Proceedings of the Eighth International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering
Edited by: P. Iványi, J. Kruis and B.H.V. Topping
Civil-Comp Conferences, Volume 12, Paper 2.2
Civil-Comp Press, Edinburgh, United Kingdom, 2025
ISSN: 2753-3239, doi: 10.4203/ccc.12.2.2
©Civil-Comp Ltd, Edinburgh, UK, 2025

Parallel Application of Multi-Freedom Constraints Using Master-Slave Method in Sparse Linear Systems

C. Topal[1], N. Muhtaroglu[2] and G. Kiziltas[1]

[1] Mechatronics Engineering, Sabanci University, Istanbul, Turkey
[2] Computer Science, Ozyegin University, Istanbul, Turkey

Abstract

Multi-freedom constraints (MFCs) are commonly used in matrix formulations to enforce dependencies among multiple components, particularly in structural analysis where they are defined based on the degrees of freedom (DOFs) at nodes or computation points. A widely used approach for implementing MFCs is the master-slave elimination method, favored for its simplicity and its ability to reduce the number of unknowns. While straightforward to implement with full matrix storage, this approach can lead to increased memory usage. Conversely, applying it to sparse matrices presents added complexity. This paper introduces a scalable and reusable implementation of the master-slave method tailored for large-scale linear systems with multiple non-homogeneous constraints. The approach leverages parallel programming and distributed processing to efficiently handle computational demands. PETSc 3.23.1 is used as a tool for parallel computing due to its higher-level encapsulation of MPI operations and built-in sparse matrix representation methods. The algorithm's syntax is designed for efficient memory handling. Parallel MPI library enables load balancing across processors and threads.Benchmark results show that the proposed algorithm speeds up process solving linear systems with multi-freedom constraints.

Keywords: linear equations, sparse matrices, multi-freedom constraints, MPI (message passing interface), parallel computation, PETSc.

1 Introduction

Linear systems presented in numerical problems often rely on multiple constraints to be solvable. A constraint condition is classified as a multi-freedom constraint (MFC) when the constraint of one unknown is given in terms of the value of another unknown in the linear system, rather than a constant, as in single-freedom constraints. Although MFCs are applied in many areas over several disciplines including dynamics [1], structural analysis [2], deep learning and finance, their use in mechanical stiffness equations is more prominent than others.

MFCs are often defined in canonical form as seen in (1), where all the unknowns (in this case, u_1, u_2, and u_3) are gathered on the left-hand side of the equation:

$$au_1 + bu_2 + cu_3 = g \tag{1}$$

A multi-freedom constraint is classified as homogenous if the prescribed value g on the right-hand side (called gap) is zero. Otherwise, it is called a non-homogenous MFC. The most general case for MFCs may include inequality instead of equality. However, it is less frequent in linear structural analysis and more important in the fields of control and optimization. MFC application is especially useful in enforcing periodic boundary conditions, where the periodicity constraint is defined between boundary points or nodes. Several methods for implementing MFCs exist that mainly

can be grouped as either in the first group which is the Penalty method [6], Lagrange multipliers and their combination in form of the augmented and perturbed Lagrange multipliers or the second group which includes the master-slave elimination method. In the first group of methods, the finite element approach with constraints is mathematically formulated as a constrained optimization problem. A comprehensive overview of multiplier techniques—including the Lagrange multiplier, augmented Lagrange multiplier, and perturbed Lagrange multiplier — as well as the penalty method, is provided in the textbook by Belytschko et al [5].

The master-slave elimination method is widely favored for its conceptual simplicity and has been shown to provide improved accuracy in enforcing displacement constraints in structural analysis \cite{zheng}. However, the reduction of slave degrees of freedom necessitates a reordering of the system equations, which can introduce additional complexity when employing sparse matrix storage schemes. The computational efficiency of this method can be significantly enhanced through parallelization. To address this need, in this study we present a generalized implementation of the master-slave approach for applying non-homogeneous multi-freedom constraints (MFCs) to large-scale linear systems. The method is evaluated quantitatively in terms of computational performance and implementation efficiency. Although MFCs can be applied to any linear equation system, for the sake of examples, stiffness equations will be used in this study. This paper is structured as follows: first, an overview of how MFCs are imposed on the system, and then a demonstration of the efficiency of the proposed algorithm, via examples comparing results in terms of speedup, and computation time, which are then analyzed with respect to the corresponding number of processes and constraints.

