

PARALLEL COMPUTATION ALGORITHMS FOR MULTIBODY DYNAMICS SIMULATIONS

A Thesis
Presented to
The Academic Faculty

by

SeunDo Heo

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
August 2017

Copyright © SeunDo Heo 2017

PARALLEL COMPUTATION ALGORITHMS FOR MULTIBODY DYNAMICS SIMULATIONS

Approved by:

Professor Olivier A. Bauchau, Advisor
Department of Aerospace Engineering
University of Maryland

Professor Dewey H. Hodges, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Professor Emeritus Kenneth M. Will
School of Civil and Environmental
Engineering
Georgia Institute of Technology

Professor Marilyn J. Smith
School of Aerospace Engineering
Georgia Institute of Technology

Professor Aldo A. Ferri
School of Mechanical Engineering
Georgia Institute of Technology

Professor George A. Kardomateas
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 4 May 2017

To
my parents,
Taeyee and Songhee

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Olivier A. Bauchau, my thesis advisor, for his guidance and advice. His guidance and advice on my research have been always clear and evident, and they have provided me with self-confidence which makes me stay on the right track. It has been a great opportunity that I could develop a multibody dynamics simulation program (Dymore) for my research under his guidance.

I truly thank Profs. Dewey H. Hodges and Jechiel I. Jagoda for their continuous support until my graduation. With their great support, I could have been allowed to focus on my research and to accomplish safely and successfully.

I was lucky to meet great Professors at Georgia Tech, especially, Olivier A. Bauchau, Dewey H. Hodges, Kenneth M. Will and Aldo A. Ferri in their challenging but fascinating lectures. I am grateful to my thesis committee members, Profs. Olivier A. Bauchau, Dewey H. Hodges, Kenneth M. Will, Marilyn J. Smith, Aldo A. Ferri and George A. Kardomateas for their valuable time, comments and suggestions. I am also grateful to Professors, Suresh Menon and Marilyn J. Smith, and my colleagues, Joachim Hodara and Brian Pogioli, who allowed and helped me to access clusters for my parallel research.

During my long study at Georgia Tech, I have spent a great time with lots of great office mates in Weber 213: Alexander Epple, Changkuan Ju, Jimmy Ho, Wei-En Li, Huimin Song, Anurag Rajagopal, Chen Wei, Pezhman Mardanpour, Hanif Hoseini and Mohit Gupta. I would like to acknowledge their kindness by taking this place. Especially I remember the pleasant time with Anurag for talking, movies and dinners at times. I also would like to thank all of my Korean friends at Georgia Tech for their friendship.

I cannot exaggerate the support from my family in South Korea and United States.

My only brother, Wondo, often called me and concerned together about my hard time. I sincerely thank my parents in law for their consistently kind heart and ever-cheerful words for me. My sister in law, Kukhee, always shares and understands my wife's happiness and concerns. My little one, Taeyee, continuously brings and fills pure happiness in my family. I don't know how to say thank you to my wife, Songhee. Her endless dedication is the first source of power for me to finish my graduate studies. Lastly, my parents. I cannot estimate how deep their dedication and support for my entire life are. Without their support, I couldn't have finished my PhD student life, and their support is the last source of power to achieve this end.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	x
LIST OF FIGURES	xii
SUMMARY	xvi
I INTRODUCTION	1
1.1 Why “Parallel Computing?”	1
1.1.1 Advanced design for rotor blades	1
1.1.2 Aeroelastic analysis	4
1.2 Why “Domain Decomposition” for Parallel Computing?	6
1.3 Parallel Computing for Finite Element Analysis	8
1.3.1 Vector processor	8
1.3.2 Multifrontal solver	9
1.3.3 Domain decomposition methods	10
1.4 Parallel Computing for Multibody Dynamics	11
1.5 Objectives	14
1.6 Chapter Overview	15
II DOMAIN DECOMPOSITION METHODS FOR FINITE ELEMENT ANALYSIS	17
2.1 Classification of Domain Decomposition Methods	18
2.1.1 Type of partitioning	19
2.1.2 Overlapping or non-overlapping subdomains	20
2.1.3 Design of interface	20
2.1.4 Solution method for subdomain problems	22
2.2 Non-overlapping Domain Decomposition Methods for Finite Element Analysis	23

2.3	Classical Substructuring Method	26
2.3.1	Condensation process for substructuring	28
2.3.2	Governing equations of substructuring	30
2.3.3	Global interface problem	31
2.4	FETI and Its Variants	32
2.5	FETI-1	32
2.5.1	Subdomains and constraints at interface	34
2.5.2	Governing equations	37
2.5.3	Global interface problem	39
2.5.4	Solution procedure for global coarse interface problem	40
2.5.5	Preconditioning	41
2.6	FETI-2	42
2.7	FETI-DP	43
2.7.1	New formulation with corner and remainder nodes	44
2.7.2	Governing equations	45
2.7.3	Global interface problem	47
2.8	Interface Problem: Classical Substructuring and FETIs	48
III DOMAIN DECOMPOSITION BY LOCALIZED LAGRANGE MULTIPLIER TECHNIQUE		51
3.1	Multibody Dynamics	51
3.1.1	Formulation of dynamics with constrained system	52
3.1.2	Scaling technique and augmented Lagrangian formulation	52
3.2	Domain Decomposition	54
3.2.1	Classical Lagrange multiplier technique	56
3.2.2	Localized Lagrange multiplier technique	57
3.3	Formulation of the Problem	59
3.3.1	Strain energy of the system	59
3.3.2	Potential of the externally applied loads	61
3.3.3	Potential of a typical constraint	61

3.3.4	Assembly procedure for the constraints	63
3.3.5	Governing equations	65
3.4	Physical Interpretation of the Proposed Approach	66
3.5	Solution Procedure with Factorization	68
3.5.1	Expected block matrices from LU factorization	69
3.5.2	Details of the factorization procedure	70
3.5.3	Details of the forward- and back-substitution procedures	71
3.5.4	Summary of the solution procedure	72
IV	PARALLEL IMPLEMENTATION	77
4.1	Parallel Finite Element Procedure	77
4.2	Domain Decomposition	79
4.2.1	Interface node detection	80
4.2.2	LLM element	80
4.2.3	Job distribution to processors	81
4.3	Finite Element Mesh Adjustment	82
4.4	Message Passing between Processors	83
4.5	Data Structures for Parallel Computations	85
4.6	Nonlinear Dynamic Analysis in Parallel	86
V	NUMERICAL EXPERIMENTS	92
5.1	Testbed for Parallel Simulations	92
5.2	Performance Assessment	93
5.2.1	CPU time	93
5.2.2	Speed-up	94
5.2.3	Efficiency	96
5.2.4	LU factorization phase	96
5.2.5	Inter-processor communication time	97
5.2.6	Contributions of sub-phases	98
5.3	Cantilever Beam	100

5.4	Grid of Beams	103
5.4.1	Model description	103
5.4.2	Domain decomposition	104
5.4.3	Predictions of dynamic response	105
5.4.4	Performance for solution procedure	107
5.4.5	Performance for LU factorization phase	115
5.4.6	Inter-processor communication time	117
5.4.7	Contributions of sub-phases	119
5.5	Rotor System	123
5.5.1	Model description	124
5.5.2	Domain decomposition	125
5.5.3	Predictions of dynamic response	128
5.5.4	Performance for solution procedure	132
5.5.5	Performance for LU factorization phase	136
5.5.6	Inter-processor communication time	141
5.5.7	Contributions of sub-phases	141
VI	CONCLUSIONS AND FUTURE WORK	146
6.1	Conclusions	146
6.2	Main Contributions of the Thesis	149
6.3	Future Work	150
	REFERENCES	151
	VITA	157

LIST OF TABLES

1.1	Computational cost estimation: Beam vs. Shell	3
1.2	3-D finite element problem size estimation for composite rotor blade	4
2.1	Comparison of interface matrices between classical substructuring (CS) and FETIs	49
2.2	Comparison of interface right-hand-side arrays between classical substructuring (CS) and FETIs	50
3.1	Phases of solution procedure	76
4.1	Various MPI commands	85
5.1	Parallel simulation cases	93
5.2	Performance indices	95
5.3	Significant sub-phases for execution time in solution procedure	99
5.4	Cantilever beam: Problem sizes for domain decomposition	101
5.5	Cantilever beam: Tip displacement	101
5.6	Cantilever beam: Convergence norms over 3 iterations	102
5.7	Grid of beams: Structural properties of a beam segment	103
5.8	Grid of beams: Problem sizes for 6 domain decomposition cases	104
5.9	Grid of beams (for one iteration with factorization): CPU time, speed-up and efficiency for solution procedure	110
5.10	Grid of beams (for the entire simulation): CPU time, speed-up and efficiency for solution procedure	110
5.11	Grid of beams (for one iteration with factorization): Ratio of CPU time to SPSD case, speed-up and efficiency	112
5.12	Grid of beams (for the entire simulation): Ratio of CPU time to SPSD case, speed-up and efficiency	112
5.13	Grid of beams: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for a subdomain problem	115
5.14	Grid of beams: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for an interface problem	115
5.15	Grid of beams (MPMD, for the entire simulation): LU factorization CPU time and its percentage of the total simulation time	116

5.16	Grid of beams (MPMD, for the entire simulation): Dofs for all LLM elements, total MPI data size and inter-processor communication time	118
5.17	Rotor system: Problem sizes for 6 domain decomposition cases	126
5.18	Rotor system (for one iteration with factorization): CPU time, speed-up and efficiency for solution procedure	133
5.19	Rotor system (for the entire simulation): CPU time, speed-up and efficiency for solution procedure	134
5.20	Rotor system (for one iteration with factorization): Ratio of CPU time to SPSD case and speed-up	134
5.21	Rotor system (for the entire simulation): Ratio of CPU time to SPSD case and speed-up	136
5.22	Rotor system: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for a subdomain problem . . .	136
5.23	Rotor system: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for an interface problem	139
5.24	Rotor system (MPMD, for the entire simulation): LU factorization CPU time and its percentage of the total simulation time	139
5.25	Rotor system (MPMD, for the entire simulation): Dofs for all LLM elements, total MPI data size and inter-processor communication time	141

LIST OF FIGURES

1.1	Advanced rotor blade configurations	1
1.2	Rotor blade modeling with beam elements or shell elements	2
1.3	Typical rotor blade section	3
2.1	Relationship between topological entities, geometric components and structural elements (Source: Dymore User’s Manual [7])	18
2.2	Types of partitioning for edge-based elements (Top: Element-based partitioning; Bottom: Vertex-based partitioning)	19
2.3	Overlapping domain decomposition	20
2.4	Various interface connections (Subscript: Local node number within a subdomain; Superscript in parentheses: Subdomain number; Other superscripts: Global interface node number)	21
2.5	Domain decomposition	24
2.6	Substructuring of an aircraft (Source: Course material for Introduction to Finite Element Methods [36])	26
2.7	Mesh of classical substructuring	27
2.8	Detached-subdomain version of mesh	28
2.9	Condensed-out mesh with respect to global interface nodes	29
2.10	FETI-1 domain decomposition method	33
2.11	Mesh of FETI-1 domain decomposition	34
2.12	FETI-DP corner and remainder nodes	44
2.13	FETI-DP domain decomposition mesh	45
2.14	FETI-DP domain relationship	47
3.1	Planar solid partitioned into four non-overlapping subdomains	55
3.2	Classical and localized Lagrange multipliers.	56
3.3	Connection through localized Lagrange multipliers.	58
3.4	CLM method vs LLM method	60
3.5	LLM domain decomposition mesh	64
3.6	LLM domain relationship	65

3.7	Physical interpretation of the penalty terms.	66
3.8	Physical interpretation of the Lagrange multipliers	67
4.1	Finite element procedures (Left: Sequential; Right: Parallel)	78
4.2	Schematic of domain decomposition process	89
4.3	Job distribution to processors in master-slave framework	90
4.4	Hierarchy of parallel data structures	91
5.1	Cantilever beam: Partitioned into three subdomains	100
5.2	Grid of beams	103
5.3	Grid of beams: Partitioning of grids	105
5.4	Grid of beams: Problem sizes for 6 domain decomposition cases	106
5.5	Grid of beams: Time history of loading	107
5.6	Grid of beams: Predictions of translational responses (Left) and their differences (Right) between two cases: Single domain vs. 32 subdomains	108
5.7	Grid of beams: Predictions of rotational responses (Left) and their differences (Right) between two cases: Single domain vs. 32 subdomains	109
5.8	Grid of beams: CPU time for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	111
5.9	Grid of beams: Speed-up or ratio of CPU time to SPMD case for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	113
5.10	Grid of beams: Efficiency for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	114
5.11	Grid of beams (MPMD): LU factorization performance (Top: Ratio of performance to single domain case for a subdomain problem; Bottom: Ratio of performance to 2-subdomain case for an interface problem)	117
5.12	Grid of beams (MPMD): Inter-processor communication time (Top: for one iteration with factorization; Bottom: for the entire simulation)	119
5.13	Grid of beams (SPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution)	121

5.14	Grid of beams (MPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution; MPI Wait: Processor synchronization; MPI Comm: Inter-processor communication) .	122
5.15	Rotor system: Four-bladed hingeless flexbeam rotor system under a driving torque at the hub	123
5.16	Rotor system: Blade sectional properties	124
5.17	Rotor system: Domain decomposition	126
5.18	Rotor system: 2^N partitioning of each rotor blade	127
5.19	Rotor system: Problem sizes for 6 domain decomposition cases	127
5.20	Rotor system: Time history of driving torque	128
5.21	Rotor system: Predictions of in-plane translational (along \bar{v}_2) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains	129
5.22	Rotor system: Predictions of in-plane rotational (about \bar{v}_3) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains	130
5.23	Rotor system: Predictions of out-of-plane translational (along \bar{v}_3) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains	131
5.24	Rotor system: Predictions of out-of-plane rotational (about \bar{v}_2) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains	132
5.25	Rotor system: CPU time for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	135
5.26	Rotor system: Speed-up or ratio of CPU time to SPSD case for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	137
5.27	Rotor system: Efficiency for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)	138
5.28	Rotor system (MPMD): LU factorization performance (Top: Ratio of performance to single domain case for a subdomain problem; Bottom: Ratio of performance to 9-subdomain case for an interface problem) .	140

5.29	Rotor system (MPMD): Inter-processor communication time (Top: for one iteration with factorization; Bottom: for the entire simulation) . .	142
5.30	Rotor system (SPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution)	144
5.31	Rotor system (MPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution; MPI Wait: Processor synchronization; MPI Comm: Inter-processor communication) .	145

SUMMARY

Flexible multibody dynamics simulations have been performed sequentially on a single processor because the problem sizes for the simulations were not large. However, advanced designs of rotor blades or CSD/CFD (Computational Structural/Fluid Dynamics) coupled problems call for more stringent accuracy requirements and faster computations in multibody dynamics simulations.

For parallel computations, a novel non-overlapping domain decomposition method is developed and implemented to perform flexible multibody dynamics simulations in parallel. Non-overlapping domain decomposition methods such as classical substructuring methods and finite element tearing and interconnecting (FETI) methods are also reviewed and compared to see how they have been developed and improved for better domain decomposition.

The proposed domain decomposition approach with a localized version of Lagrange multiplier technique and an augmented Lagrangian formulation in conjunction with the Lagrange multipliers, is formulated and discussed in detail. Within the framework of direct solvers, the solution procedure with LU factorization and forward and backward substitutions has been designed for parallel computations. The actual implementation of the parallel algorithm with the domain decomposition method on a finite-element-based multibody dynamics simulation program (Dymore), is also described.

Finally, the parallel algorithm is tested on parallel hardware with numerical experiments to evaluate the accuracy and scalability of the algorithm for various domain decomposition cases.

CHAPTER I

INTRODUCTION

1.1 Why “Parallel Computing?”

Today, multi-processors for parallel processing are everywhere and not expensive anymore. They have been already used to solve large-scale engineering problems. Flexible multibody dynamics problems can become another candidate that requires parallel processing.

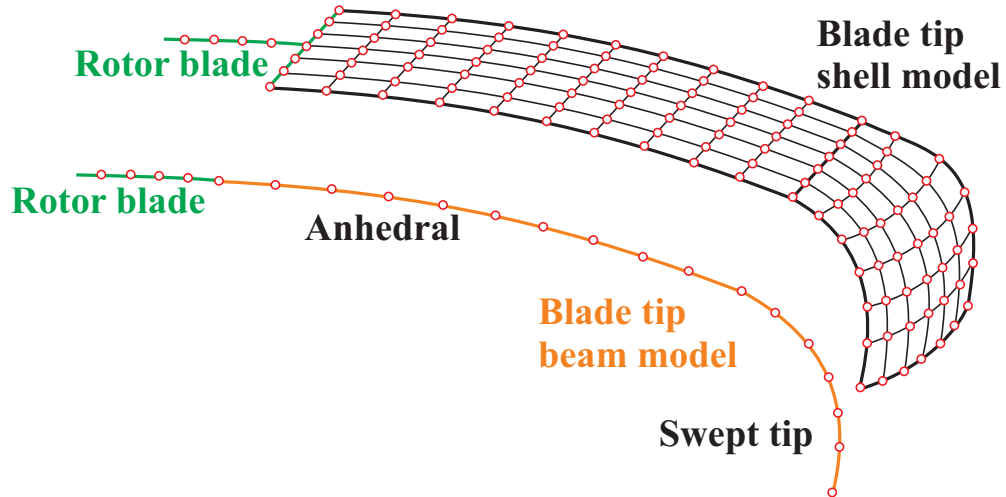


Figure 1.1: Advanced rotor blade configurations

1.1.1 Advanced design for rotor blades

Traditionally, the structural dynamics simulation of rotor systems has relied on beam models for the analysis of blades. Furthermore, to simplify the process, modal reduction is typically performed to further reduce the computational complexity of the problem [8]. In the past decade, geometrically exact beam models have been developed even for composite beams [44, 10, 16], enabling the fully nonlinear analysis of realistic rotor systems [9]. These models are now used by industry because they meet

their ever more stringent accuracy requirements.

To improve aerodynamic performance, industry is considering the use of curved blades presenting sweep and anhedral [47]. The accurate analysis of such structures calls for geometrically exact shell models [64, 65] rather than their geometrically exact beam counterparts. The shell models may provide a better environment for meshing at the interface between the rotor blade surface and the air stream over the surface. Figure 1.1 illustrates these considerations for the advanced blade design in rotorcraft industry.

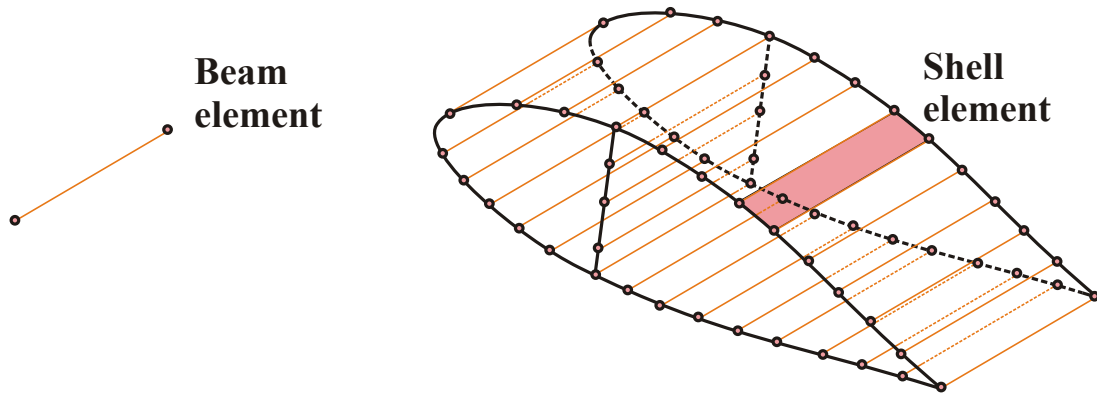


Figure 1.2: Rotor blade modeling with beam elements or shell elements

When it comes to the computational costs, the difference between beam and shell modelings is remarkable, see Figure 1.2. The most expensive operation of the solution procedure with finite element analysis is the factorization of the stiffness matrix. Since the finite element discretization mostly creates a sparse stiffness matrix, the *average or mean bandwidth* of the matrix is used to represent the sparsity of the matrix. The factorization cost of a stiffness matrix is proportional to the square of the mean bandwidth of the matrix.

A simple estimation of the factorization cost can be done. For the beam modeling, a quadratic beam element is used for rotor blade modeling so that the degrees of freedom (dofs) will be 18 and the mean bandwidth will be the same size. For the shell modeling, two different mesh cases are considered. Thus, the mean bandwidths

for the coarse and fine mesh cases will be 480 and 960, respectively.

Table 1.1: Computational cost estimation: Beam vs. Shell

Element type	Number of elements	Mean bandwidth	Cost ratio
Beam	1	18	$(18/18)^2 = 1$
Shell (Coarse mesh)	32	480	$(480/18)^2 = 711.1$
Shell (Fine mesh)	64	960	$(960/18)^2 = 2844.4$

As can be seen in Table 1.1, the difference between the factorization costs of the beam and shell elements is huge in that it can be simply said that one hour run with the beam model becomes 2,800 hour or 120 day run with the shell model for the fine mesh. This significant difference between the factorization costs with the beam and shell modelings is one of the main reason calling for parallel computations.

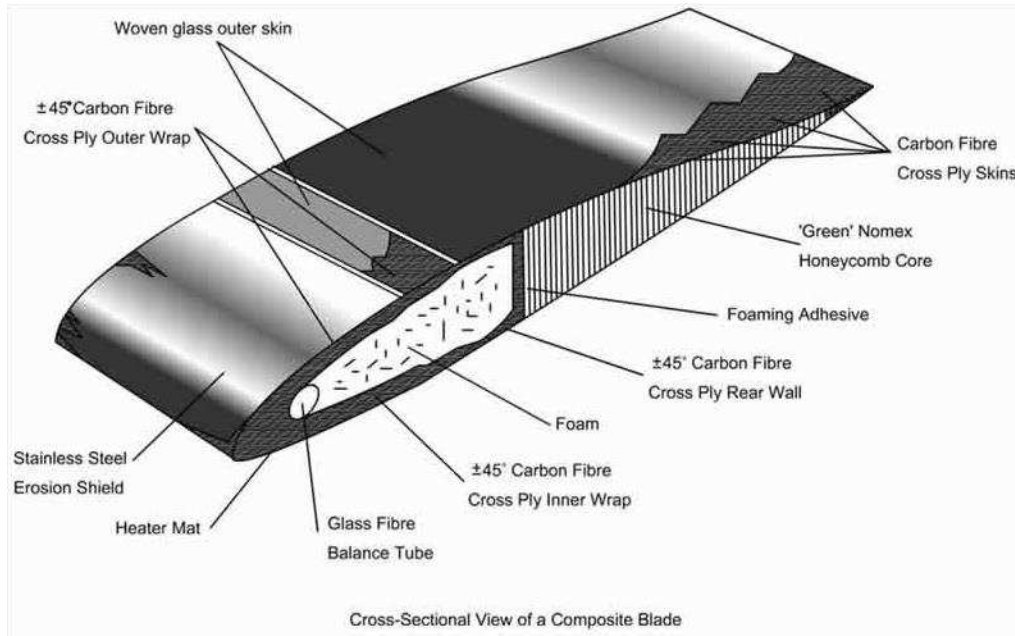


Figure 1.3: Typical rotor blade section

Most rotor blades are made of composites as seen in Figure 1.3. If the composite rotor blade of the UH-60 helicopter is modeled with 3-D elements such as the 20-node brick element, it can be meshed with a number of the brick elements. It is assumed

Table 1.2: 3-D finite element problem size estimation for composite rotor blade

Rotor blade dimensions	L = 8.6 m; Chord = 0.72 m
Main D-spar	60 plies; Thickness/ply = 125 μm
Element type	20-node brick
Element/ply	1 (assume 1/10 aspect ratio)
Elements (length \times width \times height)	6,880 \times 1,152 \times 60 \approx 476 million
Nodes (length \times width \times height)	12,760 \times 2,304 \times 120 \approx 3.5 billion
Dofs (total nodes \times dofs/node)	3527884800 \times 3 \approx 10.6 billion

that as a rough estimate, at least, one element per ply is used to capture the inter-laminar stress and the aspect ratio of the element is set to 1/10 within a reasonable range where an aspect ratio is defined as the ratio of the longest mesh edge divided by the shortest mesh edge. The dofs of the problem with 3-D finite elements becomes over 10 billion which is way beyond today's computer capabilities. In fact, only the 12 dofs or modes were considered to design the UH-60 helicopter. This estimation is tabulated in Table 1.2.

1.1.2 Aeroelastic analysis

Over the past decade, aeroelastic analysis codes have moved in a novel direction: CFD (Computational Fluid Dynamics) codes have been linked to Flexible Multibody Dynamics codes using various fluid-structure coupling techniques to provide a fundamentally new aeroelastic analysis approach that will change the way helicopters are designed [67, 68, 4, 53]. The CFD codes provide three-dimensional solutions of Navier-Stokes equations using the massive computational capabilities of parallel computers. In contrast, comprehensive rotorcraft codes solve geometrically exact beam problems on single processor machines.

When beam elements are used for the structural modeling, the computational cost of the CFD analysis outweighs that of the CSD (Computational Structural Dynamics) analysis by several orders of magnitude. Nevertheless, as parallel computers become more powerful and more widely available, the wall-clock time required by the

CFD analysis keeps dropping, whereas that required by the CSD on single processor machines remains constant or increases as the complexity of rotor systems increases. When the loose coupling strategy is used, the CFD computations are run for a few rotor revolutions only. On the other hand, comprehensive simulation codes are used to trim the rotor, requiring the simulation a far larger number of rotor revolutions. Consequently, the wall-clock times used by the parallel CFD computations and the serial CSD simulations are bound to become similar as machines with an ever increasing number of cores become available.

Clearly, the development of parallel computational schemes for rotor dynamics is a timely research topic. First, the present computations using geometrically exact beam models will be considerably quickened, helping speed-up coupled CSD/CFD simulations. Second, comprehensive modeling tools using more traditional aerodynamics, such as lifting line and table look-up procedures, will benefit from this technology greatly. Indeed, comprehensive rotorcraft analysis based on fully nonlinear beam models will require as little as a few minutes of wall-clock time in a parallel computational environment. Finally, the development of parallel computational tools for structural dynamics is an enabling technology for future rotor development. Indeed, advanced rotor concepts, such as those involving curved blades with sweep and anhedral, will require the use of shell models for better meshing at the interface between the blade surface and the air stream. On the other hand, despite of the difficulties in the application of 3-D finite elements to the design of composite rotor blades, the 3-D elements can still be used for more reliable analysis in locally developed stress concentration. Performance of all these modeling approaches such as geometrically exact beam/shell modeling and 3-D finite element modeling, can be improved with parallel algorithms, and hence, they can be all candidates in the development path to the parallel computing for the advanced design for rotor blades in conjunction with the aeroelastic analysis, although shell models and 3-D finite elements are not

investigated in this thesis. The routine use of these models is not practical with today's comprehensive codes running on single processor machines. The availability of parallel computational tools for rotor dynamics will give new impetus to the field, open new areas of research, allow higher-accuracy predictions than possible before, provide far more detailed description of the three-dimensional stress distributions in rotors, and enable designers to simulate novel rotor configurations confidently.

1.2 Why “Domain Decomposition” for Parallel Computing?

There are many ways to perform parallel computations. Parallel computations can be achieved by either computer hardware-based approach or software-based counterpart. The hardware-based approach such as pipelining of CPU, vector processors, etc., usually falls in the category of fine-grained parallelism which requires frequent inter-processor communications and is not suitable for large-scale engineering problems. On the other hand, the software-based approaches or parallel algorithms are mostly in the category of coarse-grained parallelism. This parallelism is basically accompanied by a decomposition of a task. The decomposition approach can be largely classified into functional and domain decompositions. The functional decomposition divides the algorithm of a program into parts while the domain decomposition divides the data used in a program into parts. Since a finite element analysis program deals with a huge data of model geometry and physical properties, the domain decomposition approach is more suitable than the functional counterpart because the functional decomposition for the finite element analysis could bring about severe load imbalance.

The domain decomposition method has many advantages. Unlike the ready-to-plug-in parallel algorithms, the domain decomposition algorithm for parallel computations can be tailored based on the physical characteristics of a given problem, in order to get the optimal approach to the more accurate numerical solution. For example, the analysis of fluid-structure interaction problem requires that the different

governing equations be applied to the two physically different fields. The domain decomposition method can be applied to this type of problem with the proper domain partitioning along the boundary between the fluid and structure fields. If the parallel computations proceed without the consideration about the physical difference between the heterogeneous fields, it would be questionable whether the resulting parallel solution is acceptable and accurate.

The domain decomposition method is also based on the *divide-and-conquer* concept which is the most fundamental approach to make any problem easier to solve. The concept is able to dramatically reduce the size of a problem through partitioning the problem and the partitioned subproblems would be much easier to handle. An engineering problem with a complex geometry can be divided into subproblems with simpler geometry. When the topology of a model can be expressed in a graph where vertices, edges and faces are used to define the topology of the model, the domain decomposition can be optimized by a graph partitioner to achieve optimal load balancing.

The rotor system which is the most representative model of the finite-element-based multibody systems, has a complex geometry and various mechanical components. The mechanical components can be partitioned into subproblems, which is the domain decomposition. Any challenging problem like the rotor system can be divided into subproblems, and they can be solved in parallel. The dynamic analysis of the rotor system is also often performed with a fluid-structure interaction modeling, which is a multidisciplinary problem with physically heterogeneous fields, and this can be handled by the domain decomposition method as previously stated. All in all, the domain decomposition approach is one of the most suitable approaches, especially for the engineering problems of finite-element-based multibody systems.

1.3 Parallel Computing for Finite Element Analysis

The finite element method is one of most widely used approaches for computational mechanics and it originated from structural mechanics. The method basically subdivides a problem into smaller subproblems and each subproblem is called a *finite element*. Each finite element can represent a partial solution to an original global problem. In finite element procedure, the global stiffness matrix is assembled from element stiffness matrices of local finite elements. Since the element stiffness matrices are added up in a global stiffness matrix with a proper mesh connectivity between each finite element and the global structure, it is clear that each element stiffness matrix can be computed in parallel. Hence, the underlying idea of the finite element method is naturally applicable to parallel computation algorithms because they share the same idea which is the decomposition of an original problem.

The most expensive operation of the finite element procedure is the factorization of the stiffness matrix specifically for direct solvers. For a sequential direct solver, the execution time for only matrix factorization takes about 50 ~ 60% in an iteration for typical structure problems. In order to reduce the execution time of the expensive operation, several parallel approaches have been developed and used for the finite element analysis.

1.3.1 Vector processor

A *vector* processor or *array* processor is a hardware-based approach to parallel computations. Many common supercomputers since 1970's have been designed by the vector processors. The well-known *Cray* computer is a supercomputer with vector processors.

As a vector usually means a linear array of numbers *i.e.*, multiple data, a vector processor performs an instruction (operation) on multiple data at once, in contrast to a scalar processor which does on only a single datum. This parallel approach

is referred to as SIMD (Single Instruction Multiple Data) and is considered a data parallelism. If a vector processor accesses memory for massive data simultaneously, it takes dramatically less processing time than a scalar processor does. The vector processor, however, takes more execution time to execute an instruction on a single data than a scalar processor, because the vector processor is designed to handle only vectorized multiple data accompanying large overhead. Moreover, vectorization of data is required for the efficient usage of vector processors; this increases the complexity of a computer code. Most finite element codes use various data structures for program efficiency while vector processors can maximize the efficiency with only vector-like data structures.

Today, *General-Purpose Graphics Processing Unit* (GPGPU), which is based on vector processors, has gained growing popularity because of its reasonable price in the market, unlike the expensive vector processors which were rarely used for personal computers in the past. The research about the parallel computations with efficient usage of GPGPU has been actively performed [54]. GPGPU has a strong potential to enhance the parallel performance of finite element codes.

1.3.2 Multifrontal solver

One of the popular parallel approaches for finite element analysis is the *multifrontal* method [21] which is a parallel version of the *frontal* method [45]. The frontal method basically utilizes the sparsity so that the solution procedure can effectively minimize the computations associated with zero entries like the other direct solvers for sparse matrices in the form of skyline or banded structure. Briefly speaking of the idea about the frontal solver, as a boundary along a set of elements or *a front* advances, a process of variable elimination is performed similarly to the condensation process. The original frontal method not only assembles element matrices and arrays but also eliminates variables at the same time in order to avoid constructing a full global

stiffness matrix which becomes huge to handle for large-scale problems. Rather, the method keeps the problem size small enough to fit in the core storage and performs the solution procedure faster. The original frontal method has been improved to use multi-processors for multiple fronts so that the elimination process can be performed in parallel. This improved version is the multifrontal solver. The multifrontal solvers have been studied and developed extensively until today [46, 43, 52].

