making it a one-time activity. The UDDS capability was introduced in FORTRAN90 [6] and allows users to create derived data types.

The L1 Octree (L1.Octree) approach is another implementation that uses a UDDS, similar to the Bin approach, to store the data. Instead of discretising data into Bins, octants are created, shown in Fig. 2. Here, "L1" implies that the entire data is divided into 8 octants only (further division possible). Data pre-processing or sorting is not required for Bin and L1.Octree approach. The Octree utilises a bottom-up approach, i.e., the domain is discretised from inside-out, unlike the top-down Bin approach. The octant centroid information is also stored to identify the closest octant and its limits.

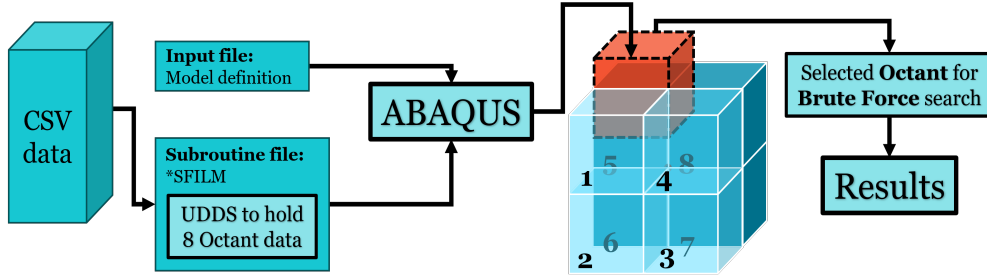


Fig. 2: L1.Octree approach flow chart

2.1 ABAQUS Analysis and Methodology Implementation:

A simplified working of the analysis step is shown in Fig. 3. The central station of the entire workflow is the ABAQUS which takes two files namely the input file and the subroutine file. The input file contains the geometric data, boundary conditions, and load details (with ABAQUS keywords). The working of the relevant keywords is explained above. The subroutine file contains the corresponding subroutines for the keywords used in the input file. During the analysis, ABAQUS applies loads and solves the problem for each node and integration point. The coordinates of these points are passed to the corresponding subroutines where the *sink* and *htc* values are initialized for that point. The values for *sink* and *htc* are from the closest point found in the CSV data file using various approaches e.g. Brute, Bin, L1-Octree, etc Fig. 3. This process repeats for all the points given by ABAQUS to the subroutine and post-completion all the points have the HTC and SINK values and the job is completed. Next is the post-processing step, where meaningful contours, plots, and other relevant charts are extracted to convey the results and present the findings from the analysis. For the present case, the results are fetched to evaluate how well the search approach has performed in finding the closest points from the CSV data file to the point required by ABAQUS and in terms of execution taken by the search approach.

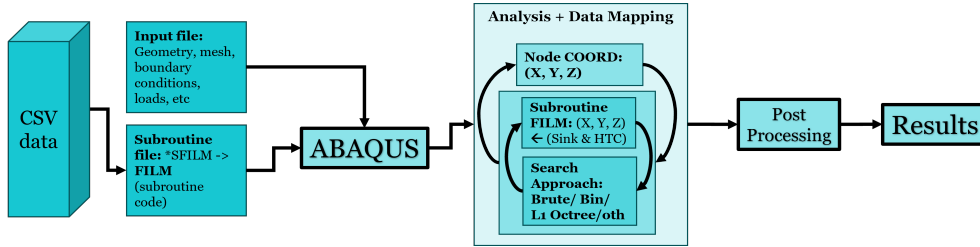


Fig. 3: ABAQUS analysis flow chart

2.1.1 ABAQUS Command: *SFILM:

The *SFILM ABAQUS keyword is used to define heat transfer or film coefficient (HTC) and sink temperature (T_{fluid}) over the surface for a fully coupled thermal stress, heat transfer, and coupled thermal electrical analysis [8], Eq. 1, Newton's law of cooling. For non-uniform film coefficient and sink temperature, the values must be defined in the user subroutine (FILM [8]). During the analysis, the *SFILM keyword mentioned in the ABAQUS input file directs to the FILM subroutine in the subroutine file. Under the FILM subroutine, the user can write the FORTRAN code to assign the value for HTC and SINK. The Keyword *SFILM and subroutine FILM are surface-based, the convection is applied to edges in 2D and faces in 3D [8]. The FILM subroutine snippet is readily available in ABAQUS documentation and hence not discussed in the report [8].

$$q = HTC * (T_{surf} - T_{fluid}), \quad (1)$$

where, q = surface heat flux due to convection, HTC = heat transfer coefficient, T_{surf} = temperature on the surface and T_{fluid} = SINK = reference sink or fluid temperature [8].