Background: Imposing Multi-Freedom Constraints in a Nutshell

Here we summarize the basics of imposing multi-freedom constraints. Multi-freedom constraints are imposed on a linear system by modifying the components of assembled stiffness equations. This produces a modified linear system as follows:

$$Ku = f \xrightarrow{\text{MFCs}} \hat{K}\hat{u} = \hat{f} \tag{2}$$

where K is the sparse stiffness matrix and u denotes the nodal constraints (e.g., displacement values in structural analysis).

Alternative methods such as the Lagrange multiplier method add unknowns to represent constraint forces, while the penalty function method introduces weighted imaginary elements to impose constraints.

The master-slave method, unlike others, separates constraint elements into master and slave categories, eliminating or modifying slave freedoms explicitly. This provides benefits in terms of spatial complexity and simplicity of implementation. A step by step implementation flowchart of the master-slave elimination method is shown in Figure 2.

Background: Master-Slave Elimination Method in a Nutshell

The master-slave elimination method in short uses a (usually sparse) transformation matrix T to apply constraints by transforming the displacement vector u into a reduced vector $\hat{u}$, where slave freedoms are excluded:

$$u = T\hat{u} + g. \tag{3}$$

The equation system then turns into the modified system described on the right-hand side in Eq. (2), in which,

$$\hat{K} = T^T K T, \quad \hat{f} = T^T(f - Kg). \tag{4}$$

After solving the system, the eliminated slave freedoms can be recovered by simply using Eq. (3), where g is the gap vector representing constraint offsets in the case of non-homogeneous constraints. For homogeneous constraints, the gap vector is a zero vector. More details of the method can be found in XXX.

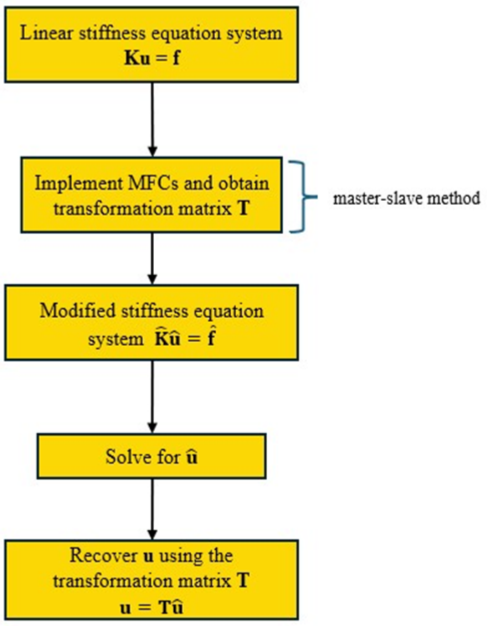
Linear stiffness equation system
$\mathbf{Ku} = \mathbf{f}$
Implement MFCs and obtain transformation matrix $\mathbf{T}$
master-slave method
Modified stiffness equation system $\hat{\mathbf{K}}\hat{\mathbf{u}} = \hat{\mathbf{f}}$
Solve for $\hat{\mathbf{u}}$
Recover $\mathbf{u}$ using the transformation matrix $\mathbf{T}$
$\mathbf{u} = \mathbf{T}\hat{\mathbf{u}}$

Figure 1: MFC application scheme

2 Proposed Methodology: Efficient Master-Slave Elimination Algorithm

Since the standard master-slave elimination method reduces slave freedoms to form a new ”master” equation system, it can be disadvantageous in computer applications because it requires rearrangement of the original equation system. The modified solution vector $\hat{u}$ is a subset of the original solution vector u. Therefore, it is important to systematically modify the transformation matrix T by storing the active row and column indices. In the proposed parallel algorithm implementation, elements (DOFs) in the constraint equations are stored in C++ objects called ”Term”, which are then assigned as either master or slave attributes within a constraint object.

The most important computational routine in constraint application is the generation of the transformation matrix T. Initially, T is an identity matrix of size $n \times n$, where n is the number of active freedoms. Constraint objects, which include index and coefficient data for slave and master freedoms, are parsed. The transformation matrix is then reduced in column size by excluding the columns with the slave terms' indices. Mapping between the original and reduced transformation matrix is established, as this mapping information is required at later stages. For each master term in the equation, the corresponding row of the master term is added onto the row positioned at the slave term's index after factoring it with the ratio between the master and slave term's coefficients.