1.3.3 Domain decomposition methods

A domain decomposition method is an approach to solve boundary value problems by partitioning a computational domain into a desired number of *subdomains*. The partitioned subdomains can be either overlapping or non-overlapping with each other depending on the type of domain decomposition methods. The domain between different subdomains is called an *interface*. Briefly speaking, the domain decomposition method with overlapping subdomains doesn't need to solve a separate interface problem, but instead solves subdomain problems including the interface, and updates the boundary solution iteratively. In contrast, the non-overlapping-subdomain domain decomposition methods solve the interface problem separately. The domain decomposition methods originated from a paper by Schwarz, who proposed an alternating approach to solve a problem with complicated domain which includes a rectangle and a circle as shown in Figure 2.3 [63]. Many domain decomposition approaches have been studied, developed and investigated until today: additive and multiplicative Schwarz methods are overlapping approaches, while iterative substructuring methods such as Balancing Domain Decomposition methods (BDD), Finite Element Tearing and Interconnecting methods (FETI), and Mortar methods are non-overlapping approaches [48, 42].

1.4 *Parallel Computing for Multibody Dynamics*

Efficient algorithms have been developed for modeling multi rigid body dynamical systems presenting chain or tree-like topologies. By taking advantage of specific system topologies, parallel $\mathcal{O}(\log(n))$ formulations have been developed by Fijani *et al.* [37]. Anderson [1] developed an $\mathcal{O}(n)$ parallel algorithm for the simulation of systems presenting tree topologies. Featherstone proposed a divide-and-conquer parallel $\mathcal{O}(\log(n))$ strategy for articulated bodies [34], which he then generalized to tree and loop configurations [35]. Anderson and Duan [2] considered general systems of rigid bodies, which may contain arbitrary joint types, multiple branches, and/or closed loops. Their proposed scheme allows a substantially higher degree of parallelization than is generally obtainable using the more conventional recursive $\mathcal{O}(n)$ procedures. Critchley and Anderson [20] further refined the approach by introducing a new recursive coordinate reduction parallel algorithm that provides improved computation performance.

In recent years, the field of flexible multibody dynamics has embraced the finite element approach for modeling flexible structures, as described in the textbook of Géradin and Cardona [39] and numerous references therein. In those formulations, the kinematic constraints found in multibody systems are typically enforced via the Lagrange multiplier technique, leading to differential algebraic governing equations. In most cases, complex flexible multibody systems do not present the tree-like topology that characterizes traditional multi rigid body dynamical systems. For instance, when modeling a shell structure, all degrees of freedom are coupled through the two-dimensional finite element mesh. On the other hand, the finite element modeling of flexible multibody systems involves orders of magnitude more degrees of freedom than those typically involved in the modeling of multi rigid body systems. Consequently, detailed and accurate time simulation of flexible multibody systems using the finite element approach can typically not be performed in real time. The use of a finite

element approach, however, does not preclude the development of parallel solution algorithms.

In fact, a voluminous body of literature deals with the development and implementation of parallel computational strategies for finite element models. A complete review of the field is beyond the scope of this thesis, but the many approaches that have been proposed fall into the following categories.

First, because one of the most expensive operations in a finite element simulation is the factorization of the stiffness matrix, strategies for parallel implementation of factorization algorithms have been developed. The use of vector computers was proposed to speed up the factorization algorithm. This approach is very robust, but seems to have met limited success in terms of scalability [18]. Multi-frontal solvers [21] have also been proposed for the task of factorization [46]. This approach seems to be very arduous to implement; furthermore, for multibody systems, the presence of Lagrange multipliers as additional solution variables would require the development of appropriate multi-frontal solvers. This helps explain why this approach does not seem to have been applied to the types of problems appearing in flexible multibody dynamics.

The second approach relies on iterative solvers, such as the conjugate gradient or generalized minimal residual algorithms [62]. This approach has been applied to the solution of structural dynamics problems. It must be noted, however, that their efficiency crucially depends on system conditioning. Hence, good pre-conditioners are required for the efficient implementation of this approach. Flexible multibody systems often involve rigid-body modes associated with zero frequencies, and the Lagrange multipliers used to enforce kinematic constraints introduce numerous “infinite frequencies.” Consequently, flexible multibody problems are typically ill-conditioned, more so than their structural dynamics counterparts, and the use of iterative solvers does not seem to be desirable.

Finally, domain decomposition techniques have been the approach of choice for the last two decades. In particular, the FETI method developed by Farhat and coworkers has received considerable attention [29, 30, 25, 24]. A distinctive feature of this approach is that an auxiliary problem appears as a byproduct of the solution process. It is based on the rigid-body modes of the floating sub-domains, and because its size is small compared to that of the overall problem, it is called the “coarse problem.” For plate and shell problems, imposing the continuity of the transverse displacement at sub-domain cross-points, also called “corner points,” is found to be beneficial and is enforced via an additional set of Lagrange multipliers. This leads to a two-level procedure [31, 32, 23, 26]. Finally, the second generation, dual-primal FETI method [50, 28, 27, 33, 12] combines many of the techniques developed earlier in a unified manner.

Limited work has been done to apply the algorithms described the previous paragraphs to flexible multibody systems. Chiou *et al.* [17] combined the null space formulation with a preconditioned conjugate gradient algorithm to obtain a natural partitioning scheme for multibody systems. Coulon *et al.* [19] investigated the application of the FETI method to flexible multibody systems. Finally, Quaranta [58] used a domain decomposition method based on the Schur complement matrix to perform parallel simulation of multibody systems.

The FETI method is an approach by which the computational domain is divided into non-overlapping sub-domains. At the interface between the sub-domains, kinematic constraints are imposed to enforce the continuity of the displacement field over the entire structure. These kinematic constraints are enforced via fields of Lagrange multipliers that act at the interface between the sub-domains and can be interpreted as the interface connection forces. The method then proceeds in two steps. First, an interface problem is solved that yields the Lagrange multipliers. Second, the sub-domains are now independent structures subjected to known boundary forces, the

fields of Lagrange multipliers. The displacement fields in each sub-domain can be computed in parallel because each sub-domain is now independent of the others.

In the literature cited above, the FETI method is seen as a purely computational algorithm, and researchers have focused on the efficiency and scalability of the approach. In this paper, domain decomposition methods are investigated from the viewpoint of constrained dynamical system. Once the system is decomposed into sub-domains for the purpose of parallel computation, these sub-domains can be viewed as flexible bodies connected by kinematic constraints, as is typically found in flexible multibody systems. Clearly, the tools and techniques developed for the analysis of such systems become relevant. In rigid and flexible multibody dynamics, the method used to enforce of the kinematic constraints plays a pivotal role in the formulation. Bauchau and Laulusa [49, 11] have reviewed the literature on this topic. Some widely used techniques are not desirable in this case. For instance, many approaches seek a minimum set of variables by eliminating the Lagrange multipliers from the formulation. While this might be a computationally efficient way to proceed for sequential processing, it is not necessarily required for parallel computations. On the other hand, the augmented Lagrangian approach is a well established procedure for solving constrained dynamical systems and seems to be very much applicable to the problem at hand.

1.5 Objectives

Multibody systems are composed of various mechanical components. The components are often connected by mechanical joints which are modeled by constraint elements. A constraint element enforces kinematic constraints via Lagrange multipliers. The Lagrange multiplier technique is a routinely used scheme for the analysis of multibody systems and has already been extensively used in the multibody dynamics community. The technique has been continuously developed and demonstrated by the community

for the robust multibody dynamics simulations.

While domain decomposition methods have been developed for finite element analysis, the methods especially for multibody dynamics have not been studied much. But it becomes clear that the schemes routinely used for multibody dynamics simulations can also be applied for a new approach to the domain decomposition methods because the Lagrange multiplier techniques used to enforce kinematic constraints between decomposed subdomains are also one of the underlying schemes used to connect various mechanical components of a multibody system. The Lagrange multiplier techniques used in the multibody community are already robust in static and dynamic simulations. Therefore, it will be guaranteed that the application of the technique to the domain decomposition is also powerful for both simulations.

In this thesis, for the development of a novel non-overlapping domain decomposition method, the Lagrange multiplier technique to ensure the continuity between subdomains will be modified so that the Lagrange multipliers will not belong to the global interface, but rather will be localized into subdomains. Due to the characteristics of multibody systems, sparse direct solvers will be used for the solution procedure within the standard finite element procedure. Parallel processing for the solution procedure will be implemented with a message passing interface (MPI). The static and dynamic simulations with parallel computations will be performed for numerical experiments to validate the accuracy and scalability of the proposed domain decomposition method on parallel hardware.

1.6 Chapter Overview

In Chapter 2, the domain decomposition methods developed for finite element analysis are discussed. They are classical substructuring methods, Finite Element Tearing and Interconnecting method and its variants. In Chapter 3, the proposed domain decomposition method especially for the finite-element-based multibody dynamic analysis,

is formulated and discussed. The application of the domain decomposition method to the solution procedure within the finite element framework is also detailed. In Chapter 4, the implementation of the parallel algorithm with the proposed domain decomposition method will be described. Implementation of the domain decomposition and inter-processor communication will be detailed. In Chapter 5, numerical experiments with three example modes will be presented and studied to validate the parallel algorithm with the proposed domain decomposition method. Finally, the conclusions of this thesis and the possible future work will be discussed in Chapter 6.

CHAPTER II

DOMAIN DECOMPOSITION METHODS FOR FINITE ELEMENT ANALYSIS

A domain decomposition method is a method that partitions a computational domain of a given problem into multiple subdomains in order to gain several advantages. First, subdomain problems can be computed in parallel *i.e.*, subproblems can be solved simultaneously. Second, domain decomposition can transform complex geometry of an original problem into much simpler geometry of subdivided problems. Third, different formulas can be applied to physically heterogeneous fields in a multiphysics problem where each subdomain problem is governed solely by a single formula with a proper domain decomposition along the exact boundary between heterogeneous fields. Four, if an original problem is too large to fit into computer memory, this issue can be resolved by splitting the problem into smaller problems. Last but not least, the *divide-and-conquer* concept mostly reduces the total computational cost which is usually higher if an original problem is not divided and is solved as a whole. With these advantages, domain decomposition methods have been one of the most popular approaches for finite element analysis with parallel algorithms.

The application spectrum of domain decomposition methods is very wide. In this chapter, however, the application area of the methods is focused on the finite element analysis. Domain decomposition can also be applied to either spatial or time domain. But the domain decompositions in this thesis are all decompositions in the spatial domain associated with finite element discretization.

In this chapter, how to classify each domain decomposition method is presented. Then historical non-overlapping domain decomposition methods will be covered to

study the root of the proposed domain decomposition approach with the comparison among the methods.

2.1 Classification of Domain Decomposition Methods





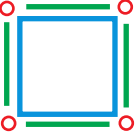
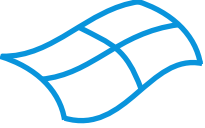
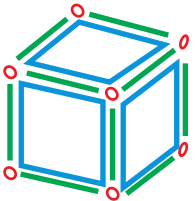

Topological entities	Geometric components	Structural elements
Vertex 	Point (x, y, z) 	Point mass Boundary condition
Edge 	Curve 	Beam, Cable Rigid bar Constraint elements Boundary condition
Face 	Surface 	2D elasticity Membrane Shell Boundary condition
Block 	Volume 	3D elasticity "brick elements"

Figure 2.1: Relationship between topological entities, geometric components and structural elements (Source: Dymore User's Manual [7])

In general, a domain decomposition method generates two types of computationally artificial problems: subdomain and interface problems. Through the domain decomposition, the original problem is divided into smaller problems and each problem is now a local problem. The interface problem is a byproduct of partitioning and is placed along the boundary of adjacent subdomains. The interface problem is always coupled with local subdomain problems and is solved with data transfer between subdomains and the interface. Saad says in his textbook [61], a domain decomposition method can be classified by four factors: type of partitioning, subdomain

overlapping, interface design and subdomain solution.

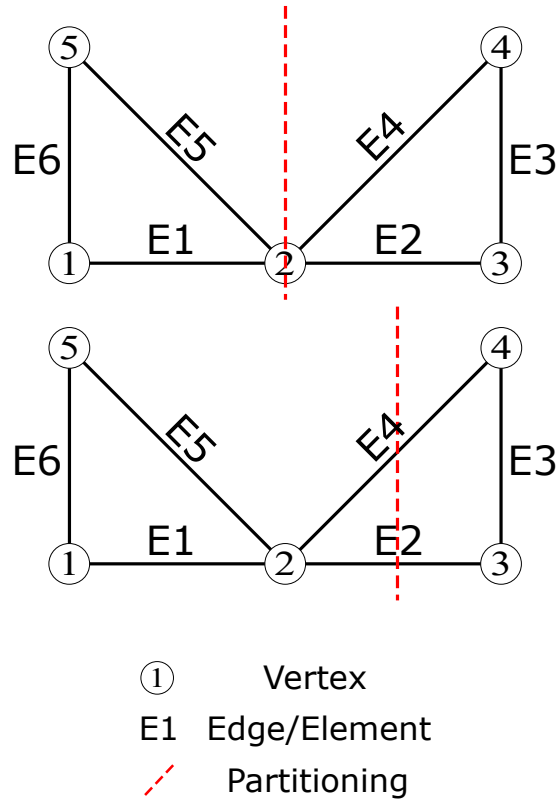


Figure 2.2: Types of partitioning for edge-based elements (Top: Element-based partitioning; Bottom: Vertex-based partitioning)

2.1.1 Type of partitioning

A type of partitioning depends on where one can split a computational domain over a set of topological entities: vertices, edges or faces. For each finite element, an entity or a set of entities can be used to define the element geometry. For example, a point mass can be defined by a vertex, a beam or a cable by an edge, a plate or a shell by a face, as seen in Figure 2.1.

In this thesis, however, only the edge-based structures will be considered for numerical experiments. Thus, it should be noted that the face-based structures will not be used in the partitioning process. Consider a grid of beams which is composed of a set of vertices and edges, see Figure 2.2. When one splits the domain over a set of vertices, it is called *edge-based* partitioning because the set of original edges is intact.

On the contrary, when one splits over a set of edges, it is called *vertex-based* partitioning because of the opposite reason to the previous. But since the vertex-based partitioning will destroy the edge-based element mesh, this type of partitioning will not be used and considered. Only the edge-based partitioning will be used in this thesis.

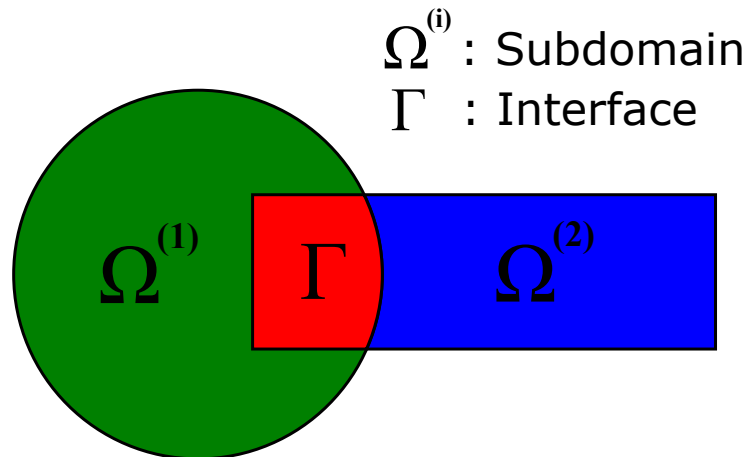


Figure 2.3: Overlapping domain decomposition

2.1.2 Overlapping or non-overlapping subdomains

Subdomains may be overlapped or not overlapped depending on the used domain decomposition method. The solution procedure of overlapping domain decomposition method is very closely related to classical iterative methods such as Gauss-Seidel or Jacobi iteration. On the other hand, non-overlapping domain decomposition methods have been used with the *Schur complement* for classical substructuring which is the early version of parallel structural analysis under the direct solver framework. Today, modern domain decomposition methods for finite element based structural analysis also use the Schur complement as a fundamental concept which is straightforward.

2.1.3 Design of interface

Although subdomain problems usually have most of the dofs out of the total dofs of the original problem, the characteristics of a domain decomposition method is

dictated by the interface problem. The local subdomain problems can be solved in parallel while the solution procedure of the interface problem must somehow pass through sequential steps, which becomes a *bottleneck* for parallel computations. Because the neighboring subdomains were not detached from each other at interface for the original unpartitioned domain, the way of interconnecting them at the interface is very important to ensure the continuity between the neighboring subdomains. How to interconnect them affects the characteristics of the interface problem such as matrix bandwidth, problem conditioning, convergence behavior, etc. Thus, the design of interface is very important to minimize the bottleneck impact on parallel performance and to achieve a well-posed problem at the interface for solid continuity across subdomains.

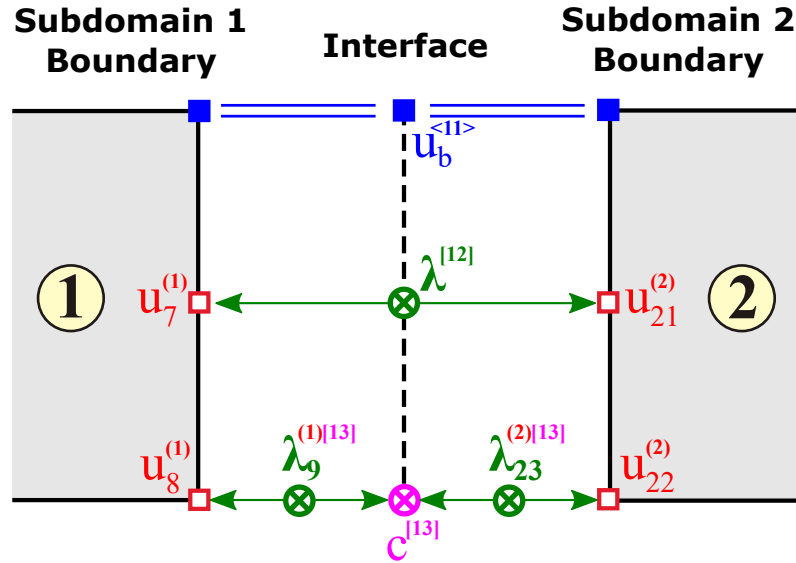


Figure 2.4: Various interface connections (Subscript: Local node number within a subdomain; Superscript in parentheses: Subdomain number; Other superscripts: Global interface node number)

Three types of the interface design are described in Figure 2.4. Multiple types of interface can be combined for a domain decomposition process, or only one type can be used. The interface at the top shares a primal node (square in blue) between two subdomains so that the primal node belongs to the global interface problem as

well as the two local subdomain problems. Since one identical primal node is used for both interface and subdomain problems, the long equal sign is used to express the connection between the interface and the two subdomain boundaries. This type of interface is used for classical substructuring methods. The interface in the middle shares a dual Lagrange multiplier node (circle with a cross in green) between two subdomains, but the subdomain boundary nodes (square in red) are localized into subdomain problems. This type of interface is used in Finite Element Tearing and Interconnecting (FETI) method. The interface at the bottom has its own independent primal node (circle with a cross in pink), but the subdomain boundary nodes and the Lagrange multiplier nodes are localized into subdomain problems. The last type of interface will be used in this thesis for the proposed domain decomposition method and this approach is advocated by Park *et al.* [56].

2.1.4 Solution method for subdomain problems

Subdomain problems can be solved by either a direct or an iterative method. Direct methods are robust and reliable even for ill-conditioned problems. Their performance depends only on the sparsity of the system matrix. Computational cost of them can also be predictable. On the other hand, iterative methods heavily depend on the conditioning of the system matrix. The more ill-conditioned, the slower convergence to the solution. Thus preconditioning of the system matrix is crucial for every iterative method. The computational cost of iterative methods is often unpredictable. But, they can better use the sparsity of the matrix so as to be faster than direct methods for sufficiently large problems. Thus, a solution method for subdomain problems should be carefully chosen by the sparsity and conditioning of the system matrix.

2.2 Non-overlapping Domain Decomposition Methods for Finite Element Analysis

In structure engineering, the *substructuring* method was developed in 1960's for the finite element analysis of complex and large-scale problems [57] allowing both global and local approaches to the problems systematically. The substructuring method falls into the category of non-overlapping domain decomposition methods and the method is also considered another origin of the modern domain decomposition methods as they are all in family methods. It should be very helpful to understand how the non-overlapping domain decomposition methods in the same family have been formulated and how they are distinguished from each other. Thus, the notations used for and the brief derivations of the classical substructuring method and its successors will be covered in the next several sections.

Subdomains and interface

Similar terminology is usually used throughout the literature about domain decomposition methods. To describe the methods without vagueness, it is necessary to define a clear concept of the subdomain, the interface and their connection under the finite element framework.

In the preprocessing phase of the finite element procedure, the finite element discretization generates a complete mesh of a structural model. The finite element mesh consists of elements and nodes. A node is an entity where a set of degrees of freedom (dofs) is defined while an element consists of a set of nodes and represents a partial solution field of a whole problem. The dofs of a node can be primal or dual variables which usually indicate generalized coordinates (displacements and rotations) or Lagrange multipliers, respectively. The type of a node is also defined as a primal node or a dual node depending on the type of dofs.

Assume that a computational domain with a finite element mesh is subdivided

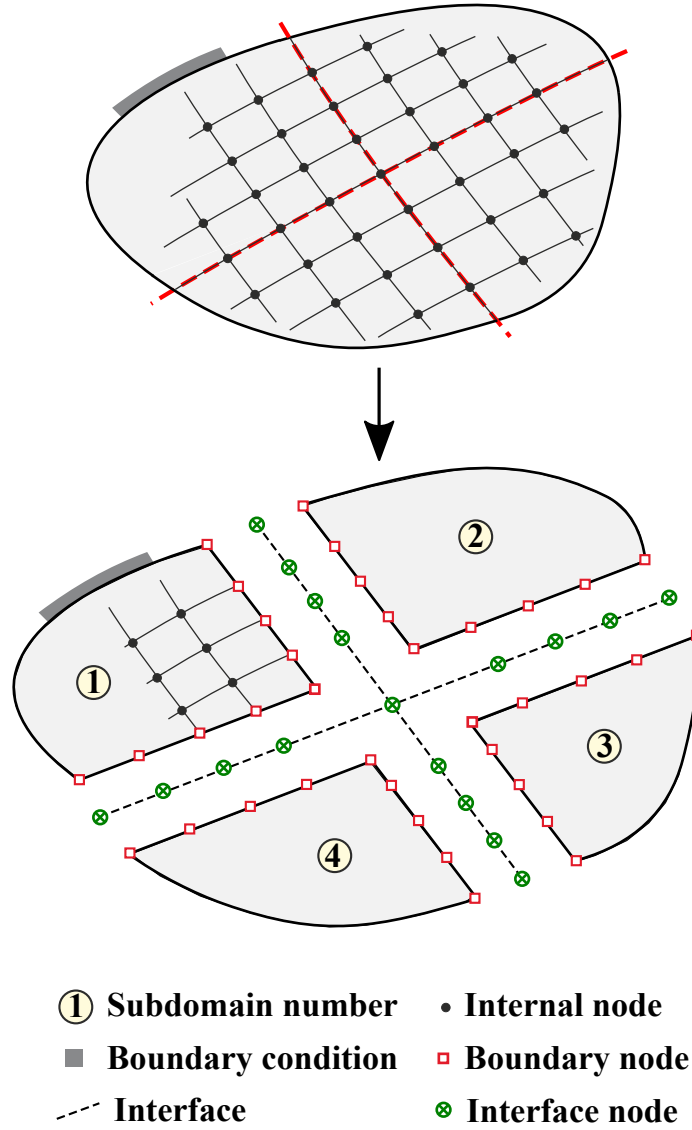


Figure 2.5: Domain decomposition

into non-overlapping subdomains, see Figure 2.5. The in-between entity between subdomains is defined as the interface. The interface nodes can be defined by either just dual variables (forces and moments) between subdomains or actual independent primal variables (displacements and rotations). Conceptually subdomains are connected at the interface. A subdomain can be composed of multiple finite elements, in turn, also composed of the associated nodes with the elements. The nodes can belong to either internal (non-boundary) or boundary of a subdomain. The boundary nodes of a subdomain connects somehow to the nodes of either other subdomain(s) or the

interface itself, while the internal nodes doesn't. The boundary nodes are special because they are the information gates between the subdomains and the interface.

The definition of a subdomain is tangible and fixed because it is just a smaller domain of the original problem, while the interface is quite conceptual, artificial and easily customizable, see Figure 2.4. For example, classical substructuring methods, see Section 2.3, define the interface problem by a set of boundary nodes of subdomains. Thus, for classical substructuring methods, the interface nodes are directly identical to the boundary nodes of subdomains. On the other hand, the methods of the FETI family have different interface structures. Interface nodes are defined by either only dual variables, see Section 2.5 or a combined set of dual and primal variables, see Section 2.7. The design difference of the interface problem is directly related to the characteristics and scalability of a domain decomposition method.

Subdomain problems are inherently *local* while the interface problem is usually global. The purpose of domain decomposition is basically to divide a large problem into small problems and each problem becomes localized. On the other hand, adjacent subdomains are somehow associated with each other at an interface. This indicates that the interface is *global*. Thus, usually, the subdomain problem is *local* while the interface problem is *global*.

Some authors [42] call a boundary of a subdomain a *local interface* because the boundary belongs to a subdomain which is local, but the boundary can also directly communicate with the other subdomain(s). On the other hand, the interface can be called a *global interface* because the interface is an aggregate entity over all subdomains and it globally communicates with all the subdomains.

2.3 Classical Substructuring Method

The history of non-overlapping domain decomposition methods started at *substructuring* for finite element analysis in structure engineering. Substructuring was initially designed for systematic approach to large scale problems with finite element discretization. In finite element framework, a structure model can be constructed element by element, just like building blocks. This means a physical problem can initially be split into smaller problems by engineer's convenience and can be re-assembled into the whole problem. From these reasons, finite element methods enable us to model any kinds of complex structures for quality predictions in structural engineering. This *divide-and-conquer* concept of finite element methods could be readily applicable to the substructuring.

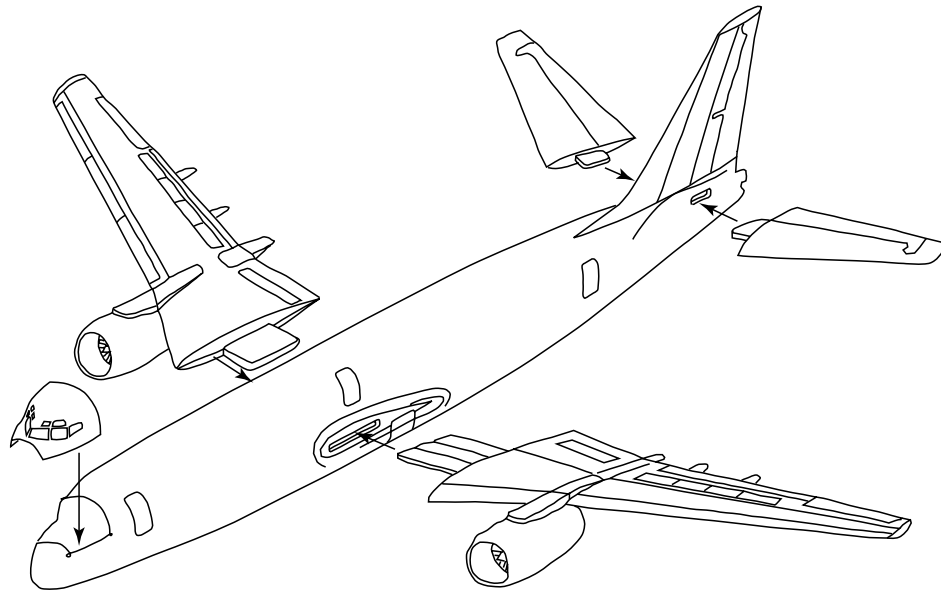


Figure 2.6: Substructuring of an aircraft (Source: Course material for Introduction to Finite Element Methods [36])

Substructuring is basically partitioning an original structure into components (substructures) at the outset of finite element analysis, see Figure 2.6. It enables engineers to be able to systematically manage a very large finite element model so

that different design groups can work on different substructures separately and independently. It is also able to control the size of each subproblem to be fitted into limited computer resources. Substructuring can be applied to the problems with repetitive structures.

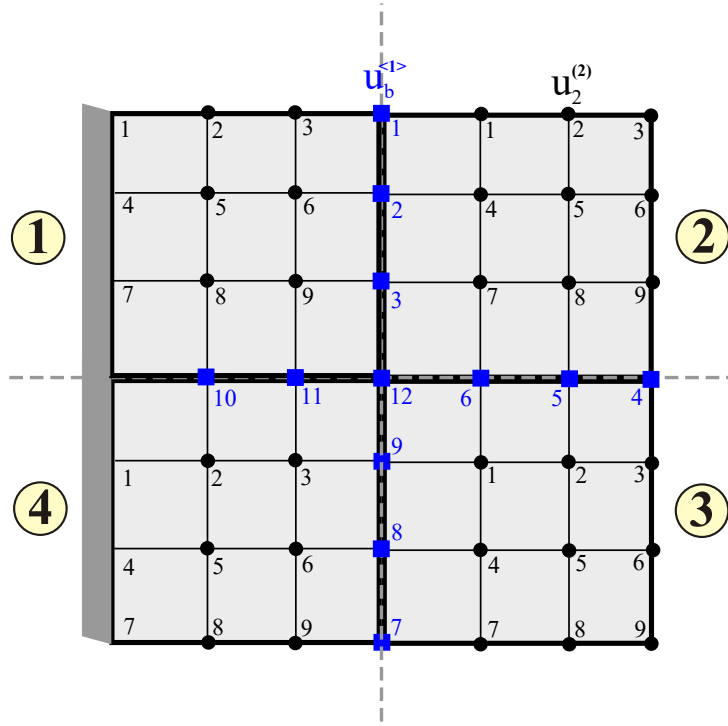


Figure 2.7: Mesh of classical substructuring

This classical substructuring has been the origin of modern domain decomposition methods such as *finite element tearing and interconnecting* (FETI) methods. The term *classical* is used to stress the original motivations of the substructuring method. But the original motivations of the substructuring methods have shifted the focus into computational efficiency with parallel computing.

As seen in Figure 2.7, in the classical substructuring method, all boundary nodes are shared between subdomains unlike the modern domain decomposition methods. Hence, subdomains are not completely detached from each other. Rather they are strongly coupled to their adjacent subdomains. Another illustration of classical substructuring is shown in Figure 2.8. The explanation of the interface in the figures are

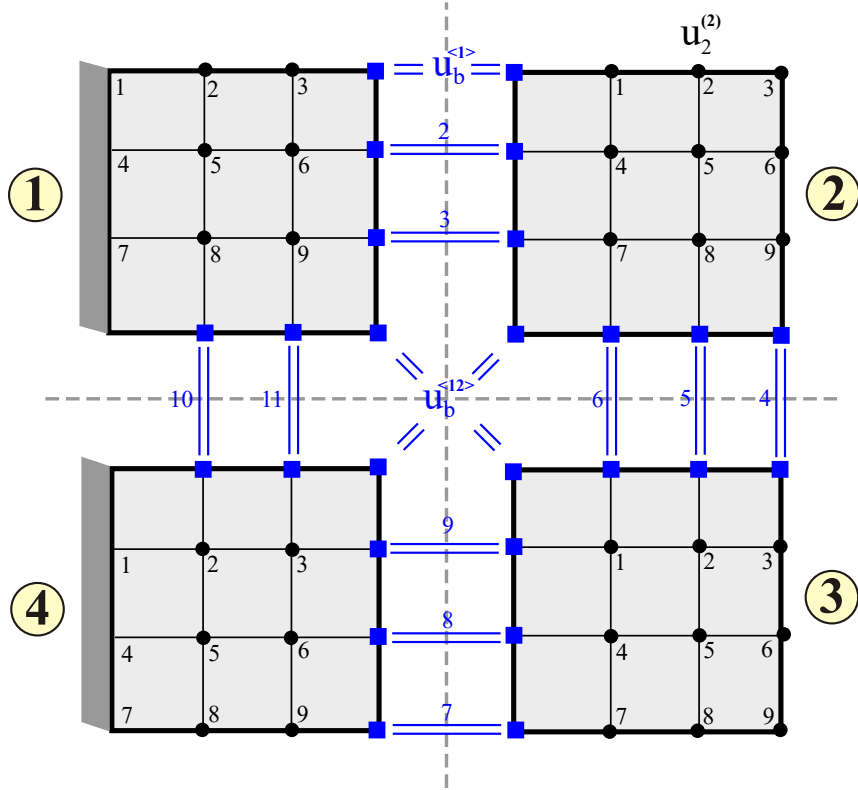


Figure 2.8: Detached-subdomain version of mesh

found in Section 2.1.3 with Figure 2.4.