2.1.2 ABAQUS Command: *FIELD:

Next, is the *FIELD keyword used to specify pre-defined field variable values and the keyword must be specified in the ABAQUS input file similar to *SFILM. When specified, *FIELD directs to the UFIELD subroutine for user-defined values of the field variables during the analysis and will be called for each node. Unlike the *SFILM and FILM combination, the values of the field variables are prescribed at the nodes of the model [8]. The user subroutine interface for UFIELD is also readily available in the ABAQUS documentation, hence not discussed in the report. The results from both *SFILM and *FIELD keywords are discussed later in the report, where the salient features are highlighted.

2.2 Case1 Bench-marking: Hollow Cylinder and Boundary Conditions:

The verification of the two improved implementations was performed on the hollow cylinder with two holes case, Fig. 5. The verification of the approaches is evaluated based on the job (analysis) execution time, contours of remapped data, and the plots of HTC and SINK values to provide visual means for better understanding. The Wall Clock and CPU time are selected along with the Compilation and Pre-Run of the FORTRAN code because the FORTRAN code is compiled first and then executed. Various meshes are created for the case geometry with HTC and SINK values generated using Eq. 2 and 3 respectively. These equations are based solely on the coordinates of the integration point and do not comply with the physics. These equations only aim to make HTC and SINK values solely dependent on the coordinates and obtain an Analytical solution reference for post-processing of the results.

$$sink = \sqrt{|coordinate(Z)|} + 30 \quad (2)$$

$$htc = \sqrt{(|coordinate(X)| + |coordinate(Y)|)} + 100 \quad (3)$$

where coordinates is the coordinate of the integration point needed by ABAQUS during the analysis.

For the readers basic understanding, a brief theory of the difference between compilation and execution of a FORTRAN code is provided. The compilation process converts the user-written FORTRAN code into computer-readable form, thus every line of code is checked to generate an object or executable file for the code [9] (a good source for FORTRAN basics). Certain syntax-related issues or lines of code incompatible with FORTRAN are identified by the compiler like the continuation line error. Once the compilation is completed, the FORTRAN code is executed and execution errors are identified like the provided file directory is invalid, etc. As a benchmark case, a hollow cylinder with two holes is selected so that the mesh refinements can be applied near the holes resulting in high node density Fig. 4. In the present work,

the data of a finer mesh is mapped onto the mesh of a coarse mesh, thus for a visual representation both are shown in Fig. 4.

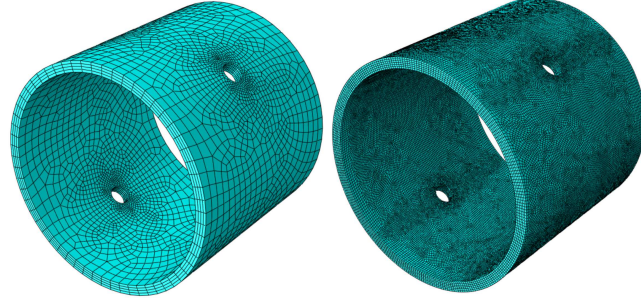


Fig. 4: Coarse (left) and fine (right) mesh: Case1 Hollow cylinder

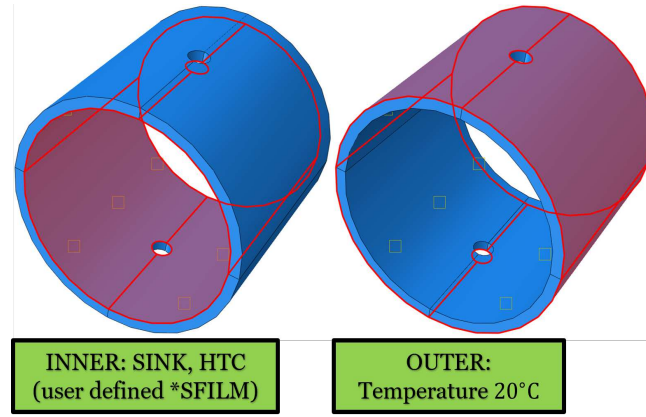


Fig. 5: Boundary conditions for Case1: Hollow cylinder

Post mesh creation, the boundary conditions and loads (thermal loads in the present case) are applied at the required regions. For the entire geometric model the *Conductivity, *Density, *Elastic, *Expansion, *Specific Heat are defined in the ABAQUS input file. The presence of (*) implies ABAQUS keywords [8]. The ABAQUS model uses two types of analysis steps 1: Initial step (20°C at all nodes) and 2: Analysis Step. The initial step is used to apply boundary conditions, predefined fields, and interactions required at the start of analysis [8]. Whereas, multiple analysis steps can be added depending on the type of analysis (steady or transient state) [8]. In the present case, only one analysis step is included to achieve a steady-state analysis. The boundary conditions are applied on the inner and outer surfaces of the model at the initial step but for the analysis step the HTC and SINK are user-defined and applied using *SFILM ABAQUS keyword [8] Fig. 5.