The gap value contained in the constraint objects are also stored in a gap vector after being factored by the coefficient of the slave terms.

The final transformation matrix T is then used to modify both the right-hand side vector f and the stiffness matrix K, as given in Eq. (4).

Although sparsity of the system improves performance, the algorithm requires distribution across computational units for large matrices.

The inputs to the algorithm are: constraint equations describing dependency among DOFs (denoted by C), the stiffness matrix K, force (boundary condition) vector f, and gap (constraint offset) vector g. The output is the nodal displacement vector u that satisfies the modified system including the multi-freedom constraints.

To solve the system computationally, interdependent objects are designed and used, such as for storing constraint equations.

While optimized matrix transposition and multiplication methods are already found built in PETSc, parallelization focuses on generating the transformation matrix T and solving the modified system. Generating the transformation matrix is especially costly while working with very large systems, making constraint application process the bottleneck when a large number of constraints are present.

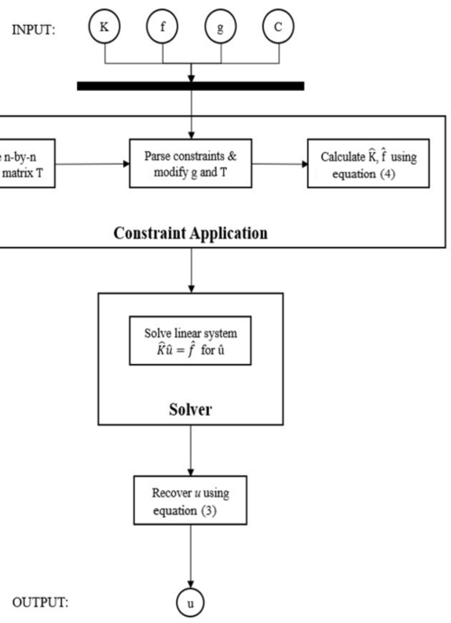
INPUT:
K
f
g
C
Create n-by-n identity matrix T
Parse constraints & modify g and T
Calculate $\hat{K}$, $\hat{f}$ using equation (4)
Constraint Application
Solve linear system $\hat{K}\hat{u} = \hat{f}$ for $\hat{u}$
Solver
Recover u using equation (3)
OUTPUT:
u

Figure 2: Proposed algorithm flowchart showing tasks & sub-tasks, all distributed among available processes

3 Benchmark Tests via Proposed Algorithm

The proposed algorithm is designed in C++ programming language. The computation time is aimed to be reduced using parallel processing. For this purpose, PETSc 3.23.1 is used as a high-performance computing toolkit for robust sparse matrix representation, as well as a higher-level wrapper for MPI (message passing interface) operations. The algorithm's syntax is designed for efficient memory handling. Parallel MPI library enables load balancing across processors and threads.

Sparse matrices for the benchmark problem demonstration are taked from the University of Florida Sparse Matrix Collection. For testing, symmetrical square matrices of structural origin are used in Matrix Market (.mtx) format. The benchmark problem focuses on applying a set of linear constraints to a sparse stiffness matrix in parallel using PETSc and solution of nodal displacement vector u. The stiffness matrix, stored in Matrix Market (.mtx) format, represents a global linear system derived from discretized physical models such as finite element meshes. Each constraint maps a "slave" degree of freedom to a weighted sum of "master" degrees of freedom with an optional constant gap. The constraints are generated randomly, subject to arbitrary bounds.

The sparse matrices used in benchmark tests are chosen to vary significantly in size, but not in sparsity ratio, to better observe the effect of size and constraint number on the computation time. The computation time across varying number of total processes are evaluated with respect to scalability using following metrics calculated for each test.

$$S_p = \frac{T_1}{T_p} \quad , \quad E_p = \frac{S_p}{p} \tag{5}$$

where T_1 and T_p is the computation time for 1 and p processes in order, S_p is the parallel speedup, and E_p is the parallel efficiency. In the post-processing stage, these metrics are compared with respect to the corresponding number of processes and constraints.