2.3.1 Condensation process for substructuring

The formulation of substructuring is based on condensation process. Assume that substructuring divides the domain of a whole structure into N_s subdomains or substructures, as in Figure 2.7. The equations of equilibrium of a local subdomain j can be written as

$$\underline{\underline{K}}^{(j)} \underline{u}^{(j)} = \underline{Q}^{(j)}, \quad (2.1)$$

where $\underline{\underline{K}}^{(j)}$, $\underline{u}^{(j)}$ and $\underline{Q}^{(j)}$ are the stiffness matrix, displacement array and externally applied load array, respectively, for the local subdomain j . This can be further

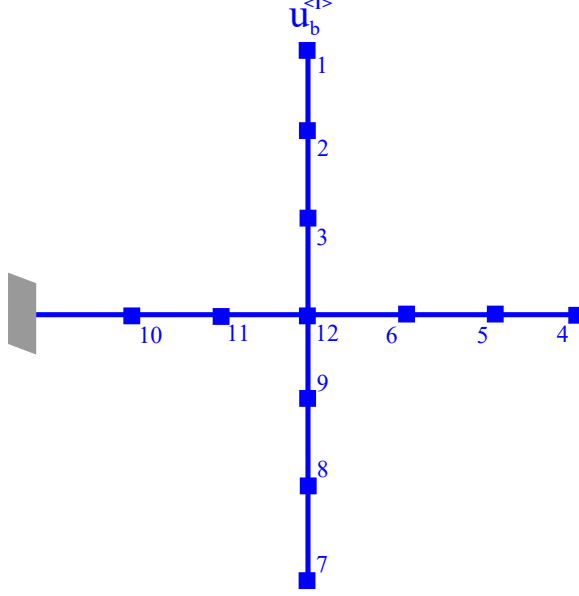


Figure 2.9: Condensed-out mesh with respect to global interface nodes

expressed with another subdivision into the internal and boundary parts as in Equation (2.2),

$$\begin{bmatrix} \underline{\underline{K}}_{ii}^{(j)} & \underline{\underline{K}}_{ib}^{(j)} \\ \underline{\underline{K}}_{ib}^{(j)T} & \underline{\underline{K}}_{bb}^{(j)} \end{bmatrix} \begin{Bmatrix} \underline{u}_i^{(j)} \\ \underline{u}_b^{(j)} \end{Bmatrix} = \begin{Bmatrix} \underline{Q}_i^{(j)} \\ \underline{Q}_b^{(j)} \end{Bmatrix}, \quad (2.2)$$

The subscripts i and b denote internal nodes and boundary nodes, respectively. It should be noted that boundary nodes are also interface nodes in classical substructuring methods. The first row of Equation (2.2) can be solved for the internal displacement array $\underline{u}_i^{(j)}$ as

$$\underline{u}_i^{(j)} = \underline{\underline{K}}_{ii}^{(j)-1} \left\{ \underline{Q}_i^{(j)} - \underline{\underline{K}}_{ib}^{(j)} \underline{u}_b^{(j)} \right\}. \quad (2.3)$$

To eliminate the internal displacement array from the local subdomain problem, substituting $\underline{u}_i^{(j)}$ into the bottom row of Equation (2.2) writes

$$\left[\underline{\underline{K}}_{bb}^{(j)} - \underline{\underline{K}}_{ib}^{(j)T} \underline{\underline{K}}_{ii}^{(j)-1} \underline{\underline{K}}_{ib}^{(j)} \right] \underline{u}_b^{(j)} = \underline{Q}_b^{(j)} - \underline{\underline{K}}_{ib}^{(j)T} \underline{\underline{K}}_{ii}^{(j)-1} \underline{u}_i^{(j)} \quad (2.4)$$

Equation (2.4) is a condensed expression of the subdomain j with respect to the boundary nodes $\underline{u}_b^{(j)}$; the boundary nodes view the subdomain j in this way. This condensation process can be done for all N_s subdomains and every local Equation (2.4)

can be properly assembled in a global matrix-vector equation. The schematic description of the condensed set of boundary nodes can be seen in Figure 2.9.

2.3.2 Governing equations of substructuring

In classical substructuring, the boundary nodes are shared between subdomains *i.e.*, the boundary nodes themselves are the interface nodes. Thus it is better to introduce a global array of all the boundary nodes from all subdomains. The array can be called the global array of boundary/interface nodes. A boolean matrix mapping from the global array of interface nodes to the local array of boundary nodes of a subdomain, is also defined as

$$\underline{u}_b^{(j)} = \underline{\underline{B}}^{(j)} \underline{u}_b. \quad (2.5)$$

The energy method can be used to derive governing equations of a substructuring problem. The total strain energy, A , can be evaluated by adding up the strain energies, $A^{(j)}$ of the N_s subdomains

$$\begin{aligned} A &= \sum_{j=1}^{N_s} A^{(j)} = \sum_{j=1}^{N_s} \frac{1}{2} \underline{u}^{(j)T} \underline{\underline{K}}^{(j)} \underline{u}^{(j)} \\ &= \sum_{j=1}^{N_s} \frac{1}{2} \begin{Bmatrix} \underline{u}_i^{(j)} \\ \underline{\underline{B}}^{(j)} \underline{u}_b \end{Bmatrix}^T \begin{bmatrix} \underline{\underline{K}}_{ii}^{(j)} & \underline{\underline{K}}_{ib}^{(j)} \\ \underline{\underline{K}}_{ib}^{(j)T} & \underline{\underline{K}}_{bb}^{(j)} \end{bmatrix} \begin{Bmatrix} \underline{u}_i^{(j)} \\ \underline{\underline{B}}^{(j)} \underline{u}_b \end{Bmatrix}. \end{aligned} \quad (2.6)$$

The potential of externally applied loads can also be evaluated by adding up the potentials for each of the subdomains

$$\begin{aligned} \Phi &= \sum_{j=1}^{N_s} \Phi^{(j)} = - \sum_{j=1}^{N_s} \underline{u}^{(j)T} \underline{Q}^{(j)} \\ &= - \sum_{j=1}^{N_s} \begin{Bmatrix} \underline{u}_i^{(j)} \\ \underline{\underline{B}}^{(j)} \underline{u}_b \end{Bmatrix}^T \begin{Bmatrix} \underline{Q}_i^{(j)} \\ \underline{Q}_b^{(j)} \end{Bmatrix}. \end{aligned} \quad (2.7)$$

Taking the variation in the total potential energy $\Pi = A + \Phi$,

$$\begin{aligned} 0 &= \delta\Pi = \delta A + \delta\Phi \\ &= \sum_{j=1}^{N_s} \left\{ \delta \underline{u}_i^{(j)T} \left[\frac{\partial A}{\partial \underline{u}_i^{(j)}} + \frac{\partial \Phi}{\partial \underline{u}_i^{(j)}} \right] \right\} + \delta \underline{u}_b^T \left[\frac{\partial A}{\partial \underline{u}_b} + \frac{\partial \Phi}{\partial \underline{u}_b} \right], \end{aligned} \quad (2.8)$$

the principle of minimum total potential energy tells that the square-bracketed terms in Equation (2.8) must vanish for all arbitrary variations in $\underline{u}_i^{(j)}$ and \underline{u}_b , yielding the governing equations from the classical substructuring method as,

$$\underline{\underline{K}}_{ii}^{(j)} \underline{u}_i^{(j)} + \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right] \underline{u}_b = \underline{Q}_i^{(j)}, \quad (2.9)$$

$$\sum_{j=1}^{N_s} \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right]^T \underline{u}_i^{(j)} + \underline{\underline{K}}_{bb} \underline{u}_b = \underline{Q}_b, \quad (2.10)$$

where

$$\underline{\underline{K}}_{bb} = \sum_{j=1}^{N_s} \underline{\underline{B}}^{(j)T} \underline{\underline{K}}_{bb}^{(j)} \underline{\underline{B}}^{(j)} \quad \text{and} \quad \underline{Q}_b = \sum_{j=1}^{N_s} \underline{\underline{B}}^{(j)T} \underline{Q}_b^{(j)}. \quad (2.11)$$

Equation (2.9) is the equations of equilibrium of a local subdomain, which is equivalent to the first row of Equation (2.2). Equation (2.10) is the equations of equilibrium of the globally condensed interface as they are expressed in the summation of the contributions from local subdomains.

2.3.3 Global interface problem

The classical substructuring method sets up the global interface problem by eliminating the local subdomain displacement array, see Equation (2.12) from the formulation, which is the condensation process explained in the previous Section 2.3.1,

$$\underline{u}_i^{(j)} = \underline{\underline{K}}_{ii}^{(j)-1} \left\{ \underline{Q}_i^{(j)} - \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right] \underline{u}_b \right\}. \quad (2.12)$$

Substituting Equation (2.12) for $\underline{u}_i^{(j)}$ in Equation (2.10) yields the global interface problem as

$$\underline{\underline{S}} \underline{u}_b = \underline{f} \quad (2.13)$$

where

$$\underline{\underline{S}} = \underline{\underline{K}}_{bb} - \sum_{j=1}^{N_s} \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right]^T \underline{\underline{K}}_{ii}^{(j)-1} \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right], \quad (2.14)$$

$$\underline{f} = \underline{Q}_b - \sum_{j=1}^{N_s} \left[\underline{\underline{K}}_{ib}^{(j)} \underline{\underline{B}}^{(j)} \right]^T \underline{\underline{K}}_{ii}^{(j)-1} \underline{Q}_i^{(j)}. \quad (2.15)$$

The matrix S in Equation (2.13) is so-called *Schur complement*. Once the global interface solution array \underline{u}_b is known by solving Equation (2.13), then every local subdomain problem, see Equation (2.12) can be solved for the local solution, $\underline{u}_i^{(j)}$, of each subdomain problem.

2.4 *FETI and Its Variants*

Finite Element Tearing and Interconnecting or FETI method was first introduced by Farhat and Roux [29] and the method has gained its popularity and continuously improved in the last two decades. As the name suggests, finite element discretization is used for meshing before domain decomposition. The *tearing* process is the domain decomposition to divide a computational domain into non-overlapping subdomains while the *interconnecting* process is interpreted as enforcing the continuity of the displacement field across subdomain interfaces. Thus, the FETI method falls in the category of non-overlapping domain decomposition methods. Since the FETI method succeeds the classical substructuring method and uses iterative methods to solve a global interface problem with Schur complement matrix, it is also classified as an *iterative substructuring* method. Because this domain decomposition method opens an avenue to a solid approach to general structural problems with parallel scalability, the original FETI method has been continuously developed in more refined and unified manners.

In order to easily distinguish the original FETI method and its variants, from now on, the original FETI method is denoted by FETI-1 (one-level FETI). Its variants, two-level FETI and Dual-Primal FETI are denoted by FETI-2 and FETI-DP, respectively.

2.5 *FETI-1*

The FETI-1 method has two fundamentally distinct features from the classical substructuring methods: Lagrange multipliers and rigid body modes. The FETI-1

method introduces Lagrange multipliers to enforce equality constraints along the boundary between subdomains. With the existence of Lagrange multipliers, subdomains don't share any primal dofs between subdomains and they are connected only through Lagrange multipliers which are dual dofs. As a result of Lagrange multipliers, rigid body modes of unrestrained subdomains must appear in the formulation to correctly account for "floating" subdomains which are non-overlapping and completely detached from each other, unlike the classical substructuring methods where subdomains share primal dofs along the boundaries and are not detached from each other. Although earlier works by Roux [59, 60] already used Lagrange multipliers for domain decomposition, they couldn't be applied to more general problems because of no consideration of rigid body modes.

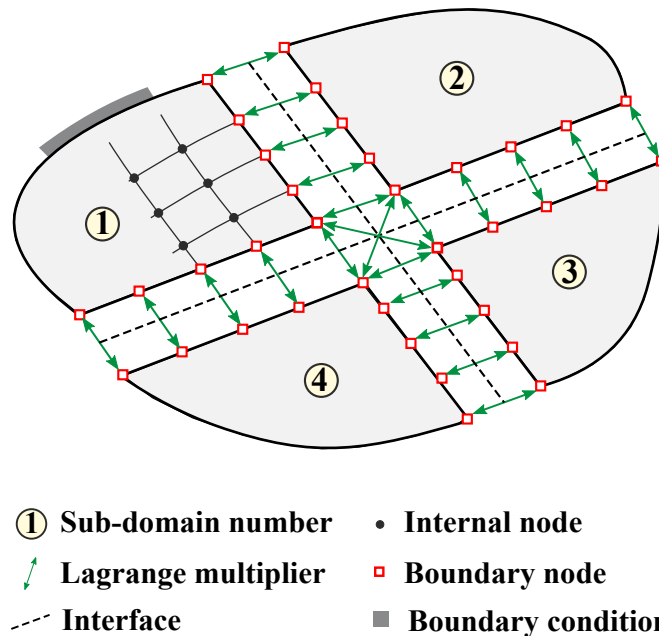


Figure 2.10: FETI-1 domain decomposition method

One more difference of the FETI-1 method from the classical substructuring methods is that it doesn't have zero-measure interface, *i.e.*, the FETI-1 method doesn't need to consider vertex interface and edge interface in 2D problems and 3D problems,

respectively. This change reduces the inter-processor communication in parallel computing so that the FETI-1 method can achieve better scalability than the classical substructuring methods.

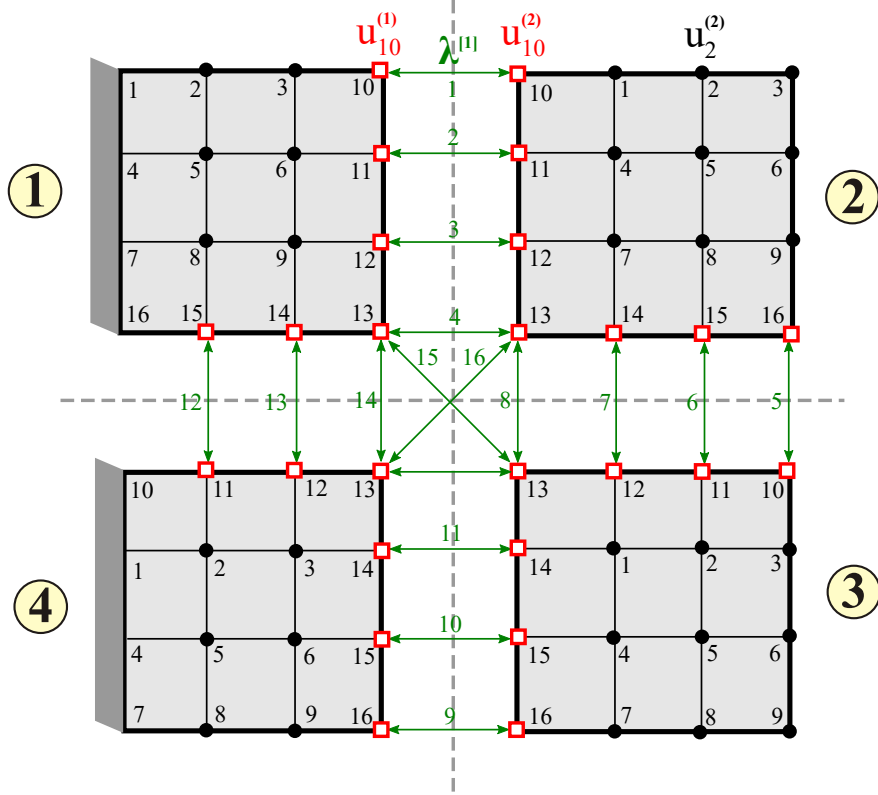


Figure 2.11: Mesh of FETI-1 domain decomposition

2.5.1 Subdomains and constraints at interface

Consider the planar solid depicted in Figure 2.10. The solid is partitioned into N_s non-overlapping subdomains. The degrees of freedom (dofs) for each subdomain are collected in arrays denoted $\underline{u}^{(i)}$, $i = 1, 2, \dots, N_s$. The array $\underline{u}^{(i)}$ stores the dofs of the subdomain i *i.e.*, the displacement components at all the nodes of the subdomain. This array is of size $n_u^{(i)}$, which is the total number of dofs for subdomain i . Notation $(\cdot)^{(i)}$ indicates quantities associated with subdomain i . The global array of dofs is defined as

$$\underline{u}^T = \left\{ \underline{u}^{(1)T}, \underline{u}^{(2)T}, \dots, \underline{u}^{(N_s)T} \right\}. \quad (2.16)$$

Array \underline{u} is of size $n_u = \sum_{i=1}^{N_s} n_u^{(i)}$, which is the total number of dofs for the complete structure. As the original domain is divided into subdomains, the nodes along the interfaces and the associated dofs are duplicated; an internal node before decomposition become two or more boundary nodes between subdomain interfaces after decomposition. Consequently, the array \underline{u} contains a large number of redundant dofs: all interface dofs appear twice or more times. The variables stored in array \underline{u} should be called *generalized coordinates* because they do not form a minimum set, but the term *dofs*, more widely used in the finite element literature, will be used here.

Now, for the interface of the problem with the FETI-1 domain decomposition, consider an equality or homogeneous kinematic constraint $\underline{\mathcal{C}}^{[1]}$ between two boundary nodes $\underline{u}_{10}^{(1)}$ and $\underline{u}_{10}^{(2)}$ of the two subdomains 1 and 2 as seen in Figure 2.11,

$$\underline{\mathcal{C}}^{[1]} = \underline{u}_{10}^{(1)} - \underline{u}_{10}^{(2)} = 0, \quad (2.17)$$

$$V_c^{[1]} = \underline{\lambda}^{[1]T} \underline{\mathcal{C}}^{[1]}, \quad (2.18)$$

where the superscript in parentheses $(\cdot)^{(i)}$ indicates a subdomain number while the square-bracketed superscript $(\cdot)^{[\alpha]}$ denotes the constraint α in the global array of all constraints. Equations (2.17) and (2.18) tells that the continuity of the displacement at the boundary between two subdomains is enforced by the equality constraint $\underline{\mathcal{C}}^{[1]}$ via a Lagrange multiplier $\underline{\lambda}^{[1]}$ which is a constraint force rigidly connecting the two boundary nodes and also yields a constraint potential $V_c^{[1]}$ with the constraint.

Due to the inter-subdomain relation of every kinematic equality constraint in the FETI-1 domain decomposition, a global collection of all the equality constraints is expressed only by the sum of boolean operations of all subdomains as seen in Equation (2.19). Each boolean operation is to extract only the boundary nodes from the node arrays of each subdomain. All $\underline{\underline{B}}^{(i)}$'s have the same number of rows for the size of a single global problem but they can have different numbers of columns because they are associated with different subdomains. To connect the N_s subdomains, let

say, a total of N_c equality constraints is defined for the domain decomposition and the size of the global constraint array $\underline{\mathcal{C}}$ is $n_c = \sum_{\alpha=1}^{N_c} n_c^{[\alpha]}$ in total because each constraint equation may have different dofs $n_c^{[\alpha]}$. Then the size of each $\underline{\underline{B}}^{(i)}$ is the size of $(n_c \times n_u^{(i)})$.

$$\underline{0} = \underline{\mathcal{C}} = \begin{Bmatrix} \underline{\mathcal{C}}^{[1]} \\ \underline{\mathcal{C}}^{[2]} \\ \vdots \\ \underline{\mathcal{C}}^{[N_c]} \end{Bmatrix} = \underline{\underline{B}}^{(1)} \underline{u}^{(1)} + \underline{\underline{B}}^{(2)} \underline{u}^{(2)} + \dots + \underline{\underline{B}}^{(N_s)} \underline{u}^{(N_s)} = \sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{u}^{(i)}. \quad (2.19)$$

A signed boolean matrix $\underline{\underline{B}}^{(i)}$ performs two operations: mapping a node array of a subdomain i onto its boundary nodes and assembling the boundary nodes into the global array of all equality constraints. It is clear that each $\underline{\underline{B}}^{(i)}$ is very sparse with many zero entries.

It should be noted that each equality constraint equation is not localized into a subdomain, but coupled with two subdomains, and this gives rise to inter-subdomain coupling via unknown dual variables, *i.e.*, Lagrange multipliers as in Equation (2.18), which are one of the key ingredients of the FETI-1 method.

For the generalization of the definition of an inter-subdomain equality constraint, more sophisticated notations may be used as the following. To better identify a constraint equation α associated with two boundary nodes of two subdomains i and j , a combined match of a parenthesis and a square bracket can be used for a superscript of a boundary node with both a subdomain number and a constraint number, *e.g.*, $\underline{u}_b^{(i,\alpha]}$ and $\underline{u}_b^{(j,\alpha]}$. Two boundary nodes associated with the constraint α can be extracted by two boolean matrices $\underline{\underline{B}}^{(i,\alpha]}$ and $\underline{\underline{B}}^{(j,\alpha]}$ from the two node arrays of the two subdomains, $\underline{u}^{(i)}$ and $\underline{u}^{(j)}$, and they yield an equality constraint equation as

$$\underline{\mathcal{C}}^{[\alpha]} = \underline{u}_b^{(i,\alpha]} - \underline{u}_b^{(j,\alpha]} = \underline{\underline{B}}^{(i,\alpha]} \underline{u}^{(i)} + \underline{\underline{B}}^{(j,\alpha]} \underline{u}^{(j)}. \quad (2.20)$$

Now a boolean matrix for a subdomain can be assembled by sub-boolean matrices $\underline{\underline{B}}^{(i,\alpha]}$ where $\alpha = 1, 2, \dots, N_c$ as in Equation (2.21),

$$\underline{\underline{B}}^{(i)} = \begin{bmatrix} \underline{\underline{B}}^{(i,1]} \\ \underline{\underline{B}}^{(i,2]} \\ \vdots \\ \underline{\underline{B}}^{(i,N_c]} \end{bmatrix}, \quad (2.21)$$

where the size of a sub-boolean matrix $\underline{\underline{B}}^{(i,\alpha]}$ is $(n_c^{[\alpha]} \times n_u^{(i)})$.

2.5.2 Governing equations

The governing equations of the problem which is domain-decomposed by the FETI-1 method can be derived by variational principles such as the principle of the minimum total potential energy.

The total strain energy in the structure, A , can be evaluated by summing the strain energies, $A^{(i)}$, of the various subdomains

$$A = \sum_{i=1}^{N_s} A^{(i)} = \sum_{i=1}^{N_s} \frac{1}{2} \underline{\underline{u}}^{(i)T} \underline{\underline{K}}^{(i)} \underline{\underline{u}}^{(i)}, \quad (2.22)$$

where $\underline{\underline{K}}^{(i)}$ is the stiffness matrix for subdomain i . The stiffness matrix of each subdomain is obtained by assembling the stiffness matrices of all finite element belonging to the subdomain. If the original solid is suitably constrained by a set of boundary conditions that prevent overall rigid body motions, its global stiffness matrix will not be singular. This property is not shared by the stiffness matrices of individual subdomains: indeed, due to the partitioning of the solid into possibly unconstrained or “floating subdomains,” the stiffness matrix of each subdomain is potentially singular. The total potential of the externally applied loads, Φ , is found by summing up the potentials for each of the subdomains

$$\Phi = \sum_{i=1}^{N_s} \Phi^{(i)} = - \sum_{i=1}^{N_s} \underline{\underline{u}}^{(i)T} \underline{\underline{Q}}^{(i)}, \quad (2.23)$$

where array $\underline{\underline{Q}}^{(i)}$, of size $n^{(i)}$, stores the load applied to subdomain i . The kinematic constraints of the problem give rise to the potential of the constraints, V_c , which is

written as

$$V_c = \underline{\lambda}^T \sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{u}^{(i)}, \quad (2.24)$$

where array $\underline{\lambda}^T = \left\{ \underline{\lambda}_1^T, \underline{\lambda}_2^T, \dots, \underline{\lambda}_{N_c}^T \right\}$, of size n_c stores all the Lagrange multipliers of the problem.

The total potential energy of the system, $\Pi = A + \Phi + V_c$, is found by combining Equations (2.22), (2.23), and (2.24) leading to

$$\Pi = \sum_{i=1}^{N_s} \left[\frac{1}{2} \underline{u}^{(i)T} \underline{\underline{K}}^{(i)} \underline{u}^{(i)} - \underline{u}^{(i)T} \underline{Q}^{(i)} \right] + \underline{\lambda}^T \sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{u}^{(i)}, \quad (2.25)$$

The variation of the total potential energy must vanish for all arbitrary variations in unknown variables (every subdomain node array and global Lagrange multipliers), and this can be written as,

$$0 = \delta\Pi = \sum_{i=1}^{N_s} \delta \underline{u}^{(i)T} \left[\underline{\underline{K}}^{(i)} \underline{u}^{(i)} - \underline{Q}^{(i)} + \underline{\underline{B}}^{(i)T} \underline{\lambda} \right] + \delta \underline{\lambda}^T \left[\sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{u}^{(i)} \right]. \quad (2.26)$$

Consequently, the bracketed terms must vanish and this writes Equations (2.27) and (2.28).

$$\underline{\underline{K}}^{(i)} \underline{u}^{(i)} + \underline{\underline{B}}^{(i)T} \underline{\lambda} = \underline{Q}^{(i)}, \quad i = 1, 2, \dots, N_s \quad (2.27)$$

$$\sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{u}^{(i)} = 0. \quad (2.28)$$

Equation (2.27) clearly indicates the equations of equilibrium of each subdomain under both externally applied loads and constraint forces from adjacent subdomains via Lagrange multipliers to enforce the continuity across the subdomain boundary. Equation (2.28) is just the global collection of equality constraint equations as seen in Equation (2.19).

However, a subdomain which is not restrained by any prescribed boundary conditions, is *floating* as explained earlier. This means that a stiffness matrix of a floating subdomain is singular and not invertible. Thus, instead of a regular inverse, a pseudo inverse of the stiffness matrix must be used for all floating subdomains. Also, the

total response of a floating subdomain must be expressed by the sum of the rigid body mode response as well as the elastic mode response. In fact, the rigid body modes of a subdomain correspond to the null space of the stiffness matrix of the subdomain. The total response of a floating subdomain i with the additional treatment for rigid body modes, can be expressed as in Equation (2.29),

$$\underline{u}^{(i)} = \underline{d}^{(i)} + \underline{r}^{(i)} = \underline{\underline{K}}^{(i)+} \left[\underline{Q}^{(i)} - \underline{\underline{B}}^{(i)T} \underline{\lambda} \right] + \underline{\underline{R}}^{(i)} \underline{\alpha}^{(i)}, \quad (2.29)$$

where, for a floating subdomain i , $\underline{d}^{(i)}$ is the elastic mode response, $\underline{r}^{(i)}$ the rigid body mode response, $\underline{\underline{K}}^{(i)+}$ the pseudo inverse of the stiffness matrix, $\underline{\underline{R}}^{(i)}$ the rigid body modes of the subdomain i , $\underline{\alpha}$ the amplitude of the rigid body modes. The size of the additional unknown vector $\underline{\alpha}$ is six or fewer because there are at most six rigid body modes in three dimensional problems. It is necessary to introduce additional equations for the additional unknowns $\underline{\alpha}$ and they are formulated from the orthogonality condition between the elastic modes and the rigid body modes as in Equation (2.30),

$$\underline{\underline{R}}^{(i)T} \left[\underline{Q}^{(i)} - \underline{\underline{B}}^{(i)T} \underline{\lambda} \right] = 0. \quad (2.30)$$

2.5.3 Global interface problem

The global coarse problem is set up by eliminating the local subdomain response $\underline{u}^{(i)}$ from the formulation. Substituting Equation (2.29) into Equation (2.28) and combining it with Equation (2.30) yields the global coarse interface problem in a matrix-vector form as in Equation (2.31),

$$\begin{bmatrix} \underline{\underline{F}}_I & -\underline{\underline{G}}_I \\ -\underline{\underline{G}}_I^T & \underline{\underline{0}} \end{bmatrix} \begin{Bmatrix} \underline{\lambda} \\ \underline{\alpha} \end{Bmatrix} = \begin{Bmatrix} \underline{d} \\ -\underline{e} \end{Bmatrix}, \quad (2.31)$$

where

$$\begin{aligned}
\underline{\underline{F}}_I &= \sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{\underline{K}}^{(i)+} \underline{\underline{B}}^{(i)T} \\
\underline{\underline{G}}_I &= \left[B^{(1)} \underline{\underline{R}}^{(1)}, \dots, \underline{\underline{B}}^{(N_s)} \underline{\underline{R}}^{(N_s)} \right] \\
\underline{\underline{\alpha}}^T &= [\underline{\underline{\alpha}}_1^T, \dots, \underline{\underline{\alpha}}_{N_s}^T] \\
\underline{\underline{d}} &= \sum_{i=1}^{N_s} \underline{\underline{B}}^{(i)} \underline{\underline{K}}^{(i)+} \underline{\underline{Q}}^{(i)} \\
\underline{\underline{e}} &= \underline{\underline{R}}^{(i)T} \underline{\underline{Q}}^{(i)} \\
\underline{\underline{K}}^{(i)+} &= \underline{\underline{K}}^{(i)-1} \quad \text{if the subdomain } i \text{ is not floating.}
\end{aligned} \tag{2.32}$$

It should be noted that the submatrix $\underline{\underline{F}}_I$ can be positive semi-definite if any redundant equality constraint equation is defined at any cross point where two or more subdomains are connected to. Even though $\underline{\underline{F}}_I$ is positive semi-definite, if the problem is solved by an iterative method, the method will yield a unique solution at convergence anyway because iterative methods don't require an explicit inverse of system matrix. But, the FETI-1 domain decomposition method doesn't provide a unique way of defining constraint condition at the cross point, the convergence behavior of the problem depends on how the constraint condition is defined because information flow or solution propagation is not uniform for different choices of the constraint condition, which is pointed out by Park *et al.*[56].

2.5.4 Solution procedure for global coarse interface problem

Equation (2.31) of the global coarse interface problem may be solved by iterative methods such as *Preconditioned Conjugate Gradient* (PCG) to avoid computing the explicit inverse matrix and matrix-matrix multiplications. Equation (2.31) is expressed in terms of both $\underline{\underline{\lambda}}$ and $\underline{\underline{\alpha}}$ and the unknown $\underline{\underline{\alpha}}$ can be eliminated by an orthogonal projection of the matrix $\underline{\underline{G}}_I$. The orthogonal projection operator $\underline{\underline{P}}$ is defined as

$$\underline{\underline{P}} = \underline{\underline{I}} - \underline{\underline{G}}_I \left[\underline{\underline{G}}_I^T \underline{\underline{G}}_I \right]^{-1} \underline{\underline{G}}_I^T \tag{2.33}$$

where $\underline{\underline{I}}$ is the identity matrix. Then, Equation (2.31) is pre-multiplied by the orthogonal projection operator and transformed into

$$\underline{\underline{P}}\underline{\underline{F}}_I\lambda = \underline{\underline{P}}d \quad \text{subject to} \quad \underline{\underline{G}}_I^T\lambda = \underline{\underline{e}}. \quad (2.34)$$

Since Equation (2.34) is equivalent to a constrained optimization problem as

$$\text{Minimize } \Phi(\lambda) = \frac{1}{2}\lambda^T \underline{\underline{P}}\underline{\underline{F}}_I\lambda - \lambda^T \underline{\underline{P}}d \quad \text{subject to} \quad \underline{\underline{G}}_I^T\lambda = \underline{\underline{e}}, \quad (2.35)$$

the residual of Equation (2.35) or the negative gradient on the paraboloid of the minimization problem can be defined as

$$\underline{\underline{P}} \left[\underline{\underline{d}} - \underline{\underline{F}}_I\lambda \right] = \underline{\underline{Pr}}. \quad (2.36)$$

Thus, the gradient must be projected at each PCG iteration and this modified PCG method is called *Preconditioned Conjugate “Projected” Gradient* (PCPG).