3 Results

The four methodologies are implemented and evaluated qualitatively and quantitatively. The results obtained are compared to evaluate the quality of the mapping and time complexity. The performance influencing factors like data sorting, level of partitioning (Number of Bins), and multi-processor execution, are also evaluated.

3.1 Case 1: Hollow Cylinder Mapping

This case serves as the benchmark case to evaluate the quality of data mapping onto a coarser mesh Fig. 6a. The coarse mesh is shown in the top-left, and the SINK values from finer meshes (increasing order) $\in \{\text{Mesh1 (M1), Mesh2 (M2), Mesh3 (M3), Mesh4 (M4)}\}$ are mapped using three approaches against the Analytical solution. The Brute-Force is the base and also a reference for comparison. An interesting result is from the influence parameter, i.e., data sorting requirement for approaches like KD tree. The data sorting results are tested on the Bin approach for various bins, showing high-unsteady wall clock time when data is sorted along the z-axis, but no data sorting results in steady and reduced wall clock time as the bins increase 6b.

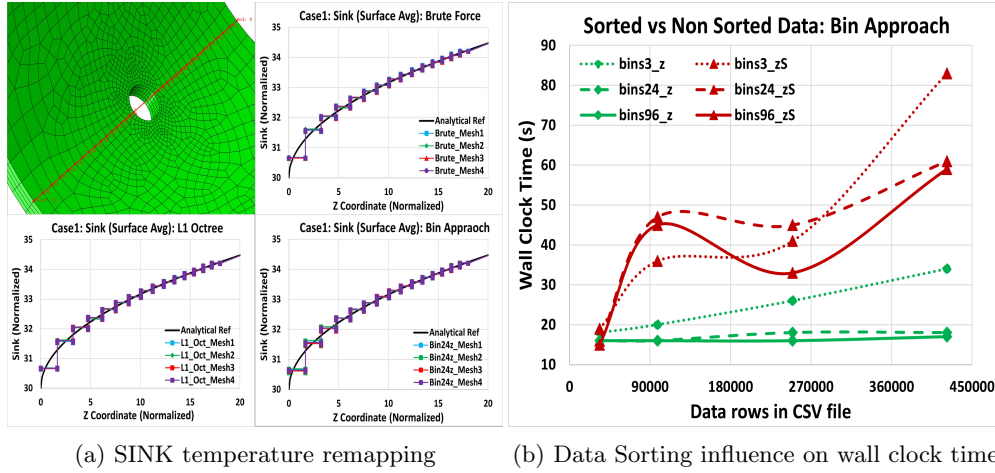


Fig. 6: Approach comparison and influence of data sorting

The SINK contour, Fig. 7, M4 data was mapped onto M1 for qualitative analysis. The Base M1 is the simulation result and M4 SINK values are mapped using the Brute, Bin, and L1-Octree approaches, showing good agreement with the reference Base M1 with negligible discrepancies at transitions. Similarly, the HTC mapping in Fig. 8 with transition direction different from the SINK due to Eq. 2.3. A closer look on L1-Octree approach captures the low-level transitions better than Bin approach. This minor difference is also quantitatively shown in Fig. 9, comparing approaches for each mesh separately. The stair-stepped plot results imply a constant value for each surface element, as shown in Fig. 6a.