4 Results and Discussion

The sparse matrix-vector multiplication, i.e., solving the linear system, is efficiently handled by PETSc. The time complexity of solving the modified system is optimally bound by $\mathcal{O}(n)$, where n is the number of nonzero elements.

Results of two benchmark problems, namely Emilia_923 matrix and bmw7st_1 are presented in terms of computation time for MFCs application vs. MPI processes, as well as number of constraints in Figure 3 and Figure 4, respectively. Speedup achievements are presented in terms of surface plots in Figure 5 and Figure 6 for respective benchmark problem. Overall, significant speedup values (as high as six fold) are observed with large matrices (in the case of Emilia_923 and $\approx$ 1 million DOF) and increasing number of constraints (such as beyond 100000), while smaller problems (smaller matrix of bmw7st_1 on the order of 140000) and less number of constraint numbers exhibit less improvement in computation time with increasing number of cores used.

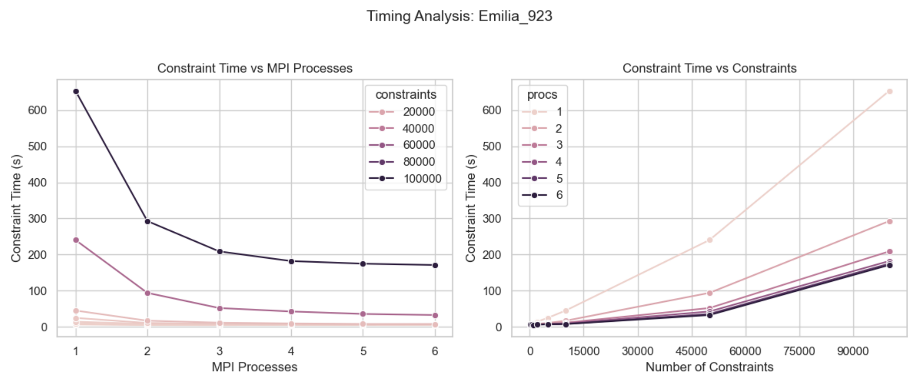

Figure 3: Proposed algorithm's performance analysis for benchmark problem of Emilia_923 matrix (size: 923136×923136): Constraint time vs. MPI processes for different number of constraints (left) and constraint time vs. number of constraints for different number of processes

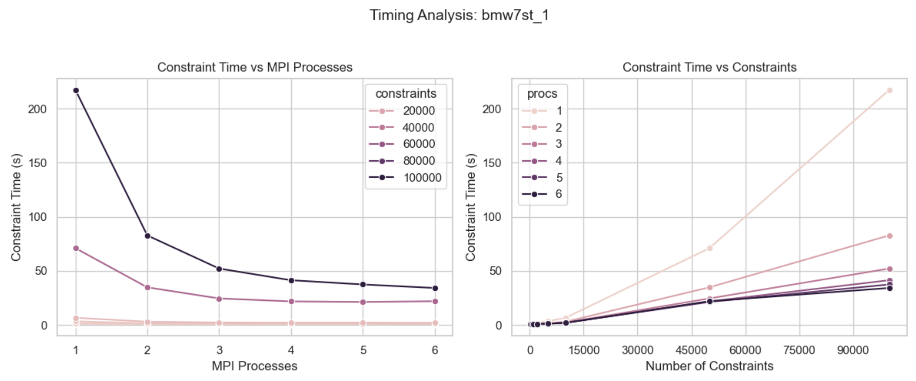

Figure 4: Proposed algorithm's performance analysis for benchmark problem of Emilia 923 matrix (size: 141347 × 141347): Constraint time vs. MPI processes for different number of constraints (left) and constraint time vs. number of constraints for different number of processes)

5 Conclusions and Future Work

In this work, we presented a parallel algorithm for solving linear equation systems with imposed MFCs. Although scalability remains a challenge, the overhead caused