In the FETI-1 method, local subdomain problems are in general solved by direct methods in parallel while the global coarse interface problem is solved by the PCPG iterative method. Direct methods are better than iterative methods for local subdomain problems because the computational time of direct methods can be predicted unlike iterative methods and every subdomain problem can be solved in similar computing time once the workloads on CPUs are well balanced.

2.5.5 Preconditioning

The convergence speed of iterative methods is directly related to the condition number of the system matrix. Thus, a good preconditioner is necessary for an iterative method to solve a problem in a reasonable computation time. The FETI-1 method also uses the preconditioner for better convergence and the condition number, κ , is expressed as

$$\kappa = O \left(1 + \log \frac{H}{h} \right)^m \quad m \leq 3, \quad (2.37)$$

where m is the dimension of a problem, H the subdomain size, h the element mesh size. Although the FETI-1 method achieved the optimal scalability with a good preconditioner for the second-order elasticity problems, it didn't for the fourth-order problems such as plate or shell problems.

2.6 FETI-2

The improved version of the original FETI method came out to resolve the corner point/node problem of fourth order plates and shells, see Figure 2.12, by introducing the second level coarse grid problem. This is why the method is called two-level FETI or FETI-2 method.

The main problem of FETI-1 was bad scalability for the fourth-order plate or shell problems because the coarse problem becomes more ill-conditioned as the number of elements of a subdomain increases. It turns out that, for example, when a rectangular plate is partitioned like a chessboard, the amplitudes of Lagrange multipliers get erroneously too high at corner points where more than two subdomains are crossing or two subdomain edges are crossing. This leads to a slow convergence to the solution. Thus, it was necessary to modify the global coarse problem of the FETI-1 method by adding corrective terms to the Lagrange multipliers and enforcing the exact continuity at subdomain corner points.

FETI-2 method has two coarse problems. One coarse problem is the same as FETI-1, which accounts for both the continuity of the displacement field along the subdomain boundary and rigid body modes of floating subdomains. Another coarse problem is generated by a projection of FETI-1 coarse problem only onto corner points. In other words, FETI-2 solves both a first-level interface problem along subdomain boundary and a second-level interface problem at corner points. This two-level interface problem approach guarantees optimal scalability even for fourth-order problems with corner points.

2.7 FETI-DP

The next improved version is called *Dual-Primal* FETI or FETI-DP method. In the very beginning of the formulation, the FETI-DP method introduces new segregation of primal nodes of a subdomain, unlike the previous FETI methods. A primal node of a subdomain falls into either a *corner* or *remainder* node, see Figure 2.12; the remainder nodes of a subdomain include internal nodes and boundary nodes, except for the corner nodes. The definition of a corner node was already raised in the FETI-2 method because the corner nodes are the major barrier to optimal scalability of the fourth-order problems with the FETI-1 method. In the FETI-DP method, the primal corner point dofs of fourth-order problems, are collected into the global problems together with Lagrange multipliers which are dual dofs. Now the global coarse problem is expressed by both dual and primal dofs, which is the origin of the Dual-Primal name. Lagrange multipliers are shared by neighboring subdomains only at the remainder nodes along the subdomain boundary, not at the corner nodes. Physical interpretation of this is that neighboring subdomains are detached only at the remainder nodes along subdomain boundary while the subdomains are attached at corner points and no longer floating, see Figure 2.12. Thus the stiffness matrices of subdomains are never singular once the whole structure is properly restrained. No floating subdomains completely eliminate the requirement of rigid body modes from the formulation. This change obviously distinguishes FETI-DP from the previous FETI methods all at once. Null spaces or rigid body modes are no longer required and this enables the FETI-DP method to be used for dynamic problems without additional treatments. As one last note, the change in the separation of nodes into the corner and remainder nodes makes the global interface matrix more sparse than the previous FETI methods.

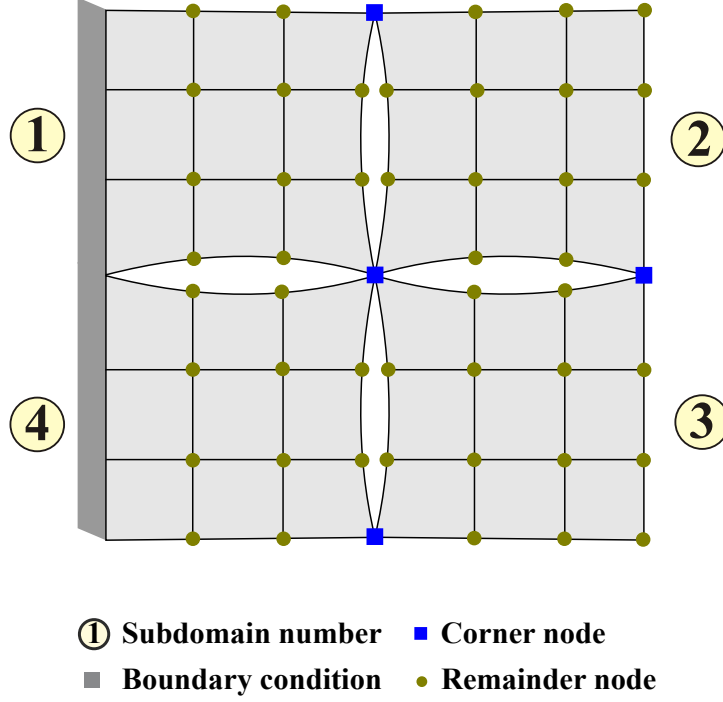


Figure 2.12: FETI-DP corner and remainder nodes

2.7.1 New formulation with corner and remainder nodes

The new arrangement of dofs of the FETI-DP method give rises to the changes in the underlying physics of the previous versions of this domain decomposition method. Instead of the traditional arrangement of the subdomain dofs where a dof falls into internal or boundary of a subdomain, the FETI-DP method uses different separation of dofs as in Equation (2.38),

$$\underline{\underline{K}}^{(i)} = \begin{bmatrix} \underline{\underline{K}}_{rr}^{(i)} & \underline{\underline{K}}_{rc}^{(i)} \\ \underline{\underline{K}}_{rc}^{(i)T} & \underline{\underline{K}}_{cc}^{(i)} \end{bmatrix}, \quad \underline{u}^{(i)} = \begin{Bmatrix} u_r^{(i)} \\ u_{bc}^{(i)} \end{Bmatrix}, \quad \underline{Q}^{(i)} = \begin{Bmatrix} Q_r^{(i)} \\ Q_{bc}^{(i)} \end{Bmatrix}, \quad (2.38)$$

where $i = 1, 2, \dots, N_s$. The subscripts r and c denote remainder and corner nodes, respectively. It should be noted that all remainder nodes of a subdomain include both internal nodes and boundary nodes except for the corner nodes of the subdomain. Equation (2.39) shows the arrays of displacement and of externally applied loads.

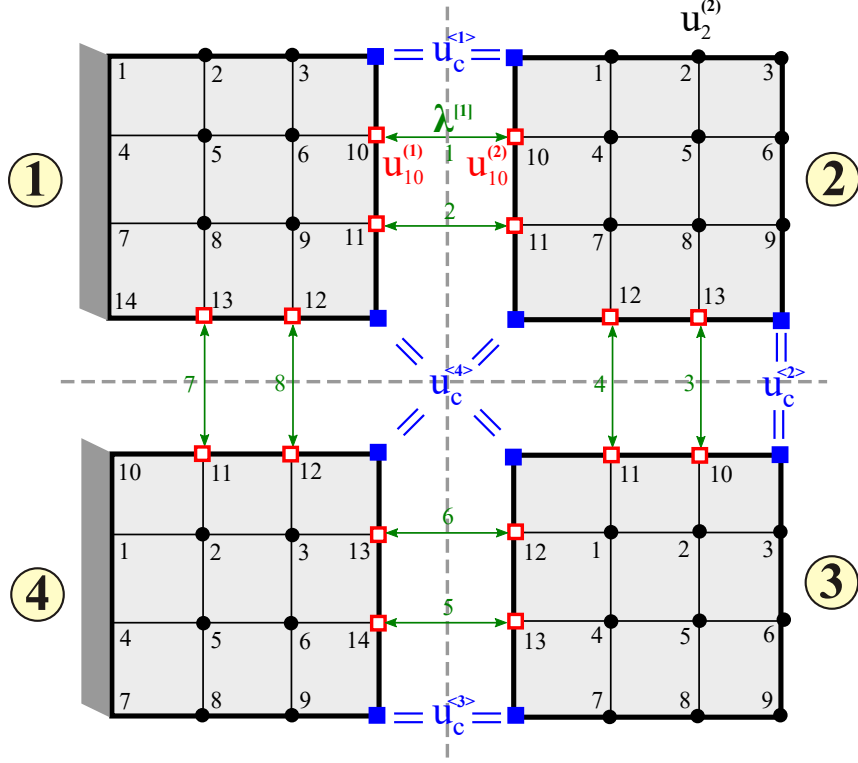


Figure 2.13: FETI-DP domain decomposition mesh

$$\underline{u}_r^{(i)} = \begin{Bmatrix} \underline{u}_i^{(i)} \\ \underline{u}_{br}^{(i)} \end{Bmatrix} \quad \text{and} \quad \underline{Q}_r^{(i)} = \begin{Bmatrix} \underline{Q}_i^{(i)} \\ \underline{Q}_{br}^{(i)} \end{Bmatrix}, \quad (2.39)$$

where the subscripts i and b denote internal and boundary nodes of a remainder node array, respectively. It is noted that the subscript i without parentheses indicates *internal* to a subdomain, not a subdomain number.

Two boolean matrices mapping to boundary nodes both from the remainder nodes of each local subdomain and from the global corner nodes, are also defined as

$$\underline{u}_{br}^{(i)} = \underline{B}_r^{(i)} \underline{u}_r^{(i)} \quad \text{and} \quad \underline{u}_{bc}^{(i)} = \underline{B}_c^{(i)} \underline{u}_c. \quad (2.40)$$

2.7.2 Governing equations

As in the derivation of the governing equations for the FETI-1 method, the FETI-DP method goes through the same derivation process. The total strain energy in the structure, A , can be evaluated by summing up the strain energies, $A^{(i)}$, of the various

subdomains,

$$A = \sum_{i=1}^{N_s} A^{(i)} = \sum_{i=1}^{N_s} \frac{1}{2} \underline{u}^{(i)T} \underline{\underline{K}}^{(i)} \underline{u}^{(i)} = \sum_{i=1}^{N_s} \frac{1}{2} \begin{Bmatrix} \underline{u}_r^{(i)} \\ \underline{B}_{\underline{c}}^{(i)} \underline{u}_c \end{Bmatrix}^T \begin{bmatrix} \underline{\underline{K}}_{rr}^{(i)} & \underline{\underline{K}}_{rc}^{(i)} \\ \underline{\underline{K}}_{rc}^{(i)T} & \underline{\underline{K}}_{cc}^{(i)} \end{bmatrix} \begin{Bmatrix} \underline{u}_r^{(i)} \\ \underline{B}_{\underline{c}}^{(i)} \underline{u}_c \end{Bmatrix}. \quad (2.41)$$

The total potential of the externally applied loads, Φ , is also found by summing up the potentials for each of the subdomains,

$$\Phi = \sum_{i=1}^{N_s} \Phi^{(i)} = - \sum_{i=1}^{N_s} \underline{u}^{(i)T} \underline{Q}^{(i)} = - \sum_{i=1}^{N_s} \begin{Bmatrix} \underline{u}_r^{(i)} \\ \underline{B}_{\underline{c}}^{(i)} \underline{u}_c \end{Bmatrix}^T \begin{Bmatrix} \underline{Q}_r^{(i)} \\ \underline{Q}_{bc}^{(i)} \end{Bmatrix}. \quad (2.42)$$

Lastly, the kinematic constraints of the problem give rise to the potential of the constraints, V_c , which is written as

$$V_c = \underline{\lambda}^T \sum_{i=1}^{N_s} \underline{B}_{\underline{r}}^{(i)} \underline{u}_r^{(i)}. \quad (2.43)$$

The total potential energy of the system, $\Pi = A + \Phi + V_c$, is found by adding up Equations (2.41), (2.42), and (2.43). Taking the variation in the total potential energy is leading to as in Equation (2.44),

$$\begin{aligned} 0 &= \delta\Pi = \delta A + \delta\Phi + \delta V_c \\ &= \sum_{i=1}^{N_s} \left\{ \delta \underline{u}_r^{(i)T} \left[\frac{\partial A}{\partial \underline{u}_r^{(i)}} + \frac{\partial \Phi}{\partial \underline{u}_r^{(i)}} + \frac{\partial V_c}{\partial \underline{u}_r^{(i)}} \right] \right\} + \delta \underline{u}_c^T \left[\frac{\partial A}{\partial \underline{u}_c} + \frac{\partial \Phi}{\partial \underline{u}_c} \right] + \delta \underline{\lambda}^T \left[\frac{\partial V_c}{\partial \underline{\lambda}} \right], \end{aligned} \quad (2.44)$$

The principle of minimum total potential energy tells that the square-bracketed terms in Equation (2.44) must vanish for all arbitrary variations in $\underline{u}_r^{(i)}$, \underline{u}_c and $\underline{\lambda}$, yielding the governing equations from the FETI-DP domain decomposition method as,

$$\underline{\underline{K}}_{rr}^{(i)} \underline{u}_r^{(i)} + \left[\underline{\underline{K}}_{rc}^{(i)} \underline{B}_{\underline{c}}^{(i)} \right] \underline{u}_c = \underline{Q}_r^{(i)} - \underline{B}_{\underline{r}}^{(i)T} \underline{\lambda}, \quad (2.45)$$

$$\sum_{i=1}^{N_s} \left[\underline{\underline{K}}_{rc}^{(i)} \underline{B}_{\underline{c}}^{(i)} \right]^T \underline{u}_r^{(i)} + \underline{\underline{K}}_{cc} \underline{u}_c = \underline{Q}_c, \quad (2.46)$$

$$\sum_{i=1}^{N_s} \underline{B}_{\underline{r}}^{(i)} \underline{u}_r^{(i)} = 0. \quad (2.47)$$

where

$$\underline{\underline{K}}_{cc} = \sum_{i=1}^{N_s} \underline{\underline{B}}_c^{(i)T} \underline{\underline{K}}_{cc}^{(i)} \underline{\underline{B}}_c^{(i)} \quad \text{and} \quad \underline{\underline{Q}}_c = \sum_{i=1}^{N_s} \underline{\underline{B}}_c^{(i)T} \underline{\underline{Q}}_{bc}^{(i)}. \quad (2.48)$$

Equation (2.45) is the equations of equilibrium of each local subdomain. The equation is also very similar to the equations of equilibrium of a local subdomain in the case of classical substructuring, see Equation (2.9), except for the different separation of nodes and the existence of Lagrange multipliers. Equation (2.46) is the equations of equilibrium of the global corner nodes. This equation more resembles the counterpart, see Equation (2.9), of the classical substructuring case. Because the global primal nodes (FETI-DP: corner nodes; Classical substructuring: boundary nodes) are directly shared by subdomains without additional kinematic constraints via Lagrange multipliers, in both domain decomposition methods of the FETI-DP and the classical substructuring. As in the FETI-1 method, the FETI-DP domain decomposition method also ensures the continuity between subdomains by enforcing kinematic constraints in global manner, which are expressed in Equation (2.47).

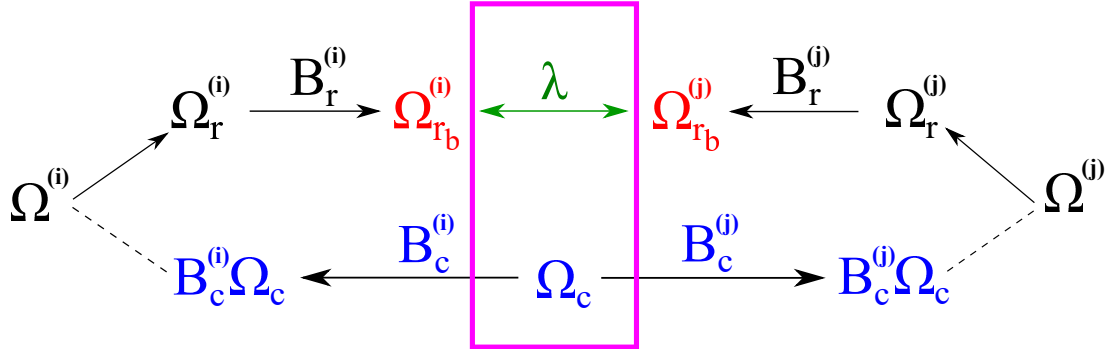


Figure 2.14: FETI-DP domain relationship

2.7.3 Global interface problem

Just as in the FETI-1 method, the FETI-DP method sets up the global coarse interface problem by eliminating the local subdomain responses from the formulation.

$$\begin{bmatrix} \underline{F}_{Irr} & \underline{F}_{Irc} \\ \underline{F}_{Irc}^T & -\underline{K}_{cc}^* \end{bmatrix} \begin{Bmatrix} \lambda \\ \underline{u}_c \end{Bmatrix} = \begin{Bmatrix} \underline{d}_r \\ -\underline{Q}_c^* \end{Bmatrix}, \quad (2.49)$$

where

$$\begin{aligned} \underline{F}_I &= \sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} \underline{B}_r^{(i)T} \\ \underline{F}_{Irc} &= \sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} \begin{bmatrix} \underline{K}_{rc}^{(i)} \underline{B}_c^{(i)} \end{bmatrix} \\ \underline{K}_{cc}^* &= \underline{K}_{cc} - \sum_{i=1}^{N_s} \begin{bmatrix} \underline{K}_{rc}^{(i)} \underline{B}_c^{(i)} \end{bmatrix}^T \underline{K}_{rr}^{(i)-1} \begin{bmatrix} \underline{K}_{rc}^{(i)} \underline{B}_c^{(i)} \end{bmatrix} \\ \underline{d}_r &= \sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} \underline{Q}_r^{(i)} \\ \underline{Q}_c^* &= \underline{Q}_c - \sum_{i=1}^{N_s} \begin{bmatrix} \underline{K}_{rc}^{(i)} \underline{B}_c^{(i)} \end{bmatrix}^T \underline{K}_{rr}^{(i)-1} \underline{Q}_r^{(i)} \end{aligned} \quad (2.50)$$

The dual-primal coarse problem of Equation (2.49) is further reduced into a dual coarse problem by eliminating \underline{u}_c ,

$$\left[\underline{F}_{Irr} + \underline{F}_{Irc} \underline{K}_{cc}^{*-1} \underline{F}_{Irc}^T \right] \lambda = \underline{d}_r - \underline{F}_{Irc} \underline{K}_{cc}^{*-1} \underline{Q}_c^*, \quad (2.51)$$

which is similar to the global coarse problem of the FETI-1 method, see Equation (2.34), but the dual coarse problem of the FETI-DP method doesn't require any projection because the corner point dofs can be completely eliminated from Equation (2.49) unlike the FETI-1 method, being able to express the coarse problem in terms solely of Lagrange multipliers. Thus the projected gradient is also not needed to use PCG method. That is, FETI-DP uses the regular PCG method, not the PCPG method.

2.8 Interface Problem: Classical Substructuring and FETIs

Interface problem dictates a domain decomposition method. The interface problem of each domain decomposition method is expressed by the different set of dofs: the

subdomain boundary nodes \underline{u}_b for the classical substructuring method, the Lagrange multipliers $\underline{\lambda}$ and the amplitudes of rigid body modes $\underline{\alpha}$ for the FETI-1 method, the Lagrange multipliers $\underline{\lambda}$ and the corner nodes \underline{u}_c for the FETI-DP method. With the importance of the interface problem for domain decomposition methods, it would be worth checking the difference among them in a table.

To express interface problems of several non-overlapping domain decomposition methods in one single format, a block-matrix-vector equation can be used and written as Equation (2.52),

$$\begin{bmatrix} \underline{\underline{A}}_{11} & \underline{\underline{A}}_{12} \\ \underline{\underline{A}}_{12}^T & \underline{\underline{A}}_{22} \end{bmatrix} \begin{Bmatrix} \underline{\lambda} \\ \underline{u}_b, \underline{\alpha} \text{ or } \underline{u}_c \end{Bmatrix} = \begin{Bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{Bmatrix}. \quad (2.52)$$

The block matrices and arrays of the interface problem of each method are summarized in Tables 2.1 and 2.2. As seen below, the classical substructuring method is associated with only the boundary nodes \underline{u}_b for the global interface problem, and thus the first row of the block matrix-vector equation is not existing. The first columns of Tables 2.1 and 2.2 are very similar between the FETI-1 and FETI-DP methods because both methods use the Lagrange multipliers to enforce the equality constraints. The second and third columns of Tables 2.1 and 2.2 are also almost identical between the classical substructuring method and FETI-DP methods because they directly share the global primal variables at the interface and generate *Schur* complement problems.

Table 2.1: Comparison of interface matrices between classical substructuring (CS) and FETIs

	$\underline{\underline{A}}_{11}$	$\underline{\underline{A}}_{12}$	$\underline{\underline{A}}_{22}$
CS	N/A	N/A	$\underline{K}_{bb} - \sum_{j=1}^{N_s} [\underline{K}_{ib}^{(j)} \underline{B}^{(j)}]^T \underline{K}_{ii}^{(j)-1} [\underline{K}_{ib}^{(j)} \underline{B}^{(j)}]$
FETI-1	$\sum_{i=1}^{N_s} \underline{B}^{(i)} \underline{K}^{(i)+} \underline{B}^{(i)T}$	$[\underline{B}^{(1)} \underline{R}^{(1)}, \dots, \underline{B}^{(N_s)} \underline{R}^{(N_s)}]$	$\underline{0}$
FETI-DP	$\sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} \underline{B}_r^{(i)T}$	$\sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} [\underline{K}_{rc}^{(i)} \underline{B}_c^{(i)}]$	$\underline{K}_{cc} - \sum_{i=1}^{N_s} [\underline{K}_{rc}^{(i)} \underline{B}_c^{(i)}]^T \underline{K}_{rr}^{(i)-1} [\underline{K}_{rc}^{(i)} \underline{B}_c^{(i)}]$

Table 2.2: Comparison of interface right-hand-side arrays between classical substructuring (CS) and FETIs

	\underline{b}_1	\underline{b}_2
CS	N/A	$\underline{Q}_b - \sum_{j=1}^{N_s} \left[\underline{K}_{ib}^{(j)} \underline{B}^{(j)} \right]^T \underline{K}_{ii}^{(j)-1} \underline{Q}_i^{(j)}$
FETI-1	$\sum_{i=1}^{N_s} \underline{B}^{(i)} \underline{K}^{(i)+} \underline{Q}^{(i)}$	$\underline{R}^{(i)T} \underline{Q}^{(i)}$
FETI-DP	$\sum_{i=1}^{N_s} \underline{B}_r^{(i)} \underline{K}_{rr}^{(i)-1} \underline{Q}_r^{(i)}$	$\underline{Q}_c - \sum_{i=1}^{N_s} \left[\underline{K}_{rc}^{(i)} \underline{B}_c^{(i)} \right]^T \underline{K}_{rr}^{(i)-1} \underline{Q}_r^{(i)}$

CHAPTER III

DOMAIN DECOMPOSITION BY LOCALIZED LAGRANGE MULTIPLIER TECHNIQUE

In this chapter, a new approach to the non-overlapping domain decomposition method and the formulation of it will be presented. Both multibody systems and domain decomposition have a fundamental routine in common; they divide into components and assemble them into a system. Multibody system dynamics community has developed many schemes to analyze many types of many-body-connected systems in a stable and efficient manner. The new domain decomposition approach is closely related to the robust and reliable schemes which were already developed and have been widely used in multibody dynamics community.

It is important for a subdomain to be as independent as possible to each other for computation in parallel. Stronger independence requires more localization of subproblems and the localization applies to Lagrange multipliers in the proposed domain decomposition method. In addition, primal nodes of the independent interface are introduced so that uncertainty of defining constraint equations at corner points has been completely eliminated for better computer implementation.

3.1 Multibody Dynamics

The Lagrange multiplier technique is an essential ingredient of multibody system modeling. Multibody dynamic analysis can be performed well when the kinematic constraint equations are well enforced via Lagrange multipliers. Lagrange multipliers have been used for constraint conditions in multibody dynamics community while the

domain decomposition method community started to use them for domain decomposition and parallel computing. The domain decomposition of a multibody system can also be done by the Lagrange multipliers just as the way multibody dynamics community has done for enforcing constraints.

3.1.1 Formulation of dynamics with constrained system

Multibody systems are characterized by mechanical joints that connect subcomponents of the system. Many connections by mechanical joints are formulated by kinematic constraint equations and they are added to the formulation in the form of algebraic equations. The formulation can lead to a set of the index-3 Differential Algebraic Equations (DAEs), see Equations (3.1) and (3.2).

$$\underline{\underline{M}}(\underline{q}, t)\underline{\ddot{q}} + \underline{\underline{B}}^T(\underline{q}, t)\underline{\lambda} = \underline{F}(\underline{q}, \underline{\dot{q}}, t), \quad (3.1)$$

$$\underline{\underline{C}}(\underline{q}, t) = \underline{0}, \quad (3.2)$$

where $\underline{\underline{M}}(\underline{q}, t)$ is the mass matrix, \underline{q} the array of generalized coordinates, $\underline{\underline{C}}(\underline{q}, t)$ the array of kinematic constraints, $\underline{\underline{B}}(\underline{q}, t)$ the constraint Jacobian matrix, $\underline{\lambda}$ the array of Lagrange multipliers, $\underline{F}(\underline{q}, \underline{\dot{q}}, t)$ the nonlinear elastic, gyroscopic and externally applied generalized forces. The notation $(\dot{\cdot})$ is used to denote a derivative with respect to time. While Equation (3.1) represents the equations of motion of the system, Equation (3.2) does the (holonomic) kinematic constraints.

3.1.2 Scaling technique and augmented Lagrangian formulation

In general, solving DAEs has been known to be much more difficult than Ordinary Differential Equations (ODEs). If the popular solution approaches to ODEs are used for solving DAEs, they usually yield erroneous results such as drift phenomenon, divergence with small time step sizes, etc. Although so many approaches have been proposed to solve DAEs such as index reduction or eliminating Lagrange multipliers from the formulation, the direct solution of index-3 DAEs has regained popularity.

The common problem of solving index-3 DAEs was about badly ill-conditioned system matrix for small time step sizes. This problem has been resolved by a physical-property-based scaling technique for DAEs, which was proposed by Bauchau *et al.* [6, 15, 22]. This approach provides a better conditioning of a system matrix, in turn, a stable time integration without the stability concerns from the time step size. The scaling technique also balances the magnitudes between generalized displacements and Lagrange multipliers; Lagrange multipliers are usually much larger than generalized displacements in magnitudes. By the scaling technique, the previous index-3 DAEs are slightly changed into the Equations (3.3) and (3.4).

$$\underline{\underline{M}} \ddot{\underline{\hat{q}}} + \underline{\underline{B}}^T s \dot{\underline{\hat{\lambda}}} = h^2 \underline{\underline{F}}, \quad (3.3)$$

$$s \underline{\underline{C}} = \underline{\underline{0}}, \quad (3.4)$$

where the *scalar* scaling factor s is defined as,

$$s = m_r + d_r h + k_r h^2, \quad (3.5)$$

and m_r , d_r and k_r represent reference mass, damping and stiffness coefficients of the system, respectively. h is the time step size. The notation $(\dot{\cdot})$ now becomes a derivative with respect to the non-dimensional time, $\tau = t/h$, and another notation $(\hat{\cdot})$ indicates generalized coordinates normalized by the reference length, l_r of the system. Because the scaling factor depends on the characteristics of a problem, it can change problem by problem. This explains and agrees with the statement “The basic recommendation is that the scaling of the equations and unknowns must proceed on a problem-by-problem basis. General scaling strategies are unreliable” by Golub and Van Loan [41].

On the other hand, the Lagrange multiplier technique to enforce kinematic constraints has experienced the *drift* phenomenon due to the numerical approximation and round-off errors as a dynamic simulation proceeds. To remedy this problem,

the penalty method has been integrated with the Lagrange multiplier technique, and this combined approach is called *Augmented Lagrangian Formulation* (ALF) [13, 14]. The ALF with appropriately chosen penalty factors makes the solution convergent within an error threshold. The ALF can be done through an iterative process which is usual for solving nonlinear problems. The ALF also provides the system matrix with *positive definiteness* which is crucial when the skyline solver should be used without pivoting. With the help of ALF, the previous scaled equations of motion are now changed into Equation (3.6), by introducing the augmented Lagrangian term $\hat{\lambda} + \frac{p}{s} \underline{\mathcal{C}}$.

$$\underline{\underline{M}} \ddot{\underline{\underline{q}}} + \underline{\underline{B}}^T s \left(\hat{\lambda} + \frac{p}{s} \underline{\mathcal{C}} \right) = h^2 \underline{\underline{F}}, \quad (3.6)$$

where p is the penalty factor. This ALF approach has been studied extensively [38, 40] and verified by Bayo *et al.* [14, 13] in order to prove that the approach is effective for the kinematic constraint enforcement in multibody dynamics.

The scaling technique and the ALF with the penalty method have helped to achieve reliable solutions to DAEs. In multibody dynamics, DAEs consist of both equations of motion and kinematic constraints. The role of kinematic constraint equations is identical to that of equality constraint equations in non-overlapping domain decomposition methods. If a structural dynamic simulation is performed with the domain decomposition, the associated governing equations are to be in the form of DAEs with no doubt. In this sense, the reliable schemes developed from the multibody community, can certainly be applied to the domain decomposition methods.

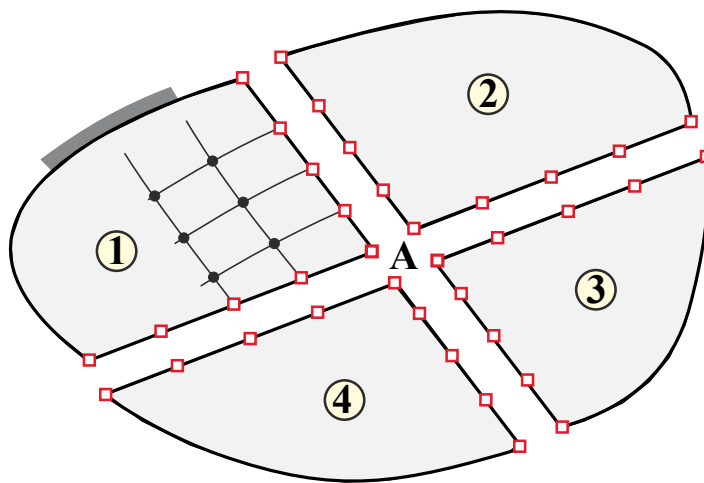
3.2 Domain Decomposition

Consider the planar solid depicted in Figure 3.1. To develop a parallel solution algorithm for this problem, the solid is partitioned into N_s non-overlapping subdomains. Each of these subdomains could themselves be multibody systems comprising both elastic elements and nonlinear kinematic constraints. For convenience, Figure 3.1 depicts a planar system, but all the developments presented here are applicable to

general, three-dimensional problems. The degrees of freedom (dofs) for each subdomain are collected in arrays denoted $\underline{u}^{(i)}$, $i = 1, 2, \dots, N_s$. Array $\underline{u}^{(i)}$ stores the dofs of subdomain i , *i.e.*, the displacement components at all the nodes of the subdomain. This array is of size $n_u^{(i)}$, which is the total number of dofs for subdomain i . Notation $(\cdot)^{(i)}$ indicates quantities associated with subdomain i . The global array of dofs is defined as

$$\underline{u}^T = \left\{ \underline{u}^{(1)T}, \underline{u}^{(2)T}, \dots, \underline{u}^{(N_s)T} \right\}. \quad (3.7)$$

Array \underline{u} is of size $n_u = \sum_{i=1}^{N_s} n_u^{(i)}$, which is the total number of dofs for the complete structure. As the original domain is divided into subdomains, the nodes along the interfaces and the associated dofs are duplicated. Consequently, array \underline{u} contains a large number of redundant dofs: all interface dofs appear twice or more times. The variables stored in array \underline{u} should be called “generalized coordinates” because they do not form a minimum set, but the term “dofs,” more widely used in the finite element literature, will be used here.