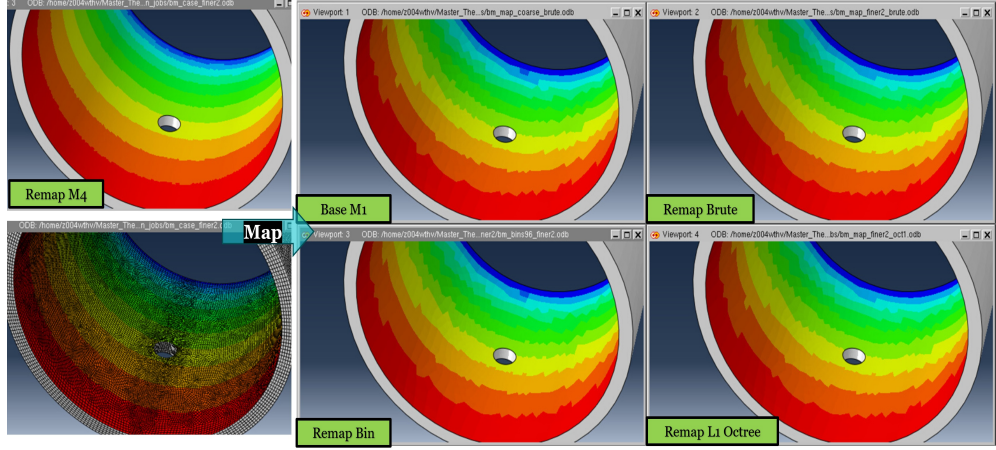


Fig. 7: Remapped normalized SINK data contour onto M1

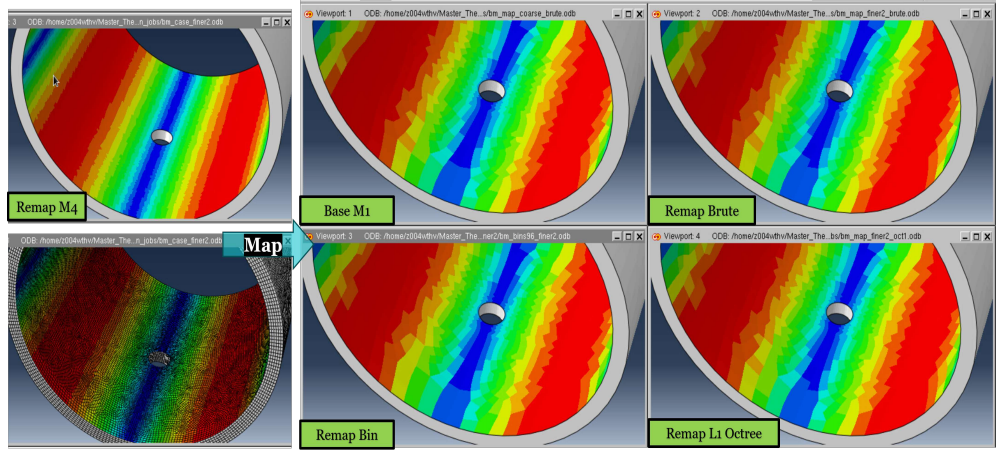


Fig. 8: Remapped normalized HTC data contour onto M1

Further analysis on Bin approach was performed to identify convergence for the number of bins, Fig. 10. The bins are ranged from 3 to 96 and evaluation is performed against the two parameters i.e. the CPU and Wall Clock time. Both parameters show similar plot trends and converge for number of bins more than 20. The CPU and Wall Clock time reduction peaks at 44.71 % and 50.99 %, respectively. The Wall Clock time comparison is shown in Fig. 11, where L1-Octree approach with 8 octants show a 44.85% - 80.68% reduction in wall clock time from lowest to highest data rows in the csv file. For the Bin approach this improves as the bins increase and both high reduction compared to the Brute approach. The reduction of 62.94% - 92.40% compared to Brute is shown by MATLAB approach, the finest mesh, M4 is used in this work.