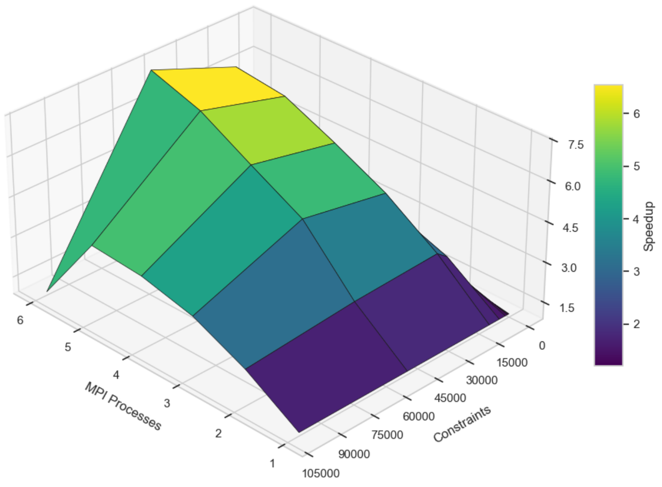

Figure 5: Speedup values for varying constraint numbers and processor counts (Matrix Size: 923136×923136)

by the initialization and communication costs appear to be compensated by the distributed computation load. Therefore, the parallel implementation is proven to increase computation efficiency. Since a single machine with 6 cores is used for the performance comparison of the presented algorithm applied to two benchmark problems, the capability of this benchmark is yet to be increased to run tests for bigger problems with the help of high-performance computing. These results thus serve as proof-of-concept rather than the demonstration of a fully optimized parallel solution.

Execution times for constraint application have been benchmarked in parallel settings. As expected, parallel implementation underperforms in smaller problems due to MPI overhead but shows promise for scaling in larger settings.

Future plans include retesting benchmarks using even larger matrices and on high-performance computing (HPC) clusters. In that case, the number of nodes inside the cluster setting can also be evaluated as a parameter. Ultimately, better load balancing, memory distribution strategies, and communication handling can further improve scalability and performance. Future runs on HPC machines will include multi-node execution for broader validation. The code is available at: `https://github.com/open-lattice/solver.git`

References

[1] J. J. Muñoz, G. Jelenić, and M. A. Crisfield, “Master–slave approach for the modelling of joints with dependent degrees of freedom in flexible mechanisms,” *Communications in Numerical Methods in Engineering*, vol. 19, pp. 689–702,

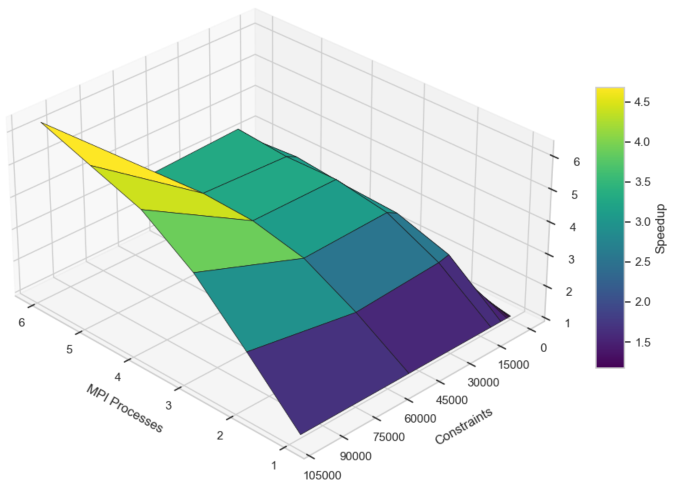

Figure 6: Speedup values for varying constraint numbers and processor counts (Matrix Size: 141347×141347)

Sep. 2003.

[2] J. A. Oliveira, J. Pinho-da-Cruz, A. Andrade-Campos, and F. Teixeira-Dias, "Stress- and Strain-based MultiFreedom Constraints for Periodic Media Optimisation," *2nd International Conference on Engineering Optimization*, Sep. 2010.

[3] Z. J. Zheng, S. Kulasegaram, P. Chen, and Y. Q. Chen, "An efficient SPH methodology for modelling mechanical characteristics of particulate composites," *Defence Technology*, vol. 17, pp. 135–146, Feb. 2021.

[4] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, Dec. 2011. DOI: `https://doi.org/10.1145/2049662.2049663`

[5] Belytschko T, Liu WK, Moran B, Elkhodary KI (2014) Nonlinear finite elements for continua and structures, 2nd edn. Wiley, Hoboken

[6] Boungard, J., Wackerfuß, J. Master–slave elimination scheme for arbitrary smooth nonlinear multi-point constraints. Comput Mech 74, 955–992 (2024). https://doi.org/10.1007/s00466-024-02463-7