① Subdomain number • Internal node
 ■ Boundary condition □ Boundary node

Figure 3.1: Planar solid partitioned into four non-overlapping subdomains

The dofs of each subdomain can be split into two mutually exclusive groups, the

internal and boundary dofs, respectively. The boundary dofs are those that are exposed in the process of dividing the original problem into subdomains, whereas the remaining dofs are internal. Kinematic constraints will be imposed at the boundary nodes to enforce the continuity of the displacement field, thereby ensuring that the behavior of the connected subdomains is identical to that of the original, un-partitioned solid.

3.2.1 Classical Lagrange multiplier technique

The continuity of the displacement field across subdomain boundaries is enforced by imposing linear constraints, the equality of the dofs of corresponding nodes in adjacent subdomains. Typically, this is achieved by using the classical Lagrange multiplier technique, which is illustrated in a conceptual manner in Figure 3.2. Let the displacement vectors at two nodes belonging to two adjacent subdomains be denoted \underline{u}_1 and \underline{u}_2 . The continuity of the displacement field across the interface of the two subdomains implies $\underline{c} = \underline{u}_1 - \underline{u}_2 = \underline{0}$, where \underline{c} is the constraint to be imposed. In the classical Lagrange multiplier technique, the constraint is imposed via the addition of a constraint potential, $V_c = \underline{\lambda}^T \underline{c}$, where $\underline{\lambda}$ is the array of Lagrange multipliers used to enforce the constraint.

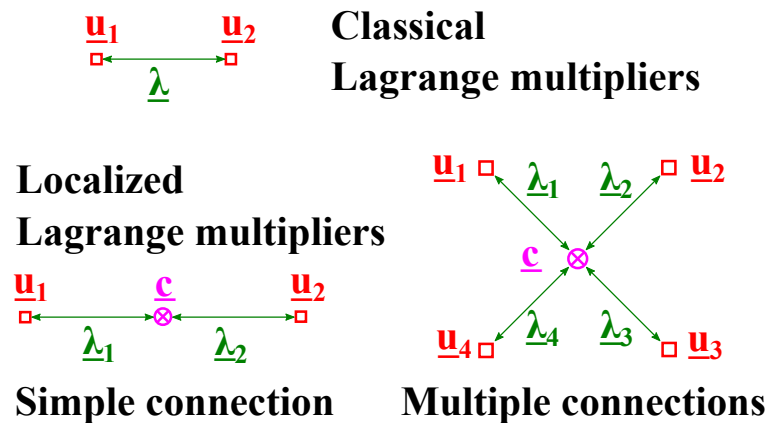


Figure 3.2: Classical and localized Lagrange multipliers.

3.2.2 Localized Lagrange multiplier technique

An alternative approach is to define an independent interface node, denoted \underline{c} , then impose two kinematic constraints: the displacement components at the boundary nodes in the two subdomains adjacent to the interface must equal those at the independent interface nodes. For the simple connection illustrated in Figure 3.2, the two kinematic constraints become $\underline{\mathcal{C}}^{[1]} = \underline{u}_1 - \underline{c} = \underline{0}$ and $\underline{\mathcal{C}}^{[2]} = \underline{u}_2 - \underline{c} = \underline{0}$, and the corresponding constraint potential is $V_c = \underline{\lambda}^{[1]T} \underline{\mathcal{C}}^{[1]} + \underline{\lambda}^{[2]T} \underline{\mathcal{C}}^{[2]}$. Notation $(\cdot)^{[j]}$ indicates quantities associated with constraint j .

In this approach, Lagrange multipliers $\underline{\lambda}^{[1]}$ enforce the constraint between the boundary dofs of subdomain 1, denoted \underline{u}_1 , and the interface dofs, \underline{c} . Similarly, Lagrange multipliers $\underline{\lambda}^{[2]}$ enforce the constraint between the boundary dofs of adjacent subdomain 2, denoted \underline{u}_2 , and the same interface dofs, \underline{c} . No direct constraint is written between the dofs of the two subdomains. Consequently, Lagrange multipliers $\underline{\lambda}^{[1]}$ and $\underline{\lambda}^{[2]}$ become “localized,” *i.e.*, $\underline{\lambda}^{[1]}$ and $\underline{\lambda}^{[2]}$ are local variables of subdomains 1 and 2, respectively. The name “localized Lagrange multiplier technique” stems from this feature of the approach. Note that constraints are localized as well: constraints $\underline{\mathcal{C}}^{[1]}$ and $\underline{\mathcal{C}}^{[2]}$ are associated with subdomains 1 and 2, respectively.

The domain decomposition process also creates corner nodes, such as that denoted **A** in Figure 3.1. Because four subdomains meet at this node, four boundary nodes were created, one for each subdomain. Note that for multiple connections, constraints and Lagrange multipliers remain localized, *i.e.*, each associated with a single subdomain. Figure 3.2 illustrates how the single interface node, denoted \underline{c} , is now connected to the four boundary nodes with the help of four sets of localized Lagrange multipliers. In finite element formulations, this approach has been used to enforce the continuity of displacement fields between adjacent incompatible elements [66]. The same approach, called “localized version of the method of Lagrange multipliers,” has been advocated by Park *et al.* [55, 56].

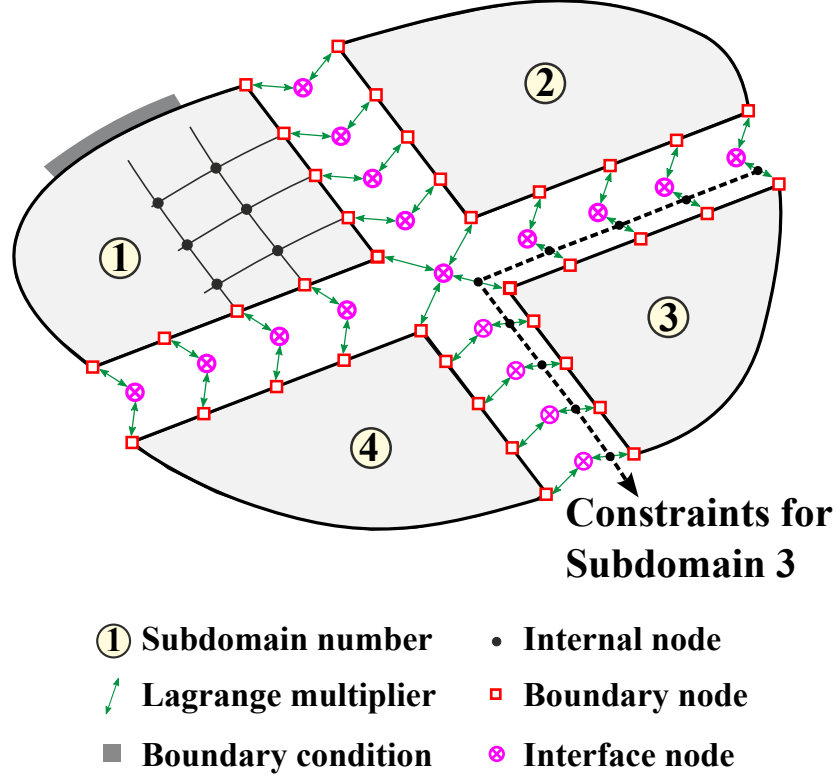


Figure 3.3: Connection through localized Lagrange multipliers.

Figure 3.3 illustrates the application of the localized Lagrange multiplier technique to enforce the continuity of the displacements field for the planar solid problem depicted in Figure 3.1. Note that each constraint and corresponding Lagrange multipliers are associated with a single subdomain unambiguously.

Let $N_b^{(i)}$ denote the total number of boundary nodes of subdomain i and $\underline{\lambda}^{[j]}$, $j = 1, 2, \dots, N_b^{(i)}$ the sets of Lagrange multipliers used to enforce the kinematic constraint at these boundary nodes. Because all Lagrange multipliers are localized, the following notation is introduced $\underline{\lambda}^{(i)T} = \left\{ \underline{\lambda}^{[1]T}, \underline{\lambda}^{[2]T}, \dots, \underline{\lambda}^{[N_b^{(i)}]T} \right\}$: array $\underline{\lambda}^{(i)}$ stores the localized Lagrange multipliers associated with subdomain i . The nodal dofs and Lagrange multipliers of subdomain i are combined into a single array

$$\tilde{\underline{u}}^{(i)T} = \left\{ \underline{u}^{(i)T}, \underline{\lambda}^{(i)T} \right\}, \quad (3.8)$$

and the array storing the dofs of all subdomains is now

$$\check{\underline{u}}^T = \left\{ \check{\underline{u}}^{(1)T}, \check{\underline{u}}^{(2)T}, \dots, \check{\underline{u}}^{(N_s)T} \right\}. \quad (3.9)$$

For the clarity of the presentation, the nodal dofs and Lagrange multipliers appear segregated in Equation (3.8), but this is not required; in practice, nodal dofs and Lagrange multipliers are interspersed so as to minimize the bandwidth of the stiffness matrix, as is done commonly in finite element implementations.

3.3 Formulation of the Problem

The method described in the previous section, called the ‘‘Localized Lagrange multiplier (LLM) technique’’ as opposed to the ‘‘Classical Lagrange multiplier (CLM) technique’’, can be combined with the scaling technique and the augmented Lagrangian formulation for robust multibody dynamics, which were already stated in Section 3.1. The LLM technique adds independent primal variables (generalized coordinates) to the interface and the dual variables (Lagrange multipliers) are now localized into subdomains as in Figure 3.4. Each equality constraint for kinematic continuity conditions between subdomain interfaces, is formulated by the LLM technique with scaling and penalty factors, as proposed by Bauchau [5]. The total potential energy of the system can be expressed as $\Pi = A + \Phi + V_c$, where A is the strain energy, Φ the potential of the externally applied loads, and V_c the potential of the constraints.

3.3.1 Strain energy of the system

The total strain energy in the structure, A , can be evaluated by summing the strain energies, $A^{(i)}$, of the various subdomains

$$A = \sum_{i=1}^{N_s} A^{(i)} = \frac{1}{2} \sum_{i=1}^{N_s} \underline{u}^{(i)T} \underline{\underline{K}}^{(i)} \underline{u}^{(i)} = \frac{1}{2} \underline{u}^T \text{diag}(\underline{\underline{K}}^{(\alpha)}) \underline{u}, \quad (3.10)$$

where $\underline{\underline{K}}^{(i)}$ is the stiffness matrix for subdomain i . The stiffness matrix of each subdomain is obtained from the stiffness matrices of each finite element of the subdomain

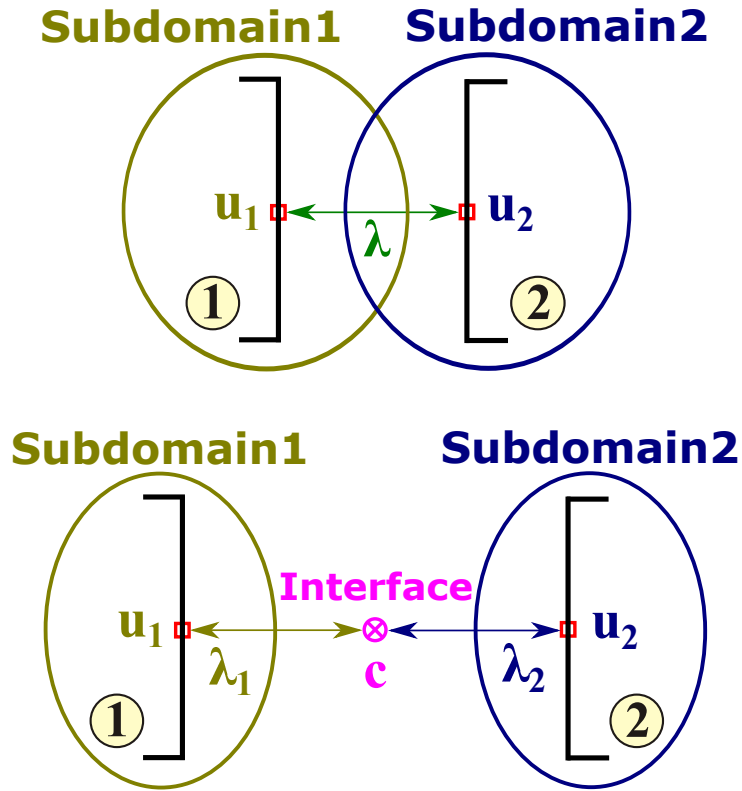


Figure 3.4: CLM method vs LLM method

using standard finite element assembly procedures [3]. The block diagonal, global stiffness matrix of the system, $\text{diag}(\underline{\underline{K}}^{(\alpha)})$, is of size $(n_u \times n_u)$. If the original solid is suitably constrained by a set of boundary conditions that prevent overall rigid body motions, the global stiffness matrix will not be singular. This property is not shared by the stiffness matrices of individual subdomains: indeed, due to the partitioning of the solid into possibly unconstrained or “floating subdomains,” the stiffness matrix of each subdomain is potentially singular.

Domain decomposition methods exploit the special structure of the global stiffness matrix. Because it is block-diagonal, its inverse is computed readily as $\text{diag}(K^{(\alpha)-1})$. Since the subdomain stiffness matrices are independent of each other, their inverses can be computed by N_s processors independently.

For subsequent developments, it will be necessary to express the total strain energy

in terms of array $\check{\underline{u}}$ defined in Equation (3.9) to find

$$A = \frac{1}{2} \check{\underline{u}}^T \text{diag}(\check{\underline{K}}^{(\alpha)}) \check{\underline{u}}, \quad (3.11)$$

where the subdomain stiffness matrix is now

$$\check{\underline{K}}^{(i)} = \begin{bmatrix} \underline{K}^{(i)} & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix}. \quad (3.12)$$

3.3.2 Potential of the externally applied loads

The total potential of the externally applied loads, Φ , is found by summing up the potentials for each of the subdomains

$$\Phi = \sum_{i=1}^{N_s} \Phi^{(i)} = - \sum_{i=1}^{N_s} \underline{u}^{(i)T} \underline{Q}^{(i)} = -\underline{u}^T \underline{Q} = -\check{\underline{u}}^T \check{\underline{Q}}, \quad (3.13)$$

where $\underline{Q}^{(i)}$ is the load array for subdomain i . The system's global load array is defined as $\underline{Q}^T = \left\{ \underline{Q}^{(1)T}, \underline{Q}^{(2)T}, \dots, \underline{Q}^{(N_s)T} \right\}$ and array $\check{\underline{Q}}$ is expanded as $\check{\underline{Q}}^T = \left\{ \underline{Q}^{(1)T}, \underline{0}^T, \underline{Q}^{(2)T}, \underline{0}^T, \dots, \underline{Q}^{(N_s)T}, \underline{0}^T \right\}$.

3.3.3 Potential of a typical constraint

As discussed in Section 3.2.2, the kinematic continuity conditions between subdomain interfaces is enforced via the localized Lagrange multiplier technique. Let $\underline{u}_b^{[j]}$ and $\underline{c}^{[j]}$ denote the arrays of dofs at a boundary node and at an interface node, respectively. Kinematic constraint j is written as $\underline{c}^{[j]} = \underline{u}_b^{[j]} - \underline{c}^{[j]} = \underline{0}$ and the associated potential is

$$V_c^{[j]} = s \underline{\lambda}^{[j]T} \underline{c}^{[j]} + \frac{p}{2} \underline{c}^{[j]T} \underline{c}^{[j]}, \quad (3.14)$$

where $\underline{\lambda}^{[j]}$ is the array of Lagrange multipliers used to enforce the constraint, and s the scaling factor for those multipliers. The second term of the potential is a penalty term and p is the penalty coefficient. The potential defined by Equation (3.14) combines the localized Lagrange multiplier technique with the penalty method. This

combination is known as the augmented Lagrangian formulation and has been studied extensively [38, 40]. It is an effective approach for the enforcement of kinematic constraints in multibody dynamics, as proposed by Bayo *et al.* [13, 14]. Furthermore, Bottasso *et al.* [15, 6] have proved that scaling of the Lagrange multipliers and the addition of penalty terms help the solution of the differential-algebraic equations that characterize flexible multibody systems.

A variation of the potential defined by Equation (3.14) is obtained easily,

$$\begin{aligned} \delta V_c^{[j]} = & \delta \underline{u}_b^{[j]T} \left[s \underline{\lambda}^{[j]} + p \underline{C}^{[j]} \right] + \delta \underline{\lambda}^{[j]T} \left[s \underline{C}^{[j]} \right] \\ & + \delta \underline{c}^{[j]T} \left[-s \underline{\lambda}^{[j]} - p \underline{C}^{[j]} \right], \end{aligned} \quad (3.15)$$

and gives rise to the following generalized forces of constraint,

$$\underline{f}^{[j]} = \begin{Bmatrix} s \underline{\lambda}^{[j]} + p \underline{C}^{[j]} \\ s \underline{C}^{[j]} \\ -s \underline{\lambda}^{[j]} - p \underline{C}^{[j]} \end{Bmatrix}. \quad (3.16)$$

Taking a derivative of these forces of constraint with respect to the dofs yields the stiffness matrix of the constraint,

$$\underline{k}^{[j]} = \begin{bmatrix} p \underline{I} & s \underline{I} & -p \underline{I} \\ s \underline{I} & \underline{0} & -s \underline{I} \\ -p \underline{I} & -s \underline{I} & p \underline{I} \end{bmatrix}, \quad (3.17)$$

where \underline{I} denotes the identity matrix of size $d \times d$. For the planar problem illustrated in Figure 3.1, each node features two dofs, and hence, $d = 2$; for three-dimensional problems, $d = 3$, and for beam problems, $d = 6$, because each node has six dofs, three displacements and three rotations.

As mentioned in Section 3.2.2, the Lagrange multipliers become localized in the proposed formulation, *i.e.*, Lagrange multipliers are associated with one subdomain unequivocally. The potential of kinematic constraint involves two types of dofs, the subdomain dofs, $\underline{u}_b^{[j]}$ and $\underline{\lambda}^{[j]}$, and the interface dofs, $\underline{c}^{[j]}$. The constraint forces and

stiffness matrix are partitioned to reflect this fact

$$\underline{f}^{[j]} = \begin{Bmatrix} \underline{f}_b^{[j]} \\ \underline{f}_c^{[j]} \end{Bmatrix}, \quad \underline{k}^{[j]} = \begin{bmatrix} \underline{k}_{bb}^{[j]} & \underline{k}_{bc}^{[j]} \\ \underline{k}_{bc}^{[j]T} & \underline{k}_{cc}^{[j]} \end{bmatrix}. \quad (3.18)$$

Subscripts $(\cdot)_b$ and $(\cdot)_c$ denote dofs associated with boundary and interface nodes, respectively. Partitioning the constraint forces defined by Equation (3.16) yields

$$\underline{f}_b^{[j]} = \begin{Bmatrix} s\underline{\lambda}^{[j]} + p\underline{\mathcal{C}}^{[j]} \\ s\underline{\mathcal{C}}^{[j]} \end{Bmatrix}, \quad \underline{f}_c^{[j]} = - \begin{Bmatrix} s\underline{\lambda}^{[j]} + p\underline{\mathcal{C}}^{[j]} \end{Bmatrix}. \quad (3.19)$$

A similar operation for the constraint stiffness matrix leads to

$$\underline{k}_{bb}^{[j]} = \begin{bmatrix} p\underline{I} & s\underline{I} \\ s\underline{I} & \underline{0} \end{bmatrix}, \quad \underline{k}_{cc}^{[j]} = \begin{bmatrix} p\underline{I} \end{bmatrix}, \quad \underline{k}_{bc}^{[j]} = \begin{bmatrix} -p\underline{I} \\ -s\underline{I} \end{bmatrix}. \quad (3.20)$$

In summary, each kinematic constraint generates an array of constraint forces and a constraint stiffness matrix. Clearly, each kinematic constraint can be viewed as finite element and in the sequel, the terms “kinematic constraint” and “constraint element” will be used interchangeably.

3.3.4 Assembly procedure for the constraints

In the previous section, the development has focused on a single constraint. As illustrated in Figure 3.3, each subdomain is connected to its neighbors via a number of boundary nodes. To connect the N_s subdomains, a total of N_c interface nodes will be defined and the following array stores the dofs at all these interface nodes,

$$\underline{c}^T = \left\{ \underline{c}_1^T, \underline{c}_2^T, \dots, \underline{c}_{N_c}^T \right\}. \quad (3.21)$$

Array \underline{c} is of size n_c . The total potential of all constraints associated with subdomain i , denoted $V_c^{(i)}$, is found by summing up the potentials of the corresponding constraint,

$$V_c^{(i)} = \sum_{j=1}^{N_b^{(i)}} V_c^{[j]}. \quad (3.22)$$

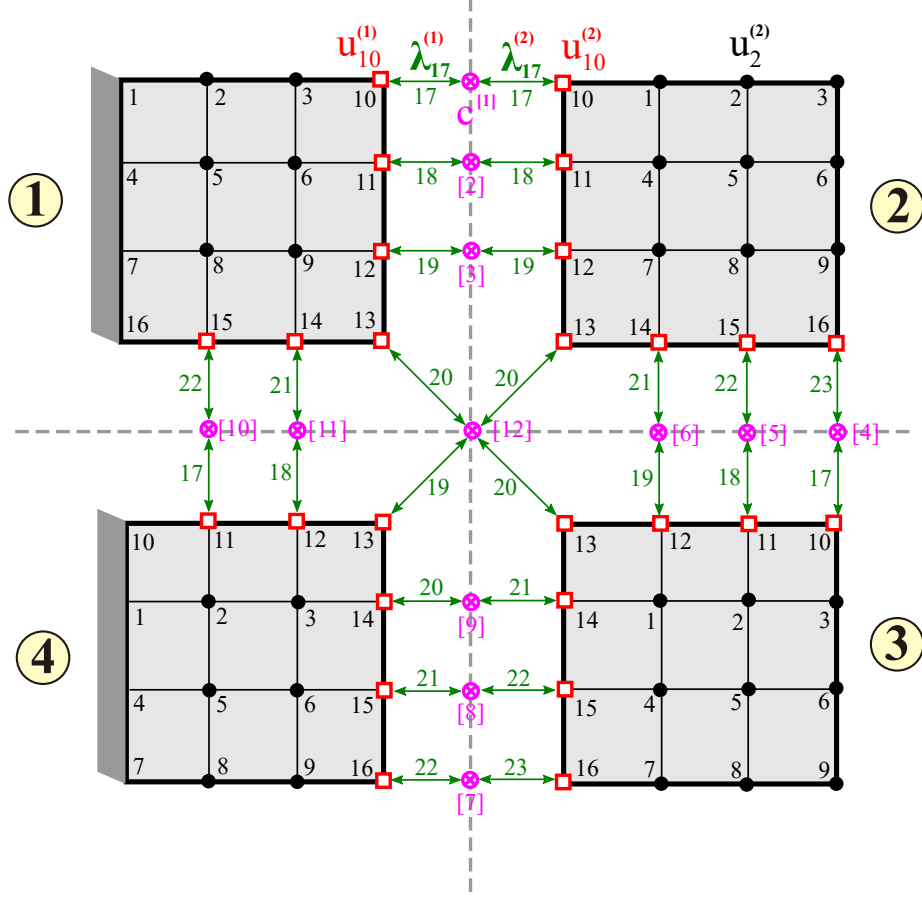


Figure 3.5: LLM domain decomposition mesh

Finally, the total potential of all kinematic constraints is

$$V_c = \sum_{i=1}^{N_s} V_c^{(i)}. \quad (3.23)$$

Each constraint element contributes constraint forces and stiffness matrices defined by Equations (3.19) and (3.20), respectively. Using the standard assembly procedure used in the finite element method [3], the force arrays and stiffness matrices generated by all the constraint elements associated with subdomain i are assembled into the following subdomain arrays and matrices

$$\check{\underline{\underline{F}}}_b^{(i)} = \sum_{j=1}^{N_b^{(i)}} \underline{\underline{B}}_b^{[j]T} \underline{f}_b^{[j]}, \quad \check{\underline{\underline{K}}}_{bb}^{(i)} = \sum_{j=1}^{N_b^{(i)}} \underline{\underline{B}}_b^{[j]T} \underline{\underline{k}}_{bb}^{[j]} \underline{\underline{B}}_b^{[j]}, \quad (3.24)$$

where $\underline{\underline{B}}_b^{[j]}$ is the Boolean matrices used for the assembly process, *i.e.*, $\underline{u}_b^{[j]} = \underline{\underline{B}}_b^{[j]} \check{\underline{u}}^{(i)}$. Of course, the assembly procedure can be performed in parallel for all subdomains.

Similarly, the constraint elements contribute force arrays and stiffness matrices to the interface problem,

$$\underline{F}_c^{(i)} = \sum_{j=1}^{N_b^{(i)}} \underline{B}_c^{[j]T} \underline{f}_c^{[j]}, \quad \underline{K}_{cc}^{(i)} = \sum_{j=1}^{N_b^{(i)}} \underline{B}_c^{[j]T} \underline{k}_{cc}^{[j]} \underline{B}_c^{[j]}, \quad (3.25)$$

where $\underline{B}_c^{[j]}$ is the Boolean matrices used for the assembly process, *i.e.*, $\underline{c}^{[j]} = \underline{B}_c^{[j]} \underline{c}$.

Finally, the constraint coupling stiffness is assembled to find

$$\underline{K}_{bc}^{(i)} = \sum_{j=1}^{N_b^{(i)}} \underline{B}_b^{[j]T} \underline{k}_{bc}^{[j]} \underline{B}_c^{[j]}. \quad (3.26)$$

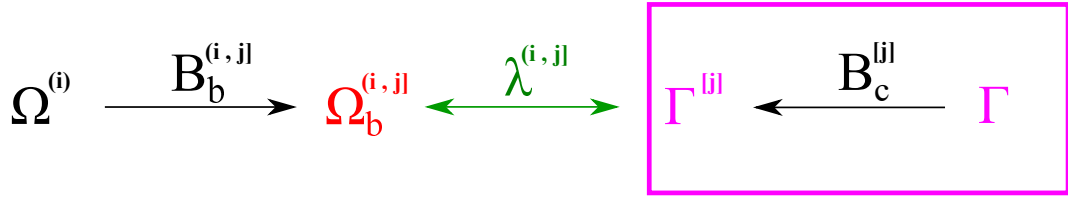


Figure 3.6: LLM domain relationship

3.3.5 Governing equations

The total potential energy of the system, $\Pi = A + \Phi + V_c$, is found by combining Equations (3.10), (3.13), and (3.23), and the principle of minimum total potential energy then yields the governing equations as

$$\begin{bmatrix} \text{diag}(\check{\underline{K}}^{(\alpha)} + \check{\underline{K}}_{bb}^{(\alpha)}) & \check{\underline{K}}_{bc}^{(\alpha)} \\ \check{\underline{K}}_{bc}^{T} & \check{\underline{K}}_{cc}^{(\alpha)} \end{bmatrix} \begin{Bmatrix} \check{\underline{u}} \\ \check{\underline{c}} \end{Bmatrix} = \begin{Bmatrix} \check{\underline{Q}} - \check{\underline{F}}_b \\ -\check{\underline{F}}_c \end{Bmatrix}, \quad (3.27)$$

where arrays $\check{\underline{F}}_b$ and \underline{F}_b are the assembly of their subdomain counterparts, $\check{\underline{F}}_b^{(i)}$ and $\underline{F}_c^{(i)}$, respectively, $\underline{K}_{cc} = \sum_{i=1}^{N_s} \underline{K}_{cc}^{(i)}$ and

$$\underline{K}_{bc}^T = \left[\underline{K}_{bc}^{(1)T}, \underline{K}_{bc}^{(2)T}, \dots, \underline{K}_{bc}^{(N_s)T} \right]. \quad (3.28)$$

The block-diagonal nature of the leading entry of the system matrix makes this approach amenable to parallel solution algorithms.

3.4 Physical Interpretation of the Proposed Approach

It is interesting to give a physical interpretation of the various matrices appearing in Equation (3.27). Matrix $\check{\underline{\underline{K}}}_{bb}^{(\alpha)}$ is the assembly of its counterparts defined for each constraint element by Equation (3.20). For each constraint element, the leading entry of matrix $\underline{\underline{k}}_{bb}^{[j]}$ is a diagonal matrix, $p\underline{\underline{I}}$, which is added to the diagonal entries of stiffness matrix $\underline{\underline{K}}^{(i)}$ associated with the boundary nodes. Physically, this corresponds to adding springs of stiffness constant p connected to the ground at each boundary node of subdomain i .

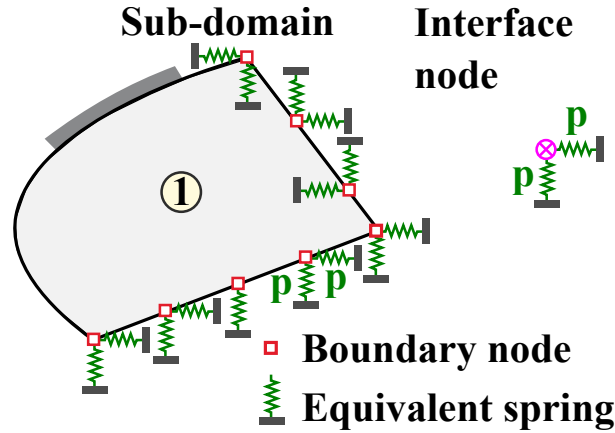


Figure 3.7: Physical interpretation of the penalty terms.

Figure 3.7 shows the structure characterized by stiffness matrix $\check{\underline{\underline{K}}}^{(1)} + \check{\underline{\underline{K}}}_{bb}^{(1)}$: the first subdomain of the planar structure shown in Figure 3.1 is now connected to the ground at each boundary node by two springs of stiffness constant that equal the penalty coefficient. Whereas matrix $\check{\underline{\underline{K}}}^{(i)}$ is singular for any floating subdomain, $\check{\underline{\underline{K}}}^{(i)} + \check{\underline{\underline{K}}}_{bb}^{(i)}$ is not.

Similarly, matrix $\underline{\underline{K}}_{cc}$ is the assembly of its counterparts defined for each constraint element by Equation (3.20). Here again, matrix $\underline{\underline{k}}_{cc}^{[j]}$ is a diagonal matrix, $p\underline{\underline{I}}$, which corresponds to connecting each interface node to the ground by springs of stiffness constant p , as illustrated in Figure 3.7. At corner points, the interface node is connected to the ground in parallel by several springs of stiffness constant p .

As expected, the Lagrange multipliers can be interpreted as the forces that interconnect the various parts of the structure. At convergence, all kinematic constraints will be satisfied, $\underline{c}^{[j]} = \underline{0}$, and the constraint forces defined in Equation (3.16) reduce to equal and opposite forces, $\pm s\underline{\lambda}^{[j]}$, applied to the boundary and interface nodes, as expected from Newton's third law; this is illustrated in Figure 3.8.

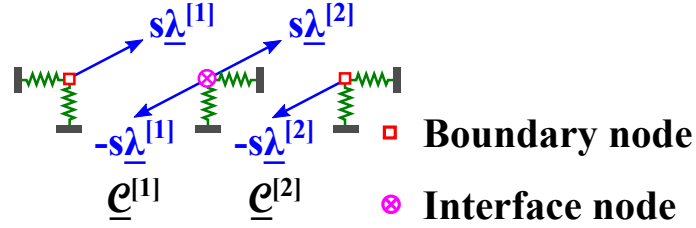


Figure 3.8: Physical interpretation of the Lagrange multipliers

Consider now two kinematic constraints, say $\underline{c}^{[1]}$ and $\underline{c}^{[2]}$, imposing the continuity of the displacement field at two adjacent boundary nodes, as shown in Figure 3.8. The connections between the boundary nodes and the common interface node are enforced by means of pairs of distinct constraint forces, $\pm s\underline{\lambda}^{[1]}$ and $\pm s\underline{\lambda}^{[2]}$, as illustrated in the figure. It is easy to prove that the system (3.27) also imposes the constraint $\underline{\lambda}^{[1]} = \underline{\lambda}^{[2]}$: this equation imposes Newton's third law for the forces acting on adjacent subdomains. While the Lagrange multipliers are localized, *i.e.*, while $\underline{\lambda}^{[1]}$ and $\underline{\lambda}^{[2]}$ are variables associated with distinct subdomains, the governing equations (3.27) impose Newton's third law across subdomain boundaries.

The proposed method involves primal variables: the nodal displacement is all subdomains and the displacements at the interface nodes. It also involves dual variables: the forces at the subdomain interfaces represented by localized Lagrange multipliers. The interface problem is expressed in terms of primal variables only, the displacements at the interface nodes. Consequently, the proposed method falls in the category of primal methods. In contrast, the original FETI [29] method is a dual method, and the more recent FETI-DP [27] is a dual-primal method.