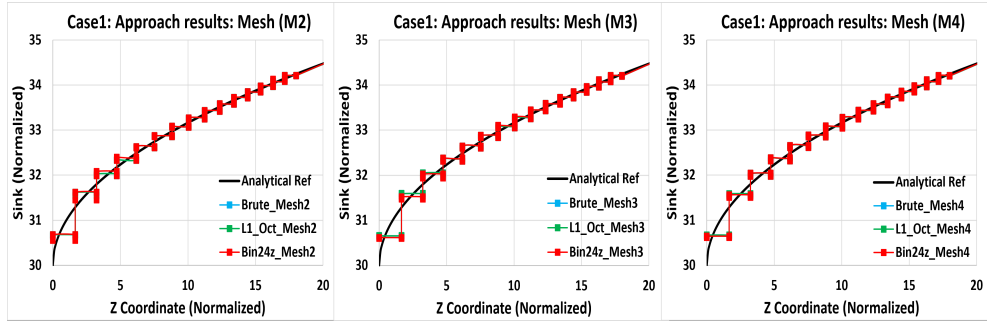


Fig. 9: Comparison of remapped SINK by different approaches on meshes

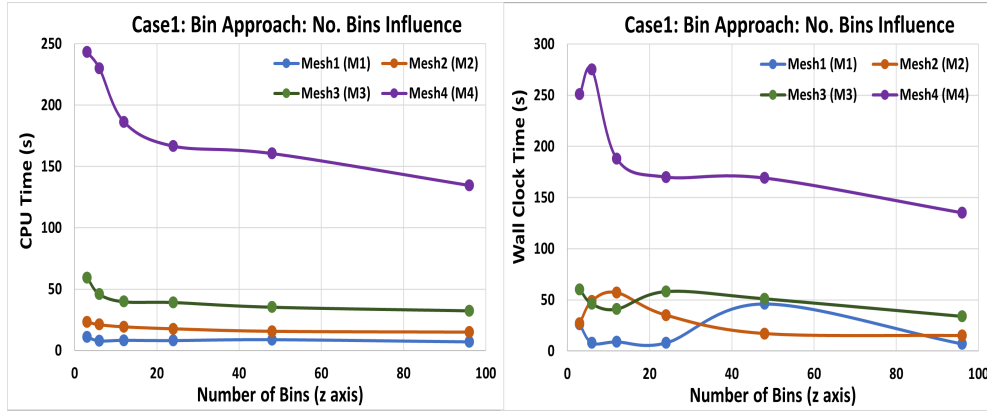


Fig. 10: Performance evaluation of Bin approach w.r.t CPU and Wall Clock time

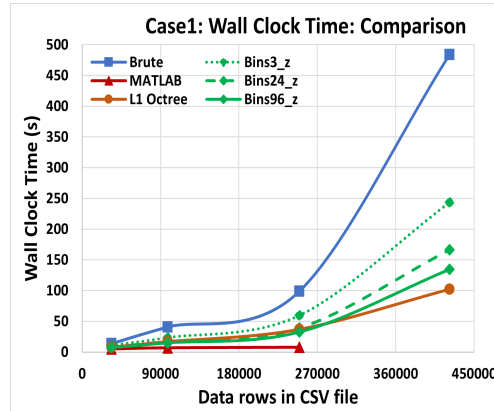


Fig. 11: Wall Clock time comparison

3.2 Case 1: Multi-Core Analysis

The approaches are implemented with thread-safe and shared memory architecture for multi-core or cluster-level executions. The shared memory is available across multiple threads or cores to parallelize and speed up the simulation. The results in Fig. 12 show the analysis for a small case with 11204 mesh elements. The CPU and the Wall Clock time trends are similar and increase with the number of cores. The CPU and Wall Clock time steadily increase for cores < 8 , but converge for cores ≥ 8 . The L1.Octree approach reduces the CPU time by 87.5% and the Wall Clock time by 87.6%, whereas the Bin approach reduces the CPU time by 60.72% and the Wall Clock time by 60.81% for 1 core, Fig. 12. The % decrement seems to be consistent w.r.t number of cores. The L1.Octree and Bin (96 Bins) approach show a negligible difference, but for 1 core the time taken by L1.Octree is significantly smaller than the Bin approach i.e. 68.18%, for the rest of the cores the offset in the curves is constant.

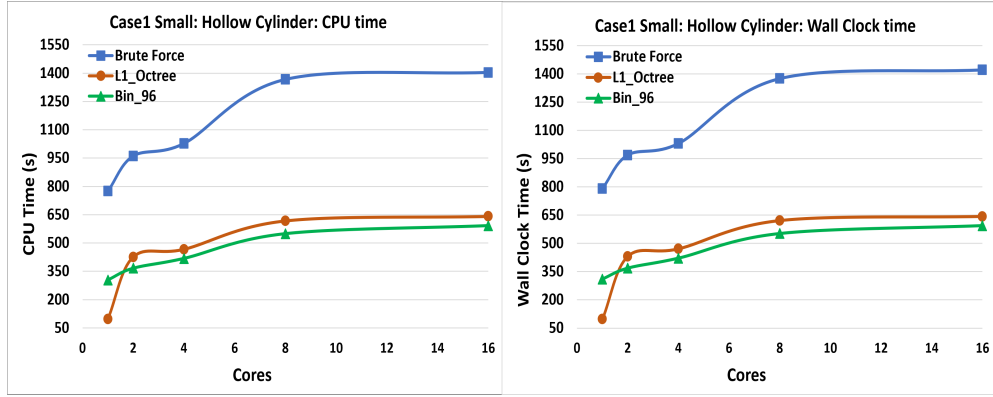


Fig. 12: Benchmark: Influence of No. of Cores for a small case with 11204 elements

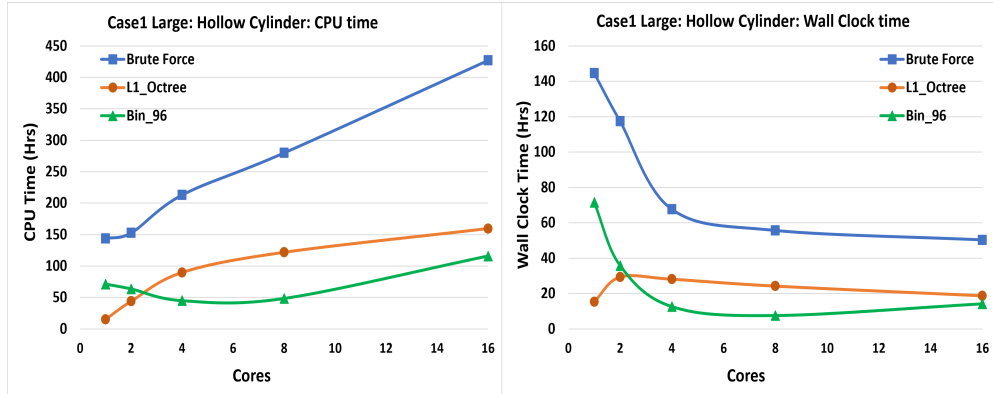


Fig. 13: Benchmark: Influence of No. of Cores for a small case with 11204 elements

The CPU Time result for a larger case with 2138819 quadratic elements (Fig. 13) is similar to Fig. 12. The CPU time in Fig. 13 shows that Bin_96 approach outperforms the Brute and L1-Octree approach for Cores ≥ 2 and then a constant offset w.r.t to L1-Octree approach. However, the L1-Octree approach CPU time on a single core performs significantly better than the Bin_96 and Brute Force approaches. The Wall Clock time in Fig. 13 shows good convergence w.r.t number of cores, but the with single core L1-Octree outperforms the Bin_96 and Brute Force approach with a great margin. The clear offsets between the three approaches for both CPU and Wall Clock time can be seen, especially for the Wall Clock time where the offset trends are constant and smooth for Cores ≥ 4 , Fig. 13.

3.3 Case 2: Exhaust value

The case, comparing 317,418 quadratic tetrahedron elements (ABAQUS DC3D10) is suited for diffusive heat transfer or mass diffusion. The Brute and MATLAB approaches serve as references in Fig. 14, against which the Bin and L1-Octree approaches show good agreement. The high HTC values at the inlet edges of 4 pipes and regions of low HTC are well captured. The time improvements are summarised in Table .1, with highest compilation time and lowest Wall clock time for the MATLAB approach w.r.t Brute approach. The Bin approach improvements show convergence as the bins increase, and L1-Octre approach results are close to Bin_96.

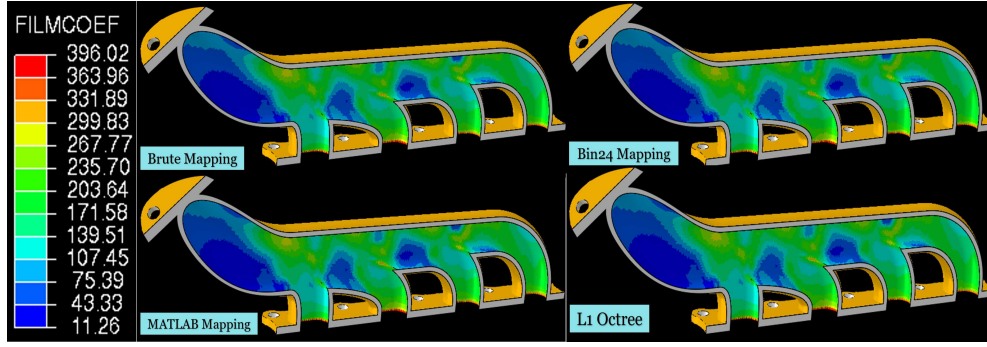


Fig. 14: Exhaust valve: Mapping comparison for HTC ($W * K/m^2$) contour

Table 1: Time improvement (%) of approaches w.r.t Brute Force, (-) = Increase

Time	Brute	MATLAB	L1-Octree	Bin_3	Bin_24	Bin_96
CPU	0	46.17	38.69	29.67	39.90	39.85
Wall Clock	0	46.23	38.69	29.37	39.48	39.68
Compilation	0	-13.33	-2.22	0	-6.67	-2.22

The analysis data for Wall Clock and CPU time by each approach is shown in Fig. 15. The Brute approach being the simplest implementation takes the highest CPU and Wall Clock time. The percentage reduction is quantified in Table. 1 which shows the Wall Clock time for the MATLAB approach is reduced by 46.23%, CPU time by 46.17% w.r.t Brute approach, but the Compilation time is increased by 13.33% begin the highest.

The stabilization in Wall Clock and CPU time for the Bin approach as the No. of Bins increase is also captured for Case2 (Fig. 15) similar to Case1 Fig. 10. The CPU time reduction for the Bin approach compared to the Brute approach is in the range of 29.67% - 39.90% and the same for the Wall Clock time, although a slight increase in Compilation time is observed (2.22% - 6.67%; Table. 1). Similarly, the Wall Clock and CPU time reduction by L1.Octree approach is 38.69% and increment of 2.22% in Compilation time w.r.t Brute approach. The result trends of the Exhaust Valve are comparable to the verification case i.e. Case1 hollow cylinder.