These approaches were developed to overcome the shortcoming of earlier primal

methods. Typically, in primal methods, the interface problem is very poorly conditioned, limiting the use of these approaches. In the present approach, however, the stiffness of the interface problem is not dictated by the physical characteristics of the structure, but rather by the user selected penalty factor. If the penalty factor is identical for all constraint elements, the interface problem is very well conditioned. These advantages stem from the combined use of the Lagrange multiplier technique and penalty method. Because the Lagrange multiplier technique enforces the constraints exactly, it is not necessary to select a “very large penalty factor” to obtain accurate solutions. The penalty term conditions the problem, but does not limit the accuracy of the solution.

3.5 Solution Procedure with Factorization

There are two classes of solving linear system of equations. One is direct methods and the other is iterative methods. Direct methods like Gauss eliminations and LU factorization solve the system of equations for exact solution within machine accuracy in a finite number of steps. On the other hand, iterative methods solve for approximate solution converging to the exact solution with iterations.

Direct methods have several advantages against iterative methods. Multiple right hand side load arrays, i.e. multiple loading cases, can be easily dealt with only by the forward- and back-substitutions, once the factorization is done at a time step. Unlike iterative methods, the direct methods are not sensitive to the conditioning of the system matrix. The solution of iterative methods converges, over iteration, fast or slow based on the condition number of the system matrix, but the direct methods do not. Thus a direct method doesn't care the condition number and this is very advantageous especially in flexible multibody dynamics where the system matrix might be often badly ill-conditioned due to the kinematic constraints and the rigid body modes.

In the previous sections, the concept of LLM technique has been introduced and the advantages of it were stressed in contrast to CLM technique, for parallel computations in multibody dynamics. This proposed domain decomposition approach using LLM technique has been implemented in Dymore, a finite element based multibody dynamics code for comprehensive analysis of rotorcraft, for better validation of the approach on general multibody structures such as rotorcrafts, wind turbines, satellite solar panels, etc. It is also important to note that Dymore uses skyline solver to solve a linearized system of equations from the nonlinear DAEs with the finite element based formulation. In order to use the LLM technique in finite element formulation, an LLM is defined to be an independent finite element to impose the equality constraint between an interface node and a boundary node of the associated subdomain.

3.5.1 Expected block matrices from LU factorization

In this section, a general solution procedure for block-diagonal systems is presented. For simplicity, the linear system is rewritten as

$$\underline{\underline{A}} \underline{x} = \underline{b}, \quad (3.29)$$

where \underline{x} is the array of unknowns, \underline{b} the known right-hand side, and matrix $\underline{\underline{A}}$ has the following form

$$\underline{\underline{A}} = \begin{bmatrix} \underline{\underline{A}}^{(1)} & \underline{0} & \underline{0} & \dots & \underline{0} & \underline{\underline{A}}_{bc}^{(1)} \\ \underline{0} & \underline{\underline{A}}^{(2)} & \underline{0} & \dots & \underline{0} & \underline{\underline{A}}_{bc}^{(2)} \\ \underline{0} & \underline{0} & \underline{\underline{A}}^{(3)} & \dots & \underline{0} & \underline{\underline{A}}_{bc}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \underline{0} & \vdots \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{\underline{A}}^{(N_s)} & \underline{\underline{A}}_{bc}^{(N_s)} \\ \underline{\underline{A}}_{bc}^{(1)T} & \underline{\underline{A}}_{bc}^{(2)T} & \underline{\underline{A}}_{bc}^{(3)T} & \dots & \underline{\underline{A}}_{bc}^{(N_s)T} & \underline{\underline{A}}_{cc} \end{bmatrix}. \quad (3.30)$$

The procedure described in the previous section leads to system matrices presenting the structure shown in Equation (3.30).

System (3.29) will be solved using the classical skyline solver [3], which is based on the factorization of the system matrix as

$$\underline{\underline{A}} = \underline{\underline{L}}\underline{\underline{U}}, \quad (3.31)$$

where $\underline{\underline{L}}$ and $\underline{\underline{U}}$ are lower and upper triangular matrices, respectively. A fundamental property of the skyline solver is that the skylines of matrices $\underline{\underline{L}}$ and $\underline{\underline{U}}$ are identical to that of matrix $\underline{\underline{A}}$. This implies that matrices $\underline{\underline{L}}$ and $\underline{\underline{U}}$ have the following structure

$$\underline{\underline{L}} = \begin{bmatrix} \underline{\underline{L}}^{(1)} & \underline{\underline{0}} & \underline{\underline{0}} & \dots & \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{L}}^{(2)} & \underline{\underline{0}} & \dots & \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{L}}^{(3)} & \dots & \underline{\underline{0}} & \underline{\underline{0}} \\ \vdots & \vdots & \vdots & \ddots & \underline{\underline{0}} & \vdots \\ \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{L}}^{(N_s)} & \underline{\underline{0}} \\ \underline{\underline{M}}^{(1)} & \underline{\underline{M}}^{(2)} & \underline{\underline{M}}^{(3)} & \dots & \underline{\underline{M}}^{(N_s)} & \underline{\underline{L}}_{cc} \end{bmatrix}, \quad (3.32)$$

$$\underline{\underline{U}} = \begin{bmatrix} \underline{\underline{U}}^{(1)} & \underline{\underline{0}} & \underline{\underline{0}} & \dots & \underline{\underline{0}} & \underline{\underline{V}}^{(1)} \\ \underline{\underline{0}} & \underline{\underline{U}}^{(2)} & \underline{\underline{0}} & \dots & \underline{\underline{0}} & \underline{\underline{V}}^{(2)} \\ \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{U}}^{(3)} & \dots & \underline{\underline{0}} & \underline{\underline{V}}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \underline{\underline{0}} & \vdots \\ \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{U}}^{(N_s)} & \underline{\underline{V}}^{(N_s)} \\ \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \dots & \underline{\underline{0}} & \underline{\underline{U}}_{cc} \end{bmatrix}. \quad (3.33)$$

Note that matrices $\underline{\underline{M}}^{(i)}$ and $\underline{\underline{V}}^{(i)}$, $i = 1, 2, \dots, N_s$ are, in general, fully populated matrices.

3.5.2 Details of the factorization procedure

The goal of the factorization procedure is to evaluate all the entries of the lower triangular matrix, $\underline{\underline{L}}$ and upper triangular matrix, $\underline{\underline{U}}$ defined by Equations (3.32) and (3.33), respectively. The factorization expressed by Equation (3.31) implies

$$\underline{\underline{A}}^{(i)} = \underline{\underline{L}}^{(i)}\underline{\underline{U}}^{(i)}, \quad i = 1, 2, \dots, N_s. \quad (3.34)$$

Clearly, the matrices associated with each subdomain, $\underline{\underline{A}}^{(i)}$, can be factorized independently to find lower and upper triangular matrices $\underline{\underline{L}}^{(i)}$ and $\underline{\underline{U}}^{(i)}$, respectively, using the classical skyline solver. Equation (3.31) also implies

$$\underline{\underline{A}}_{bc}^{(i)} = \underline{\underline{L}}^{(i)} \underline{\underline{V}}^{(i)} \quad , \quad i = 1, 2, \dots, N_s, \quad (3.35a)$$

$$\underline{\underline{A}}_{bc}^{(i)} = \underline{\underline{U}}^{(i)T} \underline{\underline{M}}^{(i)T}, \quad i = 1, 2, \dots, N_s. \quad (3.35b)$$

Once matrices $\underline{\underline{L}}^{(i)}$ and $\underline{\underline{U}}^{(i)}$ have been obtained, eqs (3.35a) and (3.35b) allow the evaluation of matrices $\underline{\underline{V}}^{(i)}$ and $\underline{\underline{M}}^{(i)}$, respectively, using back-substitution. Here again, these operations can be carried out in parallel in each subdomain.

The last relationship implied by Equation (3.31) is $\underline{\underline{A}}_{cc} = \sum_{i=1}^{N_s} \underline{\underline{M}}^{(i)} \underline{\underline{V}}^{(i)} + \underline{\underline{L}}_{cc} \underline{\underline{U}}_{cc}$, which lead to the following factorization

$$\bar{\underline{\underline{A}}}_{cc} = \underline{\underline{L}}_{cc} \underline{\underline{U}}_{cc}, \quad (3.36)$$

where

$$\bar{\underline{\underline{A}}}_{cc} = \underline{\underline{A}}_{cc} - \sum_{i=1}^{N_s} \underline{\underline{A}}_{cc}^{(i)}, \quad (3.37)$$

and

$$\underline{\underline{A}}_{cc}^{(i)} = \underline{\underline{M}}^{(i)} \underline{\underline{V}}^{(i)}, \quad i = 1, 2, \dots, N_s. \quad (3.38)$$

Matrices $\underline{\underline{A}}_{cc}^{(i)}$ can be computed in parallel. All subdomain contributions are then collected to form matrix $\bar{\underline{\underline{A}}}_{cc}$, the factorization of which yields lower and upper triangular matrices $\underline{\underline{L}}_{cc}$ and $\underline{\underline{U}}_{cc}$, respectively, using the classical skyline solver. This operation completes the factorization of the system matrix according to Equation (3.31).

3.5.3 Details of the forward- and back-substitution procedures

Once the system matrix has been factorized, the solution is obtained via forward- and back-substitution. The first step of the procedure is to write the system (3.29) as $\underline{\underline{L}} \underline{\underline{y}} = \underline{\underline{b}}$, where intermediate solution array $\underline{\underline{y}}$ is defined as $\underline{\underline{y}} = \underline{\underline{U}} \underline{\underline{x}}$. This array is partitioned as $\underline{\underline{y}}^T = \{\underline{\underline{y}}^{(1)T}, \underline{\underline{y}}^{(2)T}, \dots, \underline{\underline{y}}_c^T\}$, where $\underline{\underline{y}}^{(i)}$, $i = 1, 2, \dots, N_s$, are the

intermediate solution arrays in each of the subdomains and \underline{y}_c the corresponding interface quantities.

Given the structure of lower triangular matrix $\underline{\underline{L}}$ expressed by Equation (3.32), forward-substitution yields the components of intermediate solution array \underline{y} as

$$\underline{\underline{L}}^{(i)} \underline{y}^{(i)} = \underline{b}^{(i)}, \quad i = 1, 2, \dots, N_s, \quad (3.39a)$$

$$\underline{\underline{L}}_{cc} \underline{y}_c = \underline{b}_d, \quad (3.39b)$$

where

$$\underline{d}^{(i)} = \underline{\underline{M}}^{(i)} \underline{y}^{(i)}, \quad (3.40)$$

$$\underline{b}_d = \underline{b}_c - \sum_{i=1}^{N_s} \underline{d}^{(i)}, \quad (3.41)$$

and the right-hand side array was partitioned as $\underline{b}^T = \{\underline{b}^{(1)T}, \underline{b}^{(2)T}, \dots, \underline{b}^T\}$. While Equations (3.39a) and (3.40) are performed in parallel for each subdomain, Equation (3.39b) is not for the interface.

Finally, the solution of the problem is obtained from back-substitution

$$\underline{\underline{U}}_{cc} \underline{x}_c = \underline{y}_c, \quad (3.42a)$$

$$\underline{\underline{U}}^{(i)} \underline{x}^{(i)} = \underline{y}_v^{(i)}, \quad i = 1, 2, \dots, N_s \quad (3.42b)$$

where

$$\underline{y}_v^{(i)} = \underline{y}^{(i)} - \underline{\underline{V}}^{(i)} \underline{x}_c. \quad (3.43)$$

Again, while Equation (3.42b) is performed in parallel for each each subdomain, Equation (3.42a) is not for the interface.

3.5.4 Summary of the solution procedure

The input to the factorization procedure are as follows.

1. Stiffness matrices $\underline{\underline{A}}^{(i)}$ and $\underline{\underline{A}}_{bc}^{(i)}$, $i = 1, 2, \dots, N_s$, for each of the subdomains.

Matrices $\underline{\underline{A}}^{(i)}$ are in compact storage form. According to Equation (3.20), each

constraint applied to subdomain i generates two non-vanishing entries in $\underline{\underline{A}}_{bc}^{(i)}$; all other columns of this matrix vanish. Hence, these matrices $\underline{\underline{A}}_{bc}^{(i)}$ are not assembled. The columns of matrix $\underline{\underline{A}}_{bc}^{(i)}$ featuring non-vanishing entries are called the “active columns” of that matrix.

2. Interface stiffness matrix $\underline{\underline{A}}_{cc}$. This matrix is in compact storage form.

In summary, the solution procedure can be divided into the five phases detailed below, three of which are parallelized easily.

- Phase 1: factorize subdomain stiffness matrices (parallel)

For each subdomain, perform the following operations in parallel.

1. Perform the factorization of matrix $\underline{\underline{A}}^{(i)}$ expressed by Equation (3.34) to find matrices $\underline{\underline{L}}^{(i)}$ and $\underline{\underline{U}}^{(i)}$. Matrices $\underline{\underline{L}}^{(i)}$ and $\underline{\underline{U}}^{(i)}$ are computed “in place,” *i.e.*, they replace matrix $\underline{\underline{A}}^{(i)}$ as the computation proceeds, without additional storage requirement.
2. Evaluate matrix $\underline{\underline{V}}^{(i)}$ with the help of Equation (3.35a). Each column of this matrix can be found from the corresponding column of matrix $\underline{\underline{A}}_{bc}^{(i)}$ using back-substitution. Clearly, only the active columns of matrix $\underline{\underline{A}}_{bc}^{(i)}$ generate non-vanishing entries in matrix $\underline{\underline{V}}^{(i)}$, *i.e.*, the active columns of matrix $\underline{\underline{V}}^{(i)}$ match those of matrix $\underline{\underline{A}}_{bc}^{(i)}$. Storage must be provided for the active columns of matrix $\underline{\underline{V}}^{(i)}$ only.
3. Evaluate matrix $\underline{\underline{M}}^{(i)}$ with the help of Equation (3.35b). Each row of this matrix can be found from the corresponding column of matrix $\underline{\underline{A}}_{bc}^{(i)}$ using back-substitution. Storage must be provided for the active rows of matrix $\underline{\underline{M}}^{(i)}$ only.
4. Evaluate matrix $\underline{\underline{A}}_{cc}^{(i)}$ defined by Equation (3.38). Storage must be provided for this matrix.

In steps 2 and 3, the columns of matrix $\underline{\underline{V}}^{(i)}$ and rows of matrix $\underline{\underline{M}}^{(i)}$, respectively, can all be evaluated independently, providing fine grain parallelization opportunities.

- Phase 2: factorize interface stiffness matrix

To complete the factorization of the system matrix, perform the following operations dealing with the interface stiffness matrix.

1. Evaluate matrix $\underline{\underline{\bar{A}}}_{cc}$ using Equation (3.37).
2. Factorize matrix $\underline{\underline{\bar{A}}}_{cc}$ according to Equation (3.36).

- Phase 3: forward-substitute in subdomains (parallel)

Once the system matrix factorization has been completed, the forward-substitution phase can begin.

1. Find the intermediate solution array, $\underline{y}^{(i)}$, in each of the subdomains via forward-substitution using Equation (3.39a). Vector $\underline{y}^{(i)}$ is computed “in place,” *i.e.*, it replaces vector $\underline{b}^{(i)}$ as the computation proceeds without additional storage requirement.
2. Compute the contribution of the subdomain to interface forces, $\underline{d}^{(i)}$, using Equation (3.40). Storage must be provided for vector $\underline{d}^{(i)}$.

- Phase 4: solve for interface displacements

The solution of the interface problem proceeds in three steps.

1. Evaluate \underline{b}_d of Equation (3.41) using the intermediate solution arrays, $\underline{y}^{(i)}$, computed in the previous phase. Accumulate subdomain vectors $\underline{d}^{(i)}$ in place in array \underline{b}_c .
2. Find array \underline{y}_c via forward-substitution.

3. Evaluate interface displacements from Equation (3.42a) by back-substitution.

- Phase 5: solve for subdomain displacements by back-substitution (parallel)

In the last phase of the solution process, the subdomain nodal displacements are recovered via back-substitution.

1. Evaluate subdomain arrays $\underline{y}^{(i)}$ from Equation (3.39a).

2. Evaluate $\underline{y}_v^{(i)}$ using Equation (3.43).

3. Find subdomain nodal displacements, $\underline{x}^{(i)}$, via back-substitution using Equation (3.42b)

This solution procedure is also organized in Table 3.1 with inputs and outputs for each sub-phase.

Table 3.1: Phases of solution procedure

5 phases		Sub-phases	
Phase	Description	Sub-phase	Inputs and Outputs
Phase 1	Each subdomain: Assembly and Factorization (Parallel)	Computation and assembly of element matrices and arrays	Out: $\underline{\underline{A}}^{(i)}, \underline{\underline{b}}^{(i)}, \underline{\underline{b}}_c^{(i)}$
		LU factorization of subdomain matrix	In: $\underline{\underline{A}}^{(i)}$; Out: $\underline{\underline{L}}^{(i)}, \underline{\underline{U}}^{(i)}$
		Computation of partial sbd-int coupling matrices	In: $\underline{\underline{A}}_{bc}^{(i)}$; Out: $\underline{\underline{V}}^{(i)}, \underline{\underline{M}}^{(i)}$
		Computation of partial interface matrices	Out: $\underline{\underline{A}}_{cc}^{(i)}$
Phase 2	Interface: Assembly and Factorization (Sequential)	Data transfer (Subdomains \rightarrow Interface)	Transfer: $\underline{\underline{A}}_{cc}^{(i)}, \underline{\underline{b}}_c^{(i)}$
		Adding up partial contributions from subdomains	In: $\underline{\underline{A}}_{cc}^{(i)}, \underline{\underline{b}}_c^{(i)}$; Out: $\underline{\underline{A}}_{cc}, \underline{\underline{b}}_c$
		LU factorization of interface matrix	In: $\underline{\underline{A}}_{cc}$; Out: $\underline{\underline{L}}_{cc}, \underline{\underline{U}}_{cc}$
Phase 3	Each subdomain: Forward-substitution (Parallel)	Forward-substitution for intermediate subdomain equations	In: $\underline{\underline{L}}^{(i)}, \underline{\underline{b}}^{(i)}$; Out: $\underline{\underline{y}}^{(i)}$
		Computation for $\underline{\underline{d}}^{(i)}$	In: $\underline{\underline{M}}^{(i)}, \underline{\underline{y}}^{(i)}$; Out: $\underline{\underline{d}}^{(i)}$
Phase 4	Interface: Forward- and back- substitutions (Sequential)	Data transfer (Subdomains \rightarrow Interface)	Transfer: $\underline{\underline{d}}^{(i)}$
		Computation for $\underline{\underline{b}}_d$	In: $\underline{\underline{b}}_c^{(i)}, \underline{\underline{d}}^{(i)}$; Out: $\underline{\underline{b}}_d$
		Forward-substitution for intermediate interface equations	In: $\underline{\underline{L}}_{cc}, \underline{\underline{b}}_d$; Out: $\underline{\underline{y}}_c$
		Back-substitution for interface equations	In: $\underline{\underline{U}}_{cc}, \underline{\underline{y}}_c$; Out: $\underline{\underline{x}}_c$
		Data transfer (Interface \rightarrow Subdomains)	Transfer: $\underline{\underline{x}}_c$
Phase 5	Each subdomain: Back-substitution (Parallel)	Computation for $\underline{\underline{y}}_v^{(i)}$	In: $\underline{\underline{y}}^{(i)}, \underline{\underline{V}}^{(i)}, \underline{\underline{x}}_c$; Out: $\underline{\underline{y}}_v^{(i)}$
		Back-substitution for subdomain equations	In: $\underline{\underline{U}}^{(i)}, \underline{\underline{y}}_v^{(i)}$; Out: $\underline{\underline{x}}^{(i)}$

CHAPTER IV

PARALLEL IMPLEMENTATION

Practical implementation of the parallel algorithm with the proposed domain decomposition within the finite element framework, will be presented in this chapter. An existing sequential analysis computer program for finite element based multibody system, can be modified for the parallel counterpart. A domain decomposition process and a solution process are added and modified. Parallel programming is much more challenging than the sequential counterpart. It always requires that programmers think simultaneous flows of the algorithm on multiple processors. Thus, the implementation must be very carefully performed. To accommodate the two big processes for efficient parallel computation, data structures must be also efficiently designed in organized way.

4.1 Parallel Finite Element Procedure

The finite element procedure is largely divided into the three main steps: preprocessing, analysis and postprocessing. The preprocessing step can also be subdivided into three sub-processes: parsing user inputs, checking the consistency of a model and finite element meshing. Once the preprocessing is done and the data for finite element discretization is all ready, the analysis step can proceed. Depending on problems, the type of analysis would be a static or dynamic analysis. The problems of multibody systems are usually nonlinear and the nonlinear analysis for such problems must proceed with an iterative process such as Newton-Raphson iteration. Once the simulation through a static or dynamic analysis is done, postprocessing can proceed with the resulting data from the analysis step. This series of processes is usually performed in most sequential finite element analysis programs.

To transform the existing sequential program into a parallel version, all the three main steps can be modified to get the best parallel performance. But in this thesis, only the preprocessing and analysis steps are modified to check the parallel performance of the proposed domain decomposition method. Two new steps are added and one existing step is modified as seen in Figure 4.1. The two new steps are the domain decomposition and the adjustment for an existing finite element mesh. The existing analysis step of a sequential program needs to be considerably modified to enable parallel computations with the help of parallel library for interprocess communication.

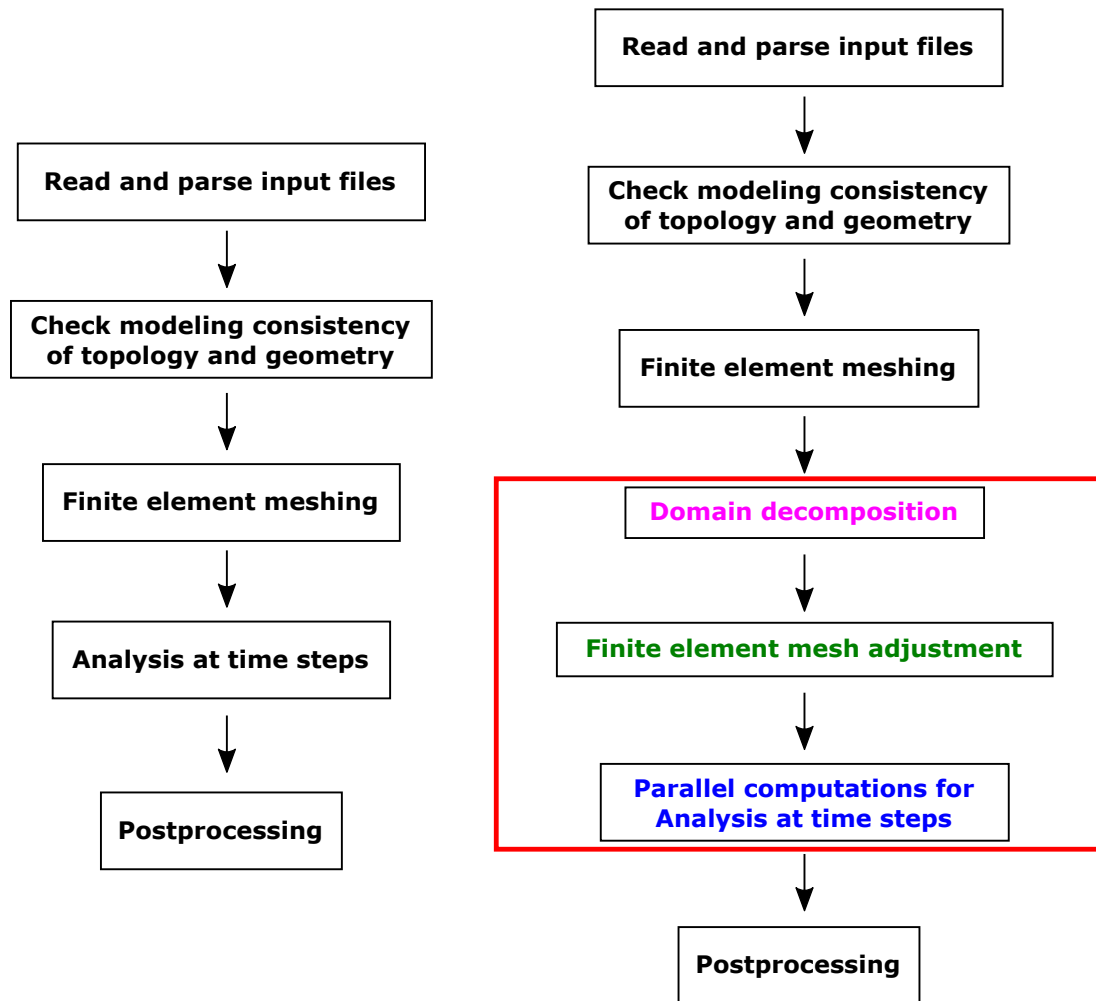


Figure 4.1: Finite element procedures (Left: Sequential; Right: Parallel)

4.2 *Domain Decomposition*

To partition a computational domain into non-overlapping subdomains, a domain decomposition step is added to the preprocessing step. The domain decomposition step is placed after the first finite element meshing for the original computational domain. Since subdomain problems are independent of each other, for implementation convenience, each subdomain problem can be managed by a *subdomain manager* which is a data structure to store finite element information of a subdomain. Similarly the interface problem can be managed by an *interface manager*. The domain decomposition process can be subdivided into the following sub-processes:

1. Generate partitioning inputs by a user.
2. Assign a subdomain number to each existing finite element based on the partitioning inputs.
3. Distribute the subdomain-number-assigned finite elements to the corresponding subdomain managers.
4. For every single node, detect whether a node is located at interface or is internal to a subdomain.
5. If a node is internal to a subdomain, no additional process is required.
6. If a node is at interface, figure out which subdomains are adjacent to the node.
7. For the node at interface, designate it as an interface node and create subdomain boundary nodes of the adjacent subdomains.
8. Create LLM elements associated with the subdomain boundary nodes around the interface node
9. Store the interface node in the interface manager

10. Store each LLM element and the associated subdomain boundary nodes, in a corresponding subdomain manager.

This series of sub-processes for domain decomposition is briefly depicted in Figure 4.2. Through the domain decomposition, for each subdomain, a subdomain manager will have a set of finite elements which belong only to the associated subdomain. The set of finite elements may include structural elements, constraint elements, LLM elements, etc.

4.2.1 Interface node detection

User-generated partitioning inputs can be used to assign subdomain numbers on every finite element. Once this is done, the interface node detection step can proceed, see Figure 4.2. To detect interface nodes, consider a reference node (each blue circle the first step) of a finite element model. If connected elements around the reference node are associated with two or more different subdomains, the reference node becomes an interface node (each red circle in the second step). Otherwise the reference node is an internal node to a subdomain. Every time the interface detection process is done for each node, an interface node will be found and the domain decomposition process is ready to create an LLM element.

4.2.2 LLM element

If a reference node turns out to be an interface node from the previous step, an LLM element for the node can be defined and created. An LLM element is defined by two primal nodes and one dual node: a subdomain boundary node, an interface node and a Lagrange multiplier node. The finite element information of the interface node is duplicated and passed to the newly generated subdomain boundary nodes. A pair of the subdomain boundary node and the Lagrange multiplier node will be passed to the corresponding subdomain manager while the interface node will be passed to the interface manager.

This element is a type of constraint elements, which enforces the equality of generalized coordinates of two adjacent nodes while they are in two different subdomains. The geometric positions and orientations of the two primal nodes must be identical at every time step at convergence because they were originally the same node before the domain decomposition.

It is interesting to note that if two independent primal nodes are somehow defined in the same subdomain and their positions and orientations must be identical, the definitions of the two different nodes are not even required because they can be implicitly expressed by a single node with the help of the *master-slave node elimination* method. For an LLM element, however, since the two primal nodes of the element are in two different subdomains and the subdomains are independent of each other, the two nodes must be explicitly and independently defined to enforce a kinematic constraint between them across the different subdomains.

4.2.3 Job distribution to processors

Through a domain decomposition process, subdomain managers and an interface manager can be ready to perform computations in parallel because they are all independent managers. For parallel computations, multiple processors must be assigned for the subdomain managers and the interface manager. For this purpose, it is helpful to divide the multiple processors into a *master* processor (or just a *master*) and *slave* processors (or just *slaves*) for the rest. The master processor can collect data from slave processors, process the collected data and distribute the processed data back to slave processors while the slave processors can do their own job and communicate only with the master processor. This separation of processors into master and slaves would be a good match with another separation of the interface manager and the subdomain managers. As expected, the interface manager can be assigned to the master processor while each subdomain manager to a slave processor. But a slave processor

can have multiple subdomain managers as seen in Figure 4.3 as a more general case. Once the distribution of managers to processors is done, the domain decomposition process for parallel computations may finish. It should be noted that, in the actual implementation, the standard master-slave framework is slightly modified so that the master processor includes a subdomain manager, say Subdomain 1, as well as the interface manager because this modified framework is not expected to significantly affect the total parallel performance and the modified framework doesn't also need to subtract one processor which is the master, from counting the total number processors used for parallel performance assessment (otherwise, counting the total number of processors for parallel performance may be confusing).

4.3 Finite Element Mesh Adjustment

Although every element of an original finite element model is already meshed before a domain decomposition process, the mesh of each subdomain must be modified because the subdomain boundary has been created after the domain decomposition process. Along the subdomain boundary, interface nodes and LLM elements have also been created. Since the LLM elements are added and localized to corresponding subdomains, each subdomain manager must reflect the changes by some mesh adjustment. For example, Lagrange multiplier nodes of LLM elements are added to subdomains and the nodes provide additional dofs to subdomain problems. While an interface node of an LLM element belong exclusively to the interface manager, a subdomain manager owning the LLM element should be able to retrieve the interface node when necessary because the LLM element always connects a subdomain boundary node, a Lagrange multiplier node and an interface node. Re-ordering all node numbers of each subdomain is also an important step to minimize the bandwidth of the subdomain system matrix. If a sparse solver is used, auxiliary arrays, such as diagonal and skyline arrays, for the sparse storage scheme also need to be re-computed and

re-generated.

4.4 Message Passing between Processors

To perform computations in parallel, one must implement an API (Application Programming Interface) for inter-processor communications. There are several parallel APIs such as POSIX threads (or just pthreads), OpenMP (Open Multi-Processing) and MPI (Message Passing Interface). For a parallel program, an API can be exclusively used or they can be implemented in combined manner. The parallel APIs have their own characteristics such as portability, performance, ease of implementation, etc. Out of the options above, MPI can provide the best portability and performance [51]. MPI can be implemented distributed memory platforms like clusters, as well as shared memory platforms like workstations or multi-core CPU PCs. Thus MPI will be the best option for large scale finite element problems at the end.

MPI commands

An inter-processor communication with MPI can be defined by the following questions:

- Which processor sends?
- Where is the data in the sending processor?
- What type of data is sent?
- How much is the data?
- Which processors receive the data?
- Where is the data stored in the receiving processor?
- How much data does the receiving processor expect?

An MPI message consists of two components: body and envelope. The body of a message can be defined by three subcomponents: buffer, data type and count. The buffer indicates a starting memory address where sending or receiving data is stored. The data type is the type of the message data and the data type must be identical for both sending and receiving processors. The count means the number of items in the data. On the other hand, the envelope of a message is composed of four subcomponents: source, destination, communicator and tag. The source simply indicates the sending processor while the destination does the receiving processor. The communicator specifies a group of processors to which both source and destination belong to. The tag is to distinguish the sending or receiving message.

If two processors need to send and receive an array of numbers, a pair of simple MPI commands for the two processors can be written as:

```
MPI_Send(array,arraySize,MPI_DOUBLE,sendRank,tag,MPI_COMM_WORLD,status)
```

```
MPI_Recv(array,arraySize,MPI_DOUBLE,recvRank,tag,MPI_COMM_WORLD,status)
```

This kind of inter-processor communication happens at several locations in the solution process of the proposed parallel algorithm. On the other hand, when a convergence norm needs to be computed at each iteration, convergence norms for all subdomains must be added up. In this case, a collective communication command can be used to add them up as the following:

```
MPI_Reduce(subNorm,norm,size,MPI_DOUBLE,MPI_SUM,rank0,MPI_COMM_WORLD)
```

Point-to-point communication commands such as `MPI_Send` and `MPI_Recv` and collective communication commands such as `MPI_Reduce` and `MPI_Bcast`, can be properly used to effectively transfer data between processors. The structure of data to

be transferred can be customized by a user just as *struct* type in C programming language. MPI also supports writing on a file in parallel. All these kinds of functions are well defined in the MPI library and the library has been continuously and actively updated. Table 4.1 shows some MPI commands.