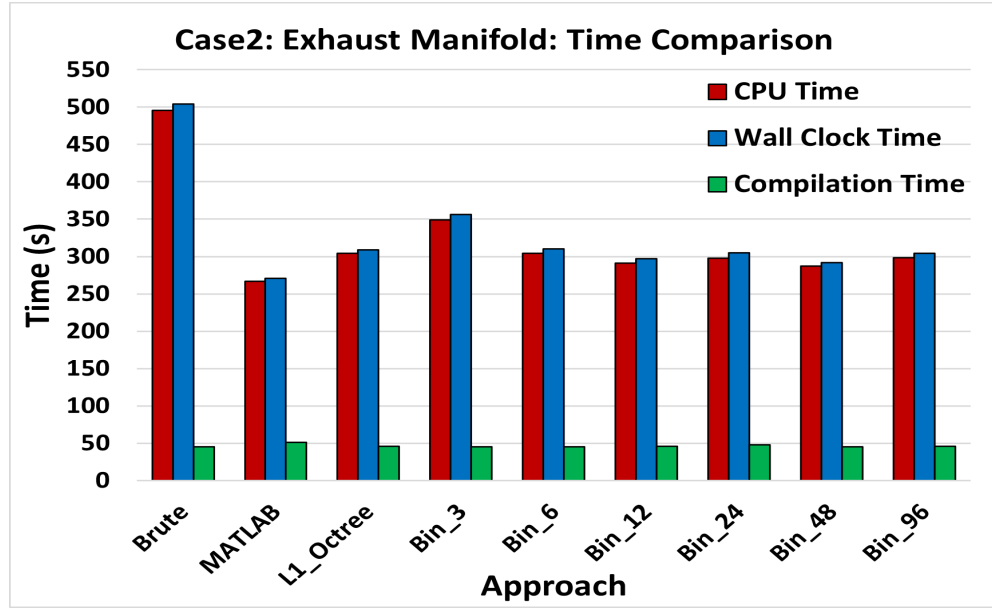


Fig. 15: Exhaust valve: Wall Clock and CPU time (s) comparison

4 Discussion

Good qualitative and quantitative agreement for SINK and HTC values is achieved using Bin and L1-Octree approaches, Fig. 6a, 7, 14. At the root level, integration points (Full & Reduced) [10] of quadratic elements (ABAQUS) give the flexibility for detailed mapping resulting in better surface averaged values. The spatial indexing and partitioning [4] lead to efficient domain segmentation and reduce the nearest neighbour search time for mapping, thus improving the Wall Clock time, Fig. 11. When a bin or octant is identified for search, the Brute force approach is applied improving the linear time complexity of overall search.

Next, two specific analyses performed for the Bin approach i.e. the data sorting operation Fig. 6b and the influence of the number of Bins on the CPU and Wall Clock time Fig. 10. The increment in the number of Bins reduces the analysis time significantly and stabilizes well for Bins > 24 for the Benchmark case i.e. a positive influence Fig. 10 because more Bins lead to the smaller subdomain and thus fewer points for the Brute Force to search.

The influence of data sorting on analysis time is negative Fig. 6b, leading to unstable trends as the dataset volume increases. Further analysis showed that the time taken for sorting the dataset was a significant percentage of the total time as FORTRAN90 does not have an optimized intrinsic sort function [6]. Based on the benchmarking case and the obtained results, the KD-tree algorithm is ruled out as it requires finding the median value and performing the space partitioning [11] [12]. Consequently, the L1-Octree and Bin approaches are implemented for the validation cases.

The multi-core analysis of large case (2,138,819 elements or DOF) shows consistent Wall Clock time reduction, Fig. 13, which complies with the ABAQUS documentation [13]. The multi-core analysis reduces the Wall Clock time, however, internally, the combined CPU time of each process is higher because of the cache coherence and resource contention [14] as each processor creates a local copy of the data, and the original shared data is updated as the input comes from multiple processors. However, for a small case (Fig.12), both CPU and Wall Clock time increase w.r.t cores. The overhead of establishing communication between processors to achieve parallelization is more than the job execution time itself, thus multi-core analysis is not recommended for $\text{DOF} < 100,000$ [13].

The data mapping results on the exhaust value (Fig. 14) are also in good agreement. It is a valve device used in the Automotive industry to collect high-temperature gases from multiple channels into a single pipe. It cools down the exhaust gases from the combustion process, thus, a good example to show CHT.

The results from the parallelization of the benchmark case offer the advantage of limiting the size of the validation case to a smaller size. The validation case in Fig. 14 has 317,418 elements (< 2 million), which will not show significant performance improvements with parallelization [13] and avoid the overhead of parallelization [15]. Therefore, the results presented in Fig. 15 and Table. 1 are for single-core analysis.