Table 4.1: Various MPI commands

	MPI command	Remarks
Minimal MPI subset	MPI_Init	Initialize MPI
	MPI_Comm_size	Get number of processors
	MPI_Comm_rank	Get processor number
	MPI_Send	Send a message
	MPI_Recv	Receive a message
	MPI_Finalize	Finalize MPI
Collective communication	MPI_Barrier	Synchronize processors
	MPI_Bcast	Broadcast a message to all processors
	MPI_Reduce	Reduce values on all processors to a single value
Custom data type	MPI_Type_create_struct	Create a derived data type
	MPI_Type_commit	Commit the data type
Writing on a file	MPI_File_open	Open a file
	MPI_File_write	Write the file in parallel
	MPI_File_close	Close the file

4.5 *Data Structures for Parallel Computations*

The parallel finite element analysis program requires different and additional data structures compared to the sequential counterpart. As stated earlier, subdomain managers and an interface manager are defined. For inter-processor communication, memory buffers are also necessary to seamlessly pass messages/data between master and slave processors. The recommended data structures for the proposed parallel algorithm are depicted in Figure 4.4 and organized as the following:

- Parallel Info
 - Rank (Processor ID)
 - Number of subdomains assigned to this processor
- **Interface Manager**
 - : Sub-fields are exclusively for **Interface** except MPI Buffer

- Finite elements and nodes with their mesh info
 - Jacobian matrix and residual array ($\underline{\underline{\bar{A}}}_{cc}$ and \underline{b}_c)
 - Auxiliary arrays for sparse solver
 - Incremental solution array ($\Delta\underline{x}^c$)
 - Reference, initial and final configurations (\underline{x}_0^c , \underline{x}_i^c and \underline{x}_f^c)
 - MPI buffer arrays for inter-processor communication
- **Subdomain i Manager**
 - : Sub-fields are for **Subdomain i** and its partial contribution to **Interface**
 - Finite elements and nodes with their mesh info
 - Jacobian matrix and residual array ($\underline{\underline{A}}^{(i)}$, $\underline{\underline{A}}_{bc}^{(i)}$, $\underline{\underline{\bar{A}}}_{cc}^{(i)}$, $\underline{b}^{(i)}$ and $\underline{b}_c^{(i)}$)
 - Auxiliary arrays for sparse solver
 - Incremental solution array ($\Delta\underline{x}^{(i)}$)
 - Reference, initial and final configurations ($\underline{x}_0^{(i)}$, $\underline{x}_i^{(i)}$ and $\underline{x}_f^{(i)}$)
 - **MPI Buffer** for inter-processor communication between **Interface** manager and **Subdomain i** manager
 - Messages to be passed from Master to Slaves: $\underline{\underline{B}}_c^{(i)} \underline{x}_f^c$ and $\underline{\underline{B}}_c^{(i)} \Delta\underline{x}_c$
($\underline{\underline{B}}_c^{(i)}$: Mapping from the interface to a subdomain)
 - Messages to be passed from Slaves to Master: $\underline{\underline{\bar{A}}}_{cc}^{(i)}$, $\underline{b}_c^{(i)}$ and $\underline{d}^{(i)}$

4.6 *Nonlinear Dynamic Analysis in Parallel*

Many multibody dynamics problems are nonlinear. The governing equations of the problem usually form a set of nonlinear differential algebraic equations (DAEs). Thus it is necessary to linearize the governing equations. Finite element discretization and

a time integration scheme further transform the linearized governing differential equations into a set of linearized algebraic equations. This linearized algebraic equations can be solved for incremental solution by an iterative process such as Newton-Raphson method.

Consider a dynamics problem of a multibody system which is partitioned into non-overlapping subdomains by the proposed domain decomposition method. Through the domain decomposition process, the original problem is now divided into subdomain problems and an interface problem. For implementation, each subdomain problem is assigned to a subdomain manager and the interface problem to an interface manager. With the slightly modified master-slave framework for efficient MPI implementation, a master processor takes the interface manager and a subdomain manager while each slave processor takes only a subdomain manager by job distribution.

Now it is ready to start the analysis procedure in order to solve the system of linearized algebraic equations for incremental solutions with an iterative process. Inter-processor communications are necessary whenever the subdomain managers need the interface solution or the interface manager needs to add up the contributions from the subdomains. The iterative process starts with a predictor step for an initial guess. The initial guess for the interface problem needs to be passed to every subdomain, which requires inter-processor communication. The main analysis step with the five phases also include inter-processor communications at the Phase 2 and 4 as stated in Section 3.5.4. At each iteration during a time step, the final configuration is updated by adding the solution increments to the final configuration at the previous iteration. The updated final configuration of the interface must be passed to every subdomain again. Once the solution reaches convergence, the initial configuration for the next time step must be updated by the final configuration at the current time step. This series of the analysis procedure for a nonlinear dynamics problem during a time step

is described below (subscripts $[0]$, $[i]$ and $[f]$ indicate reference, initial and final configurations, respectively):

Time step start: given $\underline{x}_{[0]}^{(i)}$, $\underline{x}_{c[0]}$, $\underline{x}_{[i]}^{(i)}$, $\underline{x}_{c[i]}$

1. Predict initial guess

- (a) Slaves: predict $\underline{x}_{[f]}^{(i)}$
- (b) Master: predict $\underline{x}_{c[f]}$
- (c) MPI (Master \rightarrow Slaves): $\underline{x}_{c[f]}$

2. Iteration up to convergence

(a) Solution procedure for an incremental solution

- Slaves: $\Delta\underline{x}^{(i)}$
- Master: $\Delta\underline{x}_c$

(b) Update final configuration by the increment

- i. Slaves: $\underline{x}_{[f]}^{(i)} \leftarrow \underline{x}_{[f]}^{(i)} + \Delta\underline{x}^{(i)}$
- ii. Master: $\underline{x}_{c[f]} \leftarrow \underline{x}_{c[f]} + \Delta\underline{x}_c$
- iii. MPI (Master \rightarrow Slaves): $\underline{x}_{c[f]}$

(c) Check convergence

3. Update initial configuration for the next step by $\underline{x}_{[f]}^{(i)}$ and $\underline{x}_{c[f]}$

Time step end.

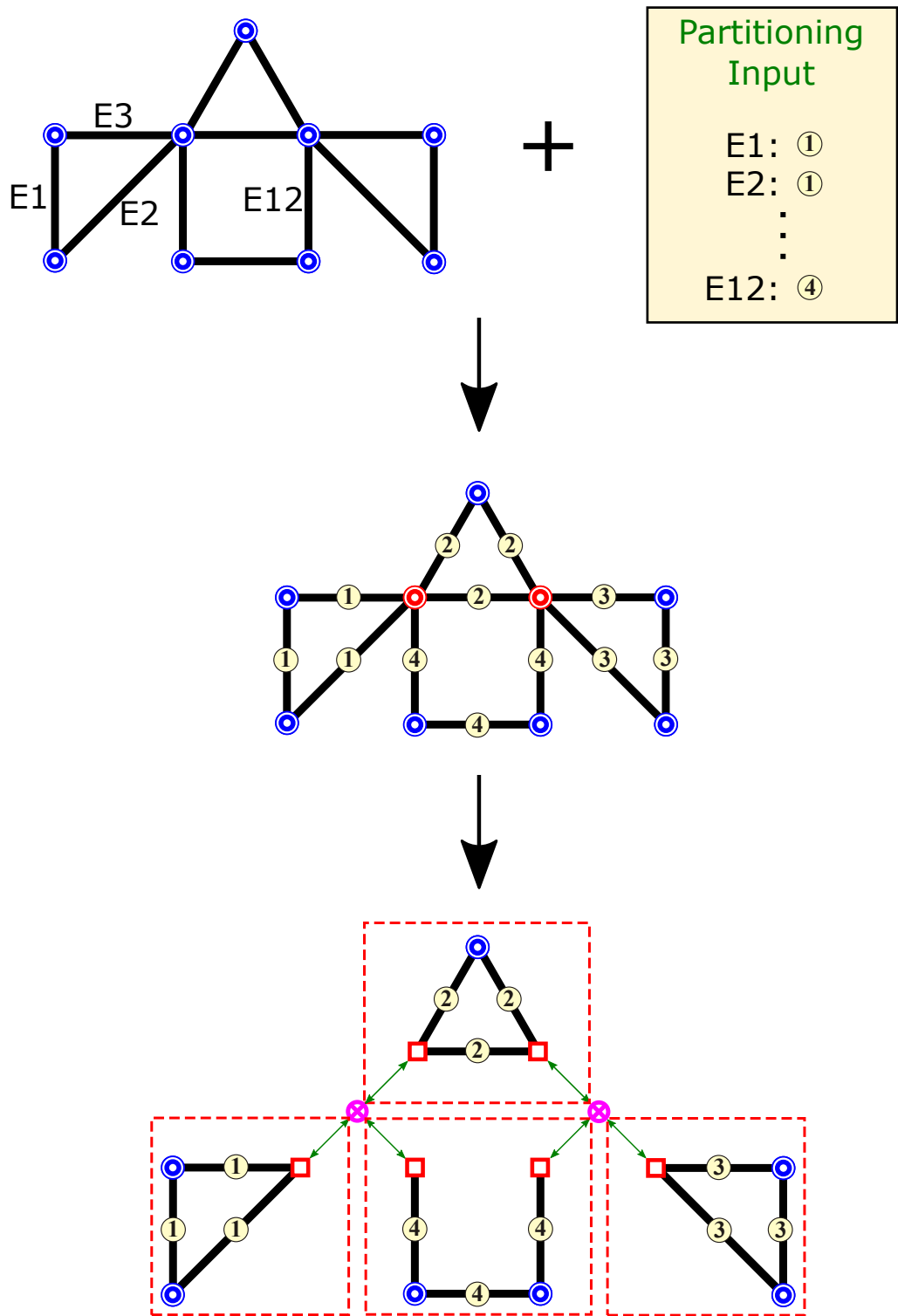


Figure 4.2: Schematic of domain decomposition process

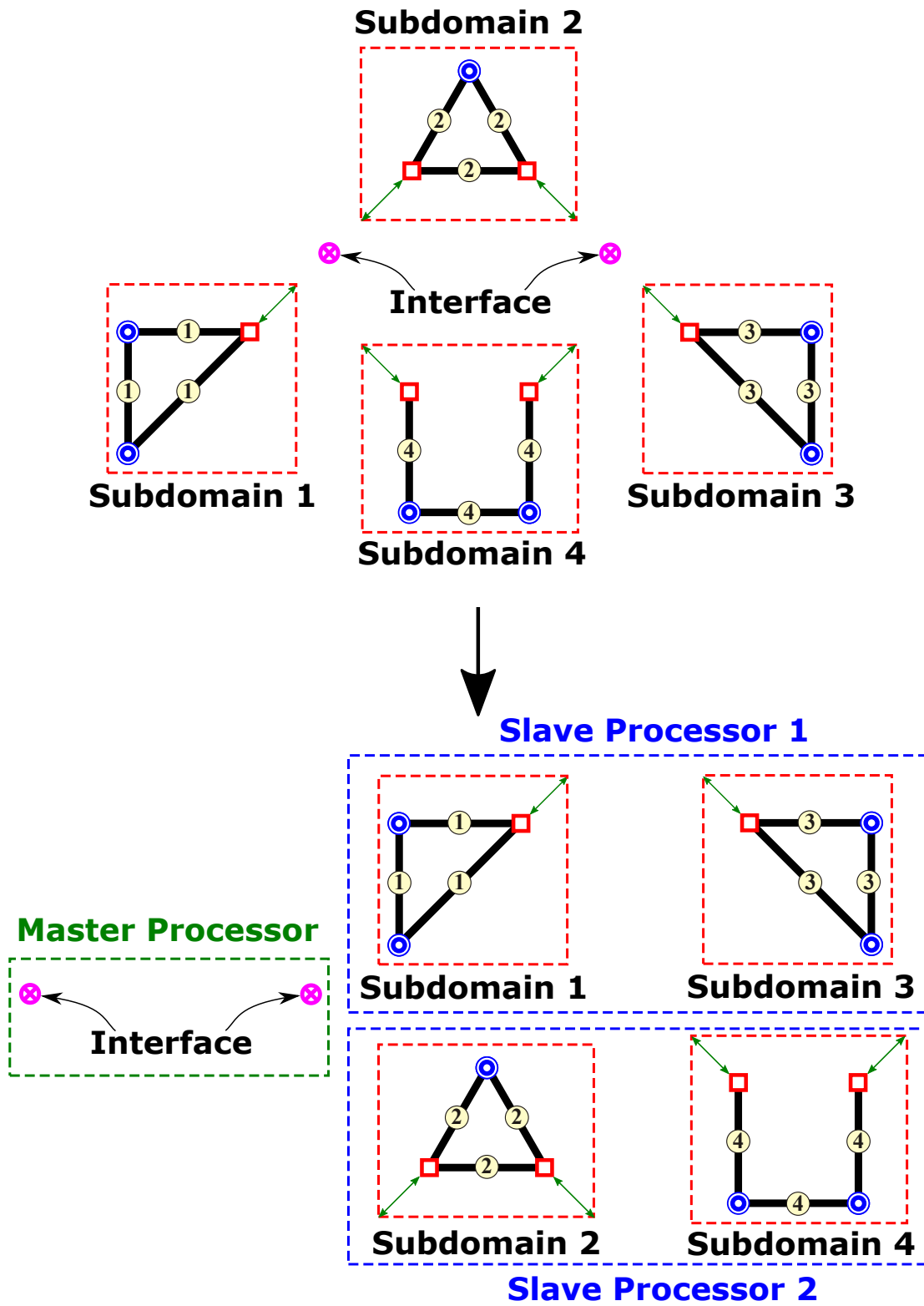


Figure 4.3: Job distribution to processors in master-slave framework

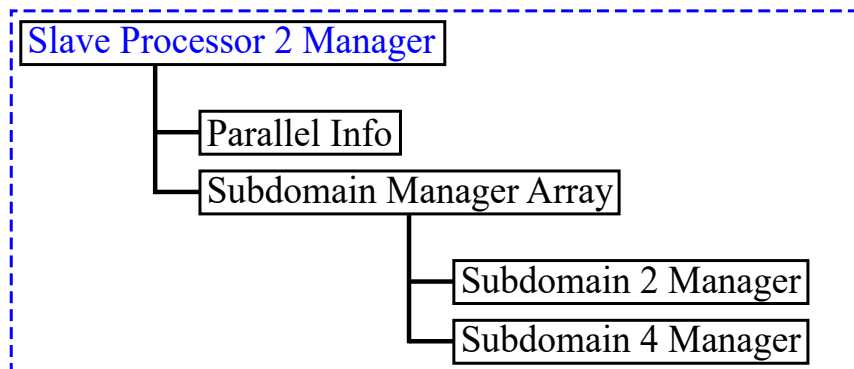
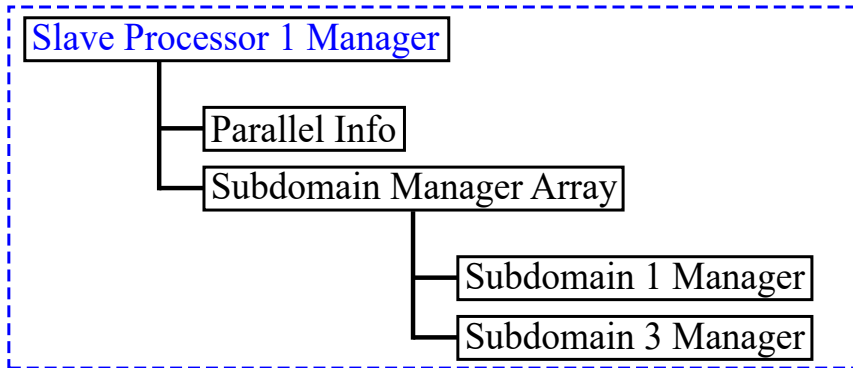
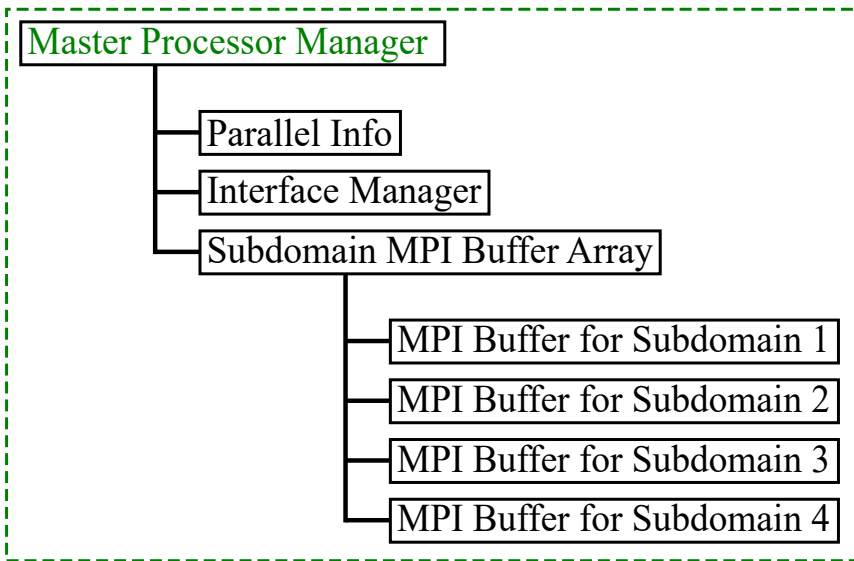


Figure 4.4: Hierarchy of parallel data structures

CHAPTER V

NUMERICAL EXPERIMENTS

In this chapter, the performance of the proposed domain decomposition method will be assessed by numerical experiments. The proposed domain decomposition method has been implemented in the flexible multibody dynamic simulation program, Dymore, and the parallel version of the program has been run and tested on parallel machines such as multi-core processor PC's or clusters.

Parallel performance of a domain decomposition method can be measured by comparing the parallel execution times on multiple processors with its sequential counterparts on a single processor. Several definitions for parallel performance measurements are described. Then, three numerical experiments will be introduced to verify the accuracy and scalability of the proposed domain decomposition method.

5.1 Testbed for Parallel Simulations

The software and hardware to run the parallel simulation tests are briefly described below:

- Software: Parallel version of Dymore
 - Multibody dynamics code
 - Direct method rather than iterative method
 - Skyline storage for sparse subdomain and interface matrices
 - Localized Lagrange multiplier technique for domain decomposition
- Hardware: “Garuda” cluster in AE at Georgia Tech
 - 608 processors on 152 nodes (4 processors per node)

- 4GB memory per node
- Intel(R) Xeon(R) CPU 5150 @ 2.66GHz

5.2 Performance Assessment

For the study of parallel performance, CPU times have been measured for the solution procedure. Two important additional metrics are used to evaluate parallel performance based on the measured CPU time: speed-up and efficiency.

Table 5.1: Parallel simulation cases

Abbrev.	Description	Remarks
SPSD	Single Processor for Single Domain	Sequential without domain decomposition
SPMD	Single Processor for Multi-Domains	Sequential with domain decomposition
MPMD	Multi-Processors for Multi-Domains	Parallel with domain decomposition

5.2.1 CPU time

The program execution time must be measured to study parallel performance. Two important definitions of the execution time are defined as: CPU time and wall clock time. The CPU time is the execution time for which the CPU has been dedicated to a process while the wall clock time measures the elapsed time from start to end including CPU time, programmed delays, wait time for resources, etc. Thus the CPU time has been measured for parallel performance to account for only the time dedicated to computations by CPU. In the sequel, CPU time and execution time are used interchangeably.

To measure the CPU time for a computer program, a programmer must implement a timer function in the program. Some factors are important to choose the timer function: precision, CPU time measurability and availability on multi-platform. Among

many timer libraries for C programming, *getrusage()* function has been used to measure the CPU time and it measures the CPU time in microseconds.

For one iteration with factorization vs. For the entire simulation

The CPU time can be measured for one iteration with factorization or for the entire simulation. The two different CPU time measures show different aspects of the performance because of the modified Newton-Raphson iteration process. In the modified Newton-Raphson iteration process, the tangent stiffness matrix is not updated at each iteration. Rather, the tangent matrix is updated at the first iteration only and remains unchanged in the subsequent iterations, leading to considerable saving of computational time.

When the CPU time is measured for one iteration only, it includes every phase of the process including the LU-factorization phase. On the other hand, when the CPU time is measured for the entire simulation, the entire simulation includes all iterations at every time step of the simulation and the LU-factorization is not performed at every iteration, in accordance with the modified Newton-Raphson iteration process.

5.2.2 Speed-up

In general, the *speed-up* is defined as the ratio of the speed of the sequential algorithm to that of its parallel counterpart,

$$\text{Speed-up} = \frac{\text{Execution time of the sequential algorithm}}{\text{Execution time of the parallel algorithm with multi-processors}}. \quad (5.1)$$

In this thesis, the parallel algorithm is enabled by the domain decomposition presented earlier. Part of the acceleration of the computation is due to the domain decomposition, part is due to the use of parallel computations. Note that the domain decomposition enables the use of parallel hardware. These two factors combine to provide better computational performance.

The effect of the domain decomposition technique on execution time can be evaluated as follows

$$\text{Effect of domain decomposition} = \frac{\text{CPU time for SPSPD}}{\text{CPU time for SPMD}}. \quad (5.2)$$

The speed-up measures the performance of the algorithm implemented on parallel hardware and is defined as

$$\text{Speed-up} = \frac{\text{CPU time for SPMD}}{\text{CPU time for MPMD}}. \quad (5.3)$$

Finally, the overall performance of the domain decomposition implemented on parallel hardware is measured by the following index

$$\text{Overall performance} = \frac{\text{CPU time for SPSPD}}{\text{CPU time for MPMD}}. \quad (5.4)$$

The overall performance is the product of the effect of the domain decomposition by the speed-up:

$$\text{Overall performance} = \text{Effect of domain decomposition} \times \text{Speed-up}. \quad (5.5)$$

Table 5.2 summarizes the definition of the various performance indices.

Table 5.2: Performance indices

Abbrev.	Definition	Remarks
SPMD	$\frac{\text{CPU time for SPSPD}}{\text{CPU time for SPMD}}$	Effect of domain decomposition only
Speed-up	$\frac{\text{CPU time for SPMD}}{\text{CPU time for MPMD}}$	Speed-up due to parallel processing
MPMD	$\frac{\text{CPU time for SPSPD}}{\text{CPU time for MPMD}}$	Overall performance

5.2.3 Efficiency

The parallel efficiency shows how efficiently the parallel hardware is used. One might expect that a parallel program runs twice faster when two processors are used. Efficiency is calculated dividing speed-up by the number of processors.

$$\text{Efficiency} = \frac{\text{Speed-up}}{\text{Number of processors}} \quad (5.6)$$

5.2.4 LU factorization phase

The LU factorization phase is the most expensive operation in the finite element procedure. Thus it is interesting to assess the parallel performance of the factorization phase only. The computational cost of the factorization is the function of the size of the stiffness matrix. When the skyline solver is used as a solution scheme, an additional metric for matrix sparsity needs to be defined to assess the effect of many vanishing entries of the stiffness matrix. Let n be the total dofs of a sparse stiffness matrix and s be the required memory size for the stiffness matrix with a sparse storage scheme. Then the *mean bandwidth*, m , of the stiffness matrix is defined as

$$m = \frac{s}{n}. \quad (5.7)$$

Because the cost of the LU factorization with a sparse matrix is approximately proportional to nm^2 , defined as the *factorization cost index*,

$$\text{Factorization cost index} = nm^2. \quad (5.8)$$

The accuracy of this estimate can be assessed by meaning the CPU time required to factorize matrix of given size and bandwidth.

As an increasing number of subdomains used, the size of the subdomain stiffness matrix becomes smaller while that of the interface stiffness matrix becomes larger. Thus, as the number of subdomains increases, the CPU time required for the factorization of the subdomain stiffness matrices is expected to decrease, whereas the CPU time required for the factorization of the interface stiffness matrix increases.

5.2.5 Inter-processor communication time

In the solution procedure, communication between subdomain managers and an interface manager is necessary to add up contributions of the subdomains to that of the interface or to broadcast the solution at the interface back to subdomains. Since the managers of subdomains and an interface are assigned to different processors, the inter-processor communication phase is an essential process during execution. The interface manager must communicate with all subdomain managers. Inter-processor communication phase is the bottleneck of a parallel algorithm because this phase must proceed sequentially, not in parallel. Thus, minimizing the inter-processor communication time is one of the challenging tasks for designing parallel algorithms.

Since MPI has been used for the inter-processor communication API, the inter-processor communication time is a function of sender/receiver overhead and communication data size. While the communication data size can be easily measured, the overhead can't be measured easily. Therefore, in this thesis, communication data size is used to assess the communication time.

During the domain decomposition process, LLM elements are generated and assigned to corresponding subdomains. The number of LLM elements of a subdomain is directly related to the size of the inter-processor communication data. Let $n_b^{(i)}$ be the dofs of all LLM elements of the subdomain i , the size of communication data can be calculated by adding up the sizes of the matrix $\underline{\underline{A}}_{cc}^{(i)}$ and the arrays $\underline{b}_c^{(i)}$, $\underline{d}^{(i)}$ and $\underline{x}_c^{(i)}$, see Table 3.1. The size of matrix $\underline{\underline{A}}_{cc}^{(i)}$ is $n_b^{(i)} \times n_b^{(i)}$ while that of the three arrays is $n_b^{(i)} \times 1$. Thus, the total data size for the inter-processor communication (MPI) per iteration is expressed as

$$\text{Total MPI data size per iteration} = \sum_{i=1}^{N_s} n_b^{(i)} \cdot n_b^{(i)} + 3n_b^{(i)}. \quad (5.9)$$

It should be noted again that the actual inter-processor communication time will include the overhead of senders and receivers in addition to the data transmission

time.

5.2.6 Contributions of sub-phases

As the number of subdomains increases, the problems handled by the various subdomains and an interface also change, leading the changes in the execution times for the various phases of the algorithm. For cases with different numbers of subdomains, relative contributions of sub-phases can be evaluated by normalizing the execution times for sub-phases. The sub-phases which are most significantly contributing to the execution time will be scrutinized. Because the normalized execution times include significant sub-phases only, the total sum of the normalized execution times may be less than 100%. For instance, sub-phases less than 2% are not included for visualization in graphs. When multi-processors are used for parallel computations, the inter-processor communication becomes relevant. The inter-processor communication is denoted by “MPI Comm” and “MPI Wait” to indicate inter-processor communication time and synchronization time between processors, respectively. The significant sub-phases are listed in Table 5.3.

Table 5.3: Significant sub-phases for execution time in solution procedure

5 phases		Sub-phases	
Phase	Description	Sub-phase	Abbrev.
Phase 1	Each subdomain: Assembly and Factorization (Parallel)	Computation and assembly of element matrices and arrays	P1 Sbd Assy
		LU factorization of subdomain matrix	P1 Sbd LU
		Computation of partial sbd-int coupling matrices	P1 Sbd Abc
		Computation of partial interface matrices	P1 Sbd Acc
Phase 2	Interface: Assembly and Factorization (Sequential)	Data transfer (Subdomains \rightarrow Interface)	MPI Comm
		Adding up partial contributions from subdomains	P2 Int Acc
		LU factorization of interface matrix	P2 Int LU
Phase 3	Each subdomain: Forward-substitution (Parallel)	Forward-substitution for intermediate subdomain equations	P3 Sbd Subs
		Computation for $\underline{d}^{(i)}$	P3 Sbd VecD
Phase 4	Interface: Forward- and back- substitutions (Sequential)	Data transfer (Subdomains \rightarrow Interface)	MPI Comm
		Forward-back-substitutions for interface equations	P4 Int Subs
		Data transfer (Interface \rightarrow Subdomains)	MPI Comm
Phase 5	Each subdomain: Back-substitution (Parallel)	Back-substitution for subdomain equations	P5 Sbd Subs
MPI	Synchronization of all processors	All processors need to synchronize	MPI Wait

5.3 Cantilever Beam

The first example is a simple cantilever beam. This simple example is used to verify the accuracy of the proposed domain decomposition method when implemented on parallel hardware. The displacement predictions and the convergence behavior are compared for its sequential and parallel implementations.

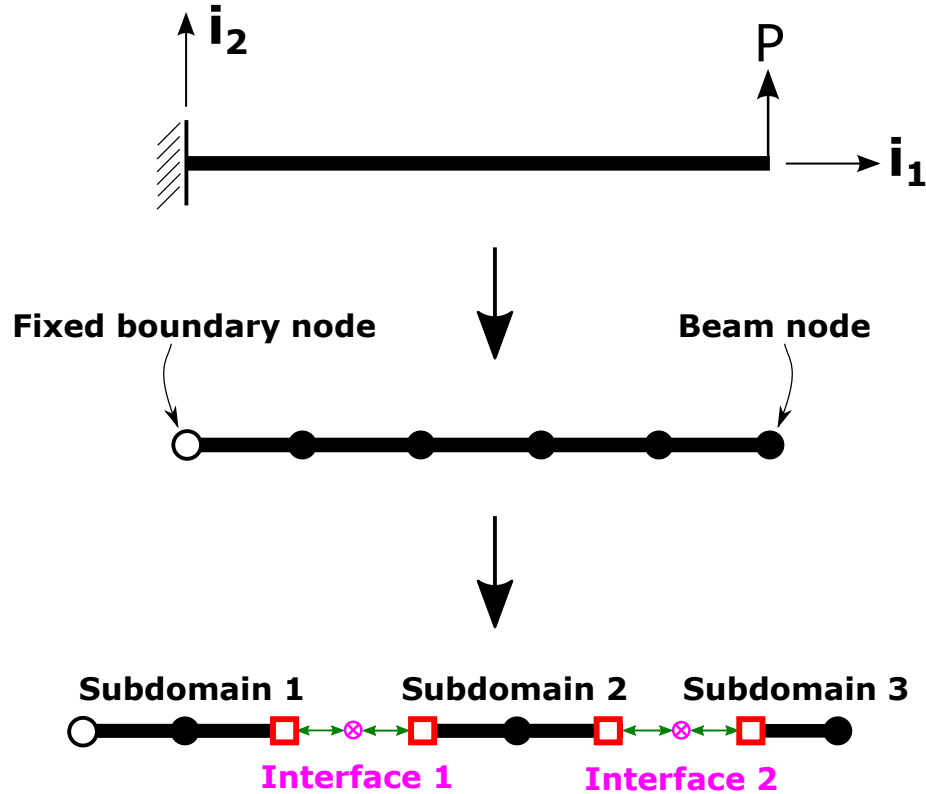


Figure 5.1: Cantilever beam: Partitioned into three subdomains

The beam length is 5 m. The cross-sectional properties are uniform over the span and are defined as following: bending stiffness $H_{33} = 1.0 \times 10^6$ N.m² and shearing stiffness $K_{22} = 1.0 \times 10^7$ N. The beam is subjected to a vertical force $P = 10$ N in the direction of i_2 at the beam tip. The beam is meshed into five elements of equal size and for simplicity, each beam element features two nodes only, as illustrated in Figure 5.1. As the node at the root is fixed by the boundary condition, the beam presents five active structural nodes and each node has six dofs for three displacements and three rotations. Thus, the total number of dofs for the beam is 30. For a static

simulation, the convergence tolerance for iteration termination is set to 1.0×10^{-9} .

Table 5.4: Cantilever beam: Problem sizes for domain decomposition

Number of subdomains	Subdomain dofs			Interface dofs	Total dofs
	1	2	3		
1	30	N/A	N/A	N/A	30
3	18	30	18	12	78

The cantilever beam is then partitioned into three subdomains by the proposed domain decomposition method. Through the domain decomposition process, four subdomain boundary nodes, four localized Lagrange multiplier nodes, and two interface nodes are added to the domains as illustrated by Figure 5.1. Subdomains 1 and 2 have two elements for each with the same length of 2 m. Subdomain 3 has one element with 1 m long. The total problem size changes when the domain decomposition is performed, as shown in Table 5.4. While the unpartitioned problem has 30 dofs, the partitioned problem with three subdomains has 78 dofs, which means that the partitioned problem size has increased by 160 % from the unpartitioned counterpart, because of the additional dofs of subdomain boundary nodes, localized Lagrange multiplier nodes and interface nodes. Therefore, the domain decomposition is not efficient for such a simple problem.