5 Conclusion & Future Scope

New methodologies (Bin and L1-Octree) were developed for the nearest neighbour search method, showing improved performance in qualitative and quantitative metrics with respect to the existing implementation (Brute and MATLAB subroutine). The influence parameters like data sorting, domain segmentation level, single- and multi-core analysis were performed to evaluate methodology robustness and limits. The recommendations are to use the new methodologies for $\text{DOF} > 100,000$, as the wall clock time reductions are 44.85% - 80.86% and offers low code compilation time. The current implementation is developed for use in ABAQUS and its internally used programming language, FORTRAN90, for subroutines. Thus, the search algorithms available in modern languages like Python, MATLAB, etc, are not readily available for use. However, the development of new approaches such as InSitu, InTransit, Hybrid approaches, GPU programming, open-source tools, leads to further developments in the data mapping domain with feasible integrations with ML models, which are highly flexible and offer significant customization opportunities for concurrent simulation, scientific inference, and data visualization with virtual reality (VR).

Declarations

Not Applicable

Editorial Policies for:

Springer journals and proceedings: <https://www.springer.com/gp/editorial-policies>

Nature Portfolio journals: <https://www.nature.com/nature-research/editorial-policies>

Scientific Reports: <https://www.nature.com/srep/journal-policies/editorial-policies>

BMC journals: <https://www.biomedcentral.com/getpublished/editorial-policies>

References

- [1] Keyes, D.E., McInnes, L.C., Woodward, C., Gropp, W., Myra, E., Pernice, M., Bell, J., Brown, J., Clo, A., Connors, J., Constantinescu, E., Estep, D., Evans, K., Farhat, C., Hakim, A., Hammond, G., Hansen, G., Hill, J., Isaac, T., Jiao, X., Jordan, K., Kaushik, D., Kaxiras, E., Koniges, A., Lee, K., Lott, A., Lu, Q., Magerlein, J., Maxwell, R., McCourt, M., Mehl, M., Pawlowski, R., Randles, A.P., Reynolds, D., Rivière, B., Rüde, U., Scheibe, T., Shadid, J., Sheehan, B., Shephard, M., Siegel, A., Smith, B., Tang, X., Wilson, C., Wohlmuth, B.: Multi-physics simulations: Challenges and opportunities. *The International Journal of High Performance Computing Applications* **27**(1), 4–83 (2013) <https://doi.org/10.1177/1094342012468181> <https://doi.org/10.1177/1094342012468181>
- [2] Shamos, M.I., Hoey, D.: Closest-point problems. In: 16th Annual Symposium on Foundations of Computer Science (sfcs 1975), pp. 151–162 (1975). <https://doi.org/10.1109/SFCS.1975.8>
- [3] Afazov, S.M., Becker, A.A., Hyde, T.H.: Development of a finite element data exchange system for chain simulation of manufacturing processes. *Advances in Engineering Software* **47**(1), 104–113 (2012) <https://doi.org/10.1016/j.advengsoft.2011.12.011>
- [4] Chen, T., Schneider, M.: Data structures and intersection algorithms for 3d spatial data types. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 148–157. Association for Computing Machinery, ??? (2009)
- [5] Dhiman, A.: Nearest neighbour search method for 3d data mapping of cfd data : Steady state thermal analysis. Master’s thesis, Linköping University, Fluid and Mechatronic Systems (2024)
- [6] Adams, J.C., Brainerd, W.S., Martin, J.T., Smith, B.T., Wagener, J.L.: *Fortran 90 Handbook* vol. 32. McGraw-Hill New York, ??? (1992)
- [7] Vaidya, P.M.: An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry* **4**, 101–115 (1989)
- [8] Hibbitt, Karlsson, Sorensen: *ABAQUS/Standard: User’s Manual* vol. 1. Hibbitt, Karlsson & Sorensen, ??? (1997)
- [9] Hiranandani, S., Kennedy, K., Tseng, C.-W.: Evaluating compiler optimizations for fortran d. *Journal of Parallel and Distributed Computing* **21**(1), 27–45 (1994)
- [10] Adeeb, S.: *Introduction to Solid Mechanics and Finite Element Analysis Using Mathematica*. Kendall Hunt, ??? (2011)

- [11] Beineke, L.W., Pippert, R.E.: Properties and characterizations of k-trees. *Mathematika* **18**(1), 141–151 (1971)
- [12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
- [13] Online: Abaqus Performance Data. <https://www.3ds.com/support/hardware-and-software/simulia-system-information/abaqus-69/performance-data/>. Accessed 05/05/2024
- [14] Thomadakis, M.E.: The architecture of the nehalem processor and nehalem-ep smp platforms. *Resource* **3**(2), 30–32 (2011)
- [15] Obbink-Huizer, C.: 7 Tips to Help Abaqus Run Faster. <https://simulation-blog.technia.com/simulation/7-tips-to-help-abaqus-run-faster>. Accessed 05/05/2024