Table 5.5: Cantilever beam: Tip displacement

Computation	Displacement prediction
Sequential	8.699999998356e-05
Parallel	8.699999998356e-05
Difference	0.000000000000e+00

For the verification of the proposed domain decomposition method, the displacement prediction is evaluated at the tip in the direction of ι_2 . Tables 5.5 and 5.6 show the displacement predictions and the achieved convergence norms over iterations, respectively, for both the sequential and parallel computations. As seen in the two

tables, both sequential and parallel computations with the proposed domain decomposition method have generated the identical predictions and convergence behaviors within machine accuracy.

Table 5.6: Cantilever beam: Convergence norms over 3 iterations

Iteration number		1	2	3
Convergence norm	Sequential	1.643167672515e-02	2.439493338641e-06	9.432866543467e-10
	Parallel	1.643167672515e-02	2.439493338641e-06	9.432866555410e-10
Norm difference		0.000000000000e+00	0.000000000000e+00	1.194299912016e-18

5.4 Grid of Beams

The second example is a *grid of beams*. This structure is built up only by multiple beams and the beams are interwoven as seen in Figure 5.2. Various domain decompositions of this structure can generate interfaces at many places so that the effect of interface size can be investigated.

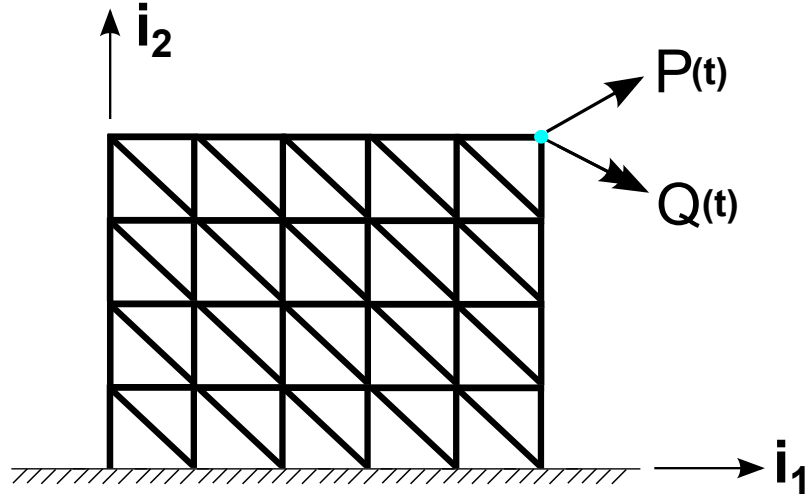


Figure 5.2: Grid of beams

Table 5.7: Grid of beams: Structural properties of a beam segment

Property	Unit	Value
Axial stiffness (S)	N	4.0×10^7
Bending stiffness ($EI_{22}, EI_{33}, EI_{23}$)	N.m ²	$2.4 \times 10^6, 2.4 \times 10^6, 0.0$
Torsional stiffness (J)	N.m ²	2.8×10^5
Shearing stiffness (K_{22}, K_{33}, K_{23})	N	$2.0 \times 10^6, 2.0 \times 10^6, 0.0$
Mass per unit span (m)	kg/m	3.2
Moment of inertia (m_{11}, m_{22}, m_{33})	kg.m	$2.4 \times 10^{-2}, 1.2 \times 10^{-2}, 1.2 \times 10^{-2}$

5.4.1 Model description

Each cell of the grid is a square of size 1.2×1.2 m with a diagonal connection. This grid structure consists of five cells in width and four cells in height. Thus this model is designed to have the total of 64 beam segments to match 2^N partitioning,

see Figure 5.3. The finite element mesh for each beam segment is generated by three cubic (four-node) beam elements.

5.4.2 Domain decomposition

The proposed domain decomposition process has been performed to partition the structure into 2^N subdomains where $N = 0, 1, \dots, 5$. The problem sizes of the six cases for different domain decompositions are summarized in Table 5.8 and depicted in Figure 5.4. The problem size for each case can be expressed by dofs of the subdomains and the interface. Because there are multiple subdomains, the subdomain problem dofs for each case in Table 5.8 is an averaged value for all subdomains, unlike the interface problem. The average number of dofs for each case are rounded for easier reading in the table. The total number of dofs is the sum of the number of dofs of the interface problem and all subdomain problems.

For the domain decompositions of the grid of beams, as the number of subdomains increases by the factor of two, the average number of dofs of a subdomain problem decreases at a constant rate while the interface problem size increases at a slower rate except for the increase of the number of subdomains from two to four, as seen in Figure 5.4. The slower increase rate of the interface problem size is due to the feature of the independent interface node of the proposed domain decomposition method.

Table 5.8: Grid of beams: Problem sizes for 6 domain decomposition cases

Number of subdomains	Subdomain dofs (Avg.)	Interface dofs	Total dofs
1	3216	N/A	3216
2	1662	36	3360
4	885	108	3648
8	454	114	3744
16	237	132	3924
32	128	138	4248

As the domain decomposition becomes finer, the total number of dofs increases slightly because of the increase in the numbers of independent interface nodes and

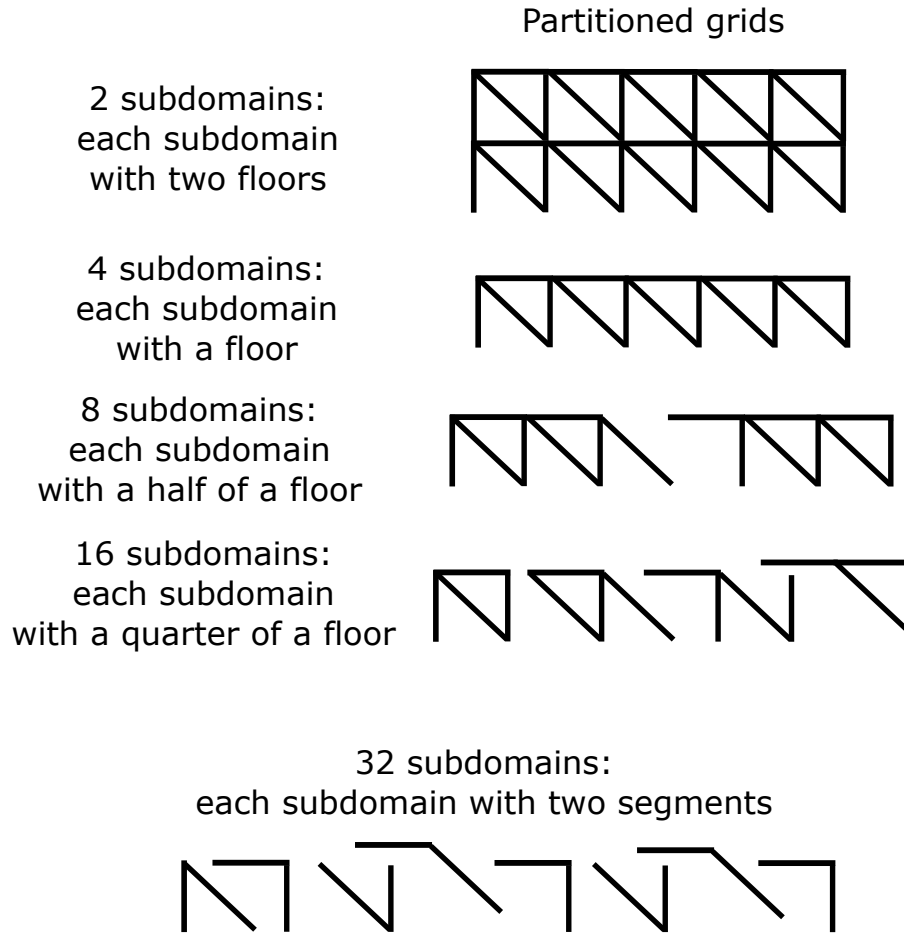


Figure 5.3: Grid of beams: Partitioning of grids

localized Lagrange multiplier nodes which are added to the subdomain problems. The number of interface dofs exceeds the number of dofs per subdomain for the 32-subdomain case. Note that, for the multi-processors for multi-domain (MPMD) simulation cases, one processor is assigned to each subdomain.

5.4.3 Predictions of dynamic response

The dynamic simulations of the grid of beams were performed for 400 time steps with a constant step size of $\Delta t = 5.0 \times 10^{-3}$, which means 2.0 sec simulation. Harmonic loading of force $\underline{P}(t)$ and moment $\underline{Q}(t)$ are applied at the upper right corner of the

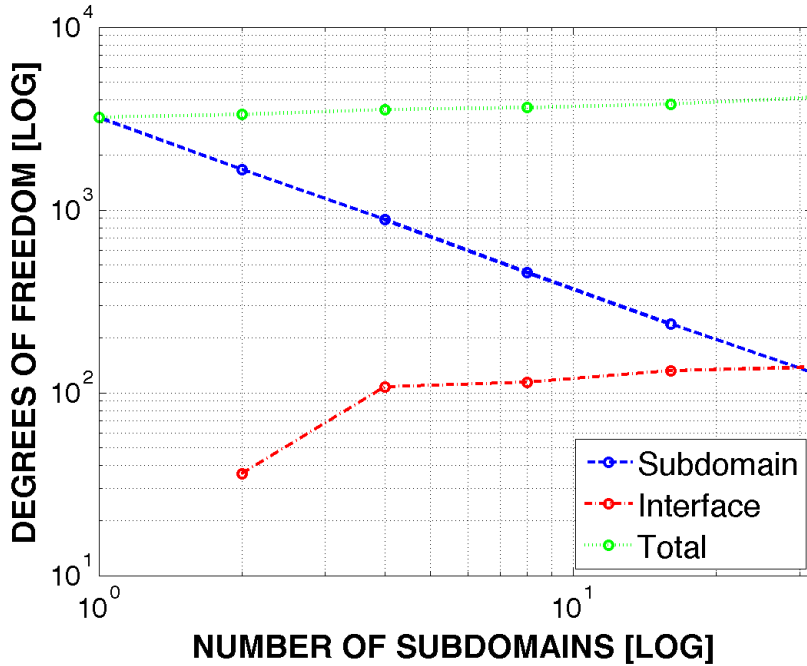


Figure 5.4: Grid of beams: Problem sizes for 6 domain decomposition cases

structure. The time history of loading is depicted in Figure 5.5 and expressed as,

$$\underline{P}(t) \text{ or } \underline{Q}(t) = \begin{cases} 0.0 & \text{for } t < 0 \\ 3(1 - \cos 2\pi t) \times 10^3 & \text{for } 0 \leq t \leq 2.0. \end{cases} \quad (5.10)$$

The structural properties of a beam segment are summarized in Table 5.7. The convergence tolerance for the iteration termination has been set to 10^{-6} .

The predictions of dynamic responses such as displacements, velocities and accelerations are investigated at the point of application of the loading in order to validate the proposed parallel approach and implementation.

The single processor for the single domain case (SPSD) and the multi-processors for the 32-subdomain case (MPMD) are compared as shown in Figures 5.6 and 5.7. The time histories of translational or rotational response predictions from both sequential and parallel computations are on the left of the figure sets. On the other hand, the response differences between sequential and parallel computations are on the right of the figure sets. As seen from the figure sets, both sequential and parallel

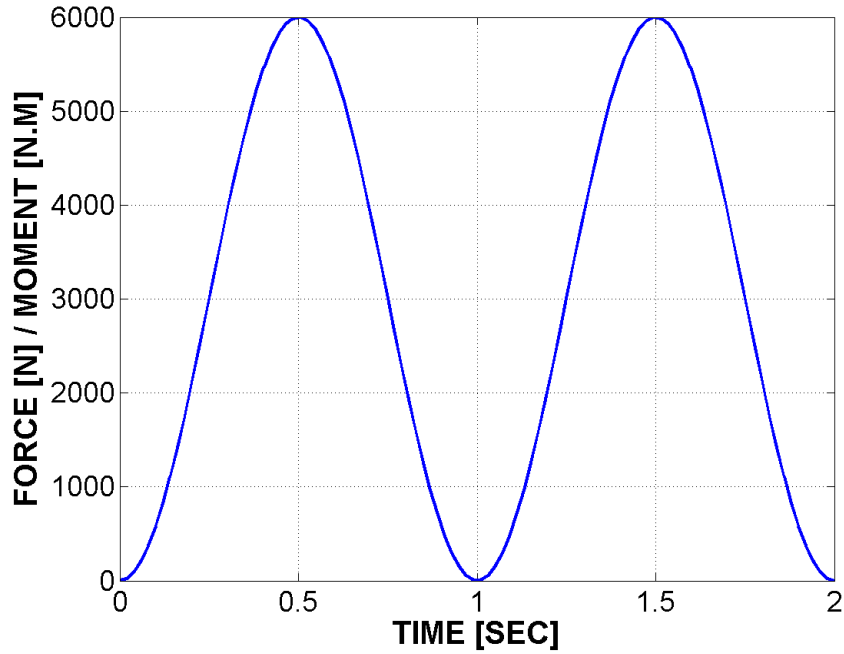


Figure 5.5: Grid of beams: Time history of loading

solutions are identical within machine accuracy through all the dynamic simulations.

5.4.4 Performance for solution procedure

With the dynamic simulations of the grid of beams, the parallel performance is presented in three metrics: CPU time, speedup and efficiency. The base metric is the CPU time only for the solution procedure which is explained in Section 3.5. The CPU time, speed-up and efficiency are listed in Tables 5.9 and 5.10 for the two different periods: for one iteration with factorization and for the entire simulation. For one iteration with factorization, the domain decomposition with 32 subdomains has achieved the shortest CPU times for both SPMD and MPMD cases with the largest speed-up of 7.60. On the other hand, for the entire simulation, the SPMD case has obtained the shortest CPU time with 32 subdomains while the MPMD case has obtained that with 16 subdomains, and the speed-up starts to drop for more than 16 subdomains.

The measured CPU times for various cases are plotted in Figure 5.8. For both one

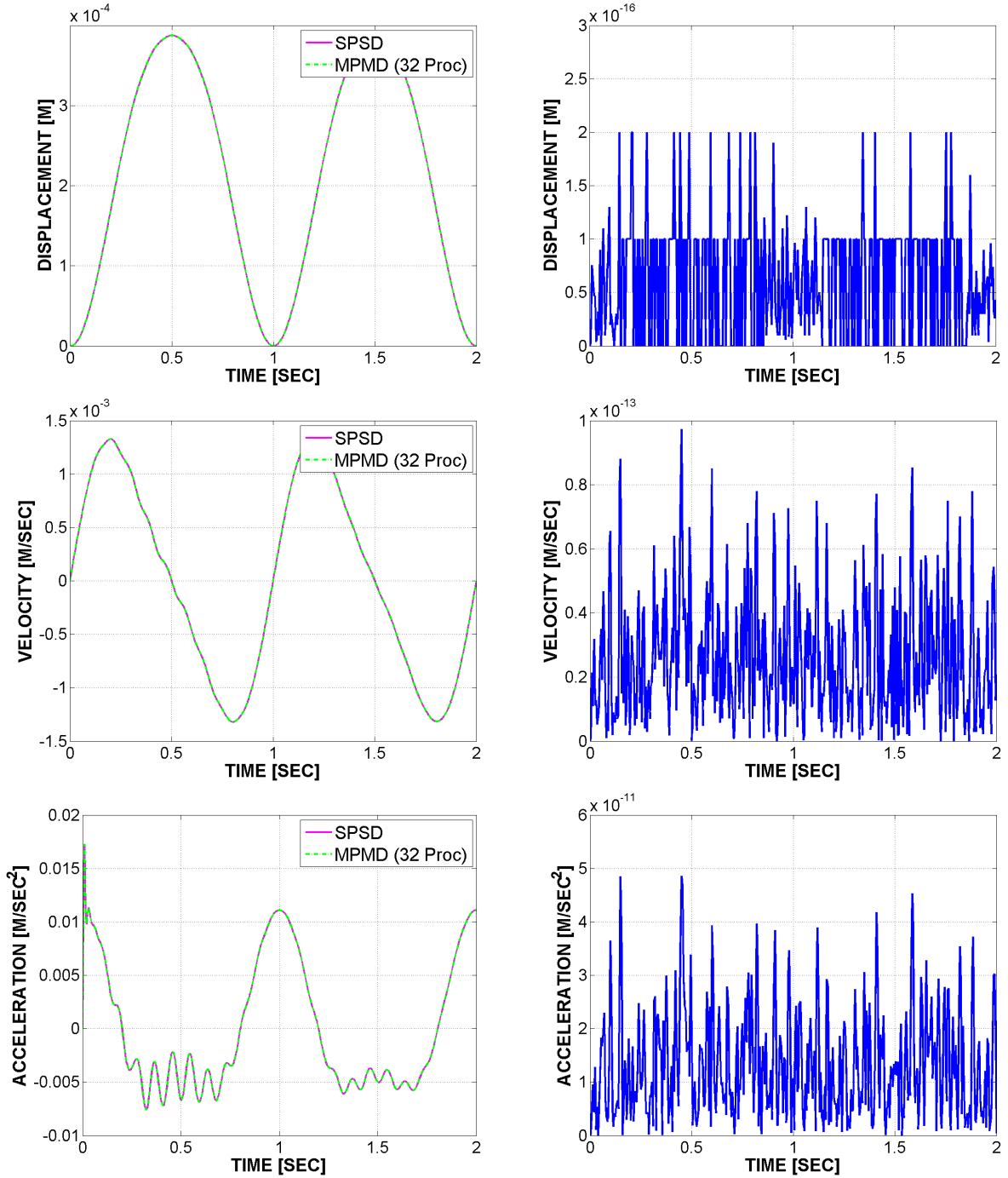


Figure 5.6: Grid of beams: Predictions of translational responses (Left) and their differences (Right) between two cases: Single domain vs. 32 subdomains

iteration and the entire simulation cases, the CPU time for the SPMD case (solid line in blue) monotonically decreases. The CPU time for the MPMD case (dash-dot line in green) is, however, going back up after hitting the shortest time with 16 subdomains.

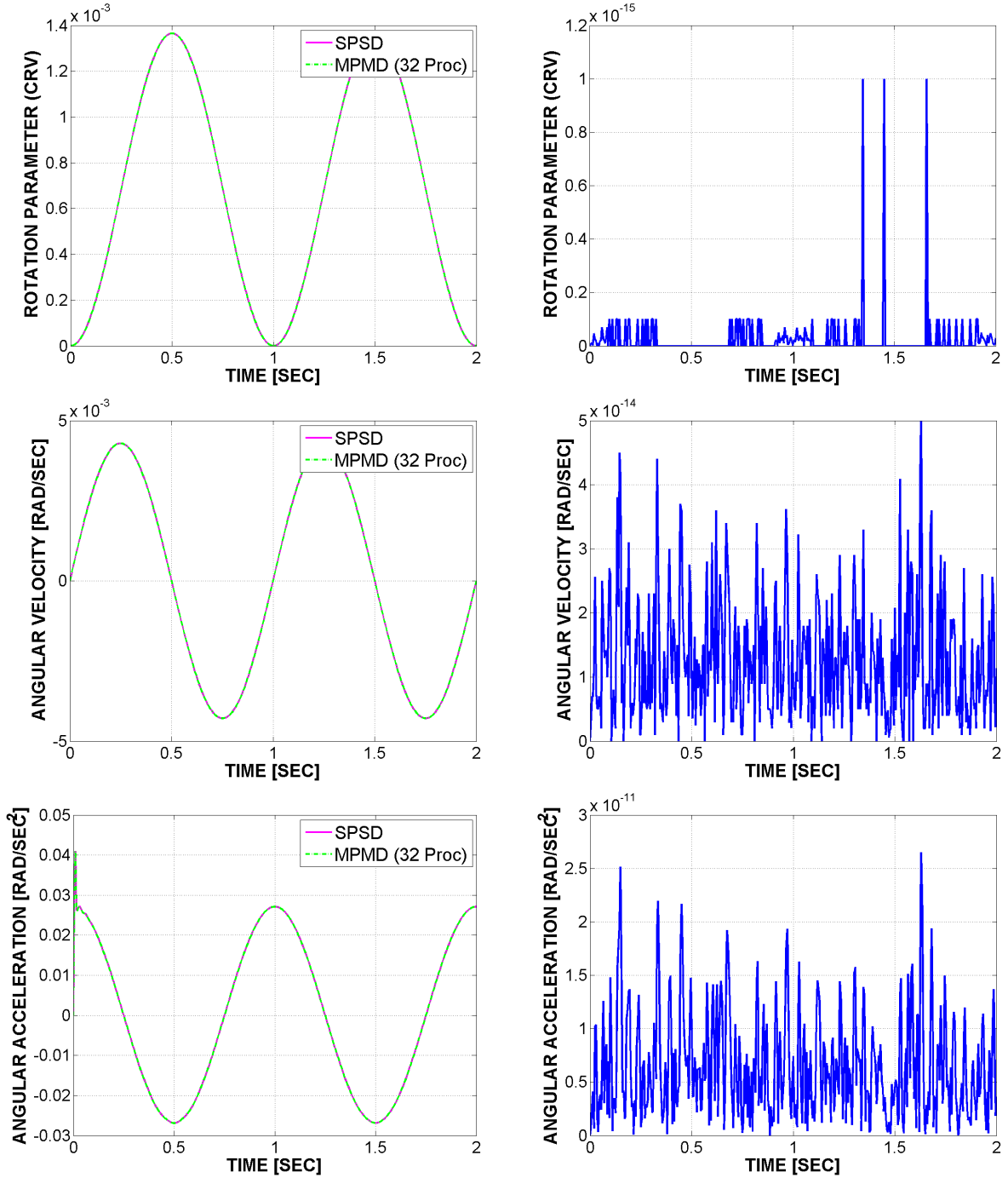


Figure 5.7: Grid of beams: Predictions of rotational responses (Left) and their differences (Right) between two cases: Single domain vs. 32 subdomains

When the plots of the speed-up or the ratio of CPU time to the SPSD case are considered, see Figure 5.9, the performance aspect is more remarkable. The speed-up (dashed line in red) clearly has a performance limit which is for 16 subdomains or

Table 5.9: Grid of beams (for one iteration with factorization): CPU time, speed-up and efficiency for solution procedure

Number of subdomains	CPU time [sec]		Speed-up	Efficiency
	SPMD	MPMD		
1	0.0727	0.0727	1.00	1.00
2	0.0704	0.0349	2.02	1.01
4	0.0674	0.0220	3.06	0.76
8	0.0485	0.0082	5.95	0.74
16	0.0378	0.0039	9.60	0.60
32	0.0265	0.0035	7.60	0.24

Table 5.10: Grid of beams (for the entire simulation): CPU time, speed-up and efficiency for solution procedure

Number of subdomains	CPU time [sec]		Speed-up	Efficiency
	SPMD	MPMD		
1	52.05	52.10	1.00	1.00
2	44.57	23.12	1.93	0.96
4	40.24	12.89	3.12	0.78
8	29.50	5.13	5.75	0.72
16	23.59	3.25	7.27	0.45
32	18.20	3.76	4.85	0.15

processors for this example model. For more than 16 processors, it is evident that the increase rate of both the speed-up and the overall performance (MPMD, dash-dot line in green) starts to decrease. It is also confirmed here that as the number of subdomains increases, the effect of domain decomposition (SPMD, solid line in blue) steadily increases the performance ratio to the SPSD case. Thus, it is clear from the plots that the overall performance (MPMD) with the domain decomposition and the multi-processors is dictated by the speed-up or parallel processing. The ratios of the CPU time to the SPSD case, speed-up and efficiency are also listed in Tables 5.11 and 5.12.

The efficiency for multi-processor usage shows slightly different aspects for the parallel performance, as shown in Figure 5.10. Since this metric applies only to multi-processor cases, the SPMD case cannot be shown here. A notable point is at

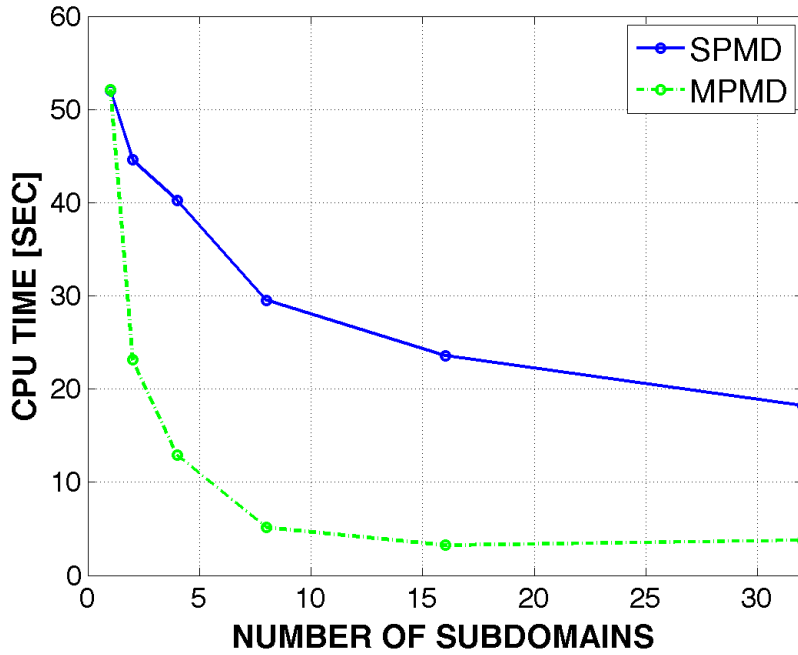
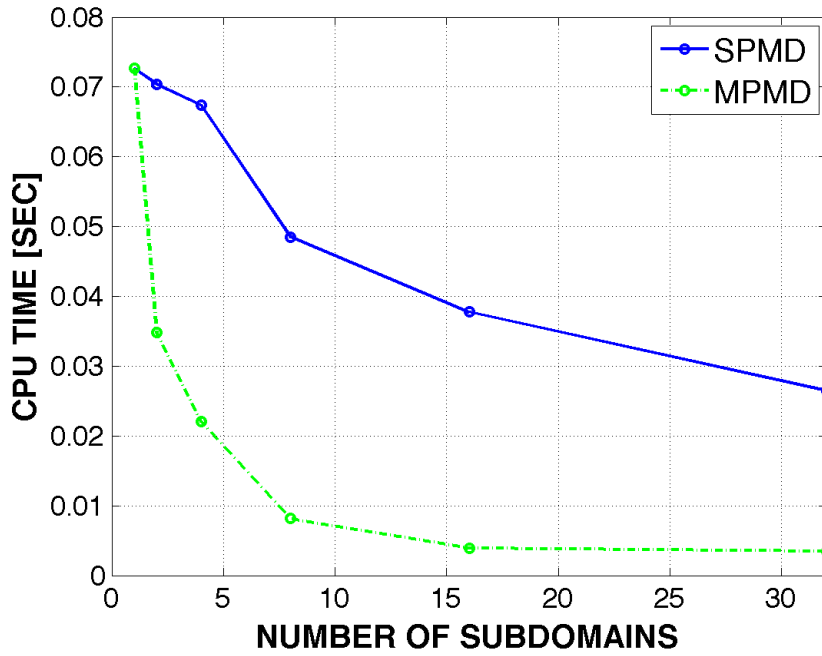


Figure 5.8: Grid of beams: CPU time for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

the four subdomain case. Unlike the neighboring cases such as two or eight subdomain cases, the efficiency plots show the V-shaped drops at the four subdomain case for

Table 5.11: Grid of beams (for one iteration with factorization): Ratio of CPU time to SPSPD case, speed-up and efficiency

Number of subdomains	Ratio to SPSPD		Speed-up	Efficiency
	SPMD	MPMD		
1	1.00	1.00	1.00	1.00
2	1.03	2.09	2.02	1.01
4	1.08	3.30	3.06	0.76
8	1.50	8.91	5.95	0.74
16	1.92	18.47	9.60	0.60
32	2.75	20.85	7.60	0.24

Table 5.12: Grid of beams (for the entire simulation): Ratio of CPU time to SPSPD case, speed-up and efficiency

Number of subdomains	Ratio to SPSPD		Speed-up	Efficiency
	SPMD	MPMD		
1	1.00	1.00	1.00	1.00
2	1.17	2.25	1.93	0.96
4	1.29	4.04	3.12	0.78
8	1.76	10.15	5.75	0.72
16	2.21	16.03	7.27	0.45
32	2.86	13.87	4.85	0.15

both one iteration and the entire simulation cases. This is because while each V-shaped drop is a normal degradation, the efficiency-going-back-up for more-than-four-subdomain cases is due to the slow increase of the number of interface nodes, which is beneficial. The efficiency dramatically reduces and becomes poor as the number of subdomains increases. This is simply because the size of the problem is not large. It is expected that the efficiency degradation will be more delayed as the size of the problem increases more.

Speaking of difference between one iteration case and the entire simulation case, it should be first understood that one full iteration includes every sub-phases of the solution procedure while every iteration of the entire simulation doesn't include every sub-phase because of the modified Newton-Raphson process, which is explained in Section 5.2.1. The increase of the speed-up or the performance ratio to the SPSPD

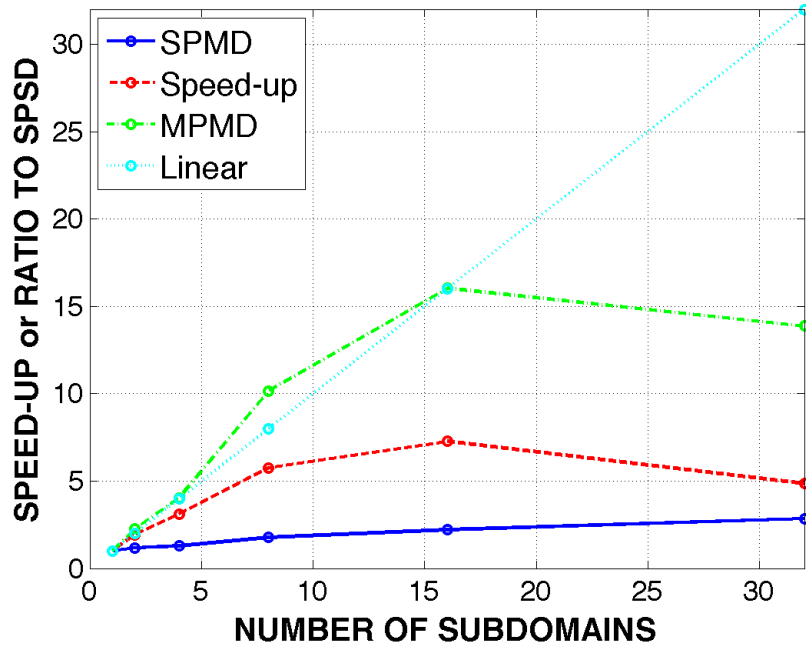
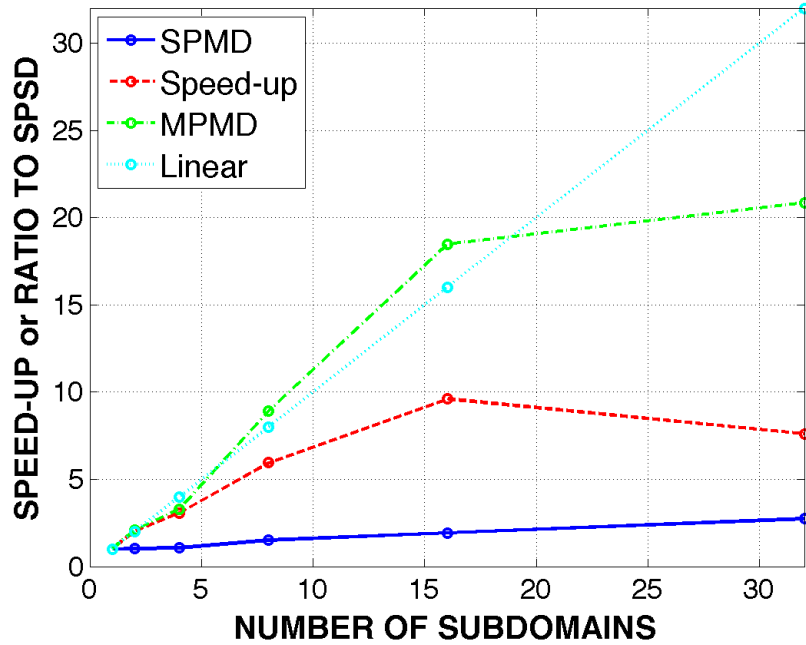


Figure 5.9: Grid of beams: Speed-up or ratio of CPU time to SPSD case for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

case for the entire simulation case is more rapid than that for one iteration case, but, at the same time, the entire simulation case also reaches the *saturation* of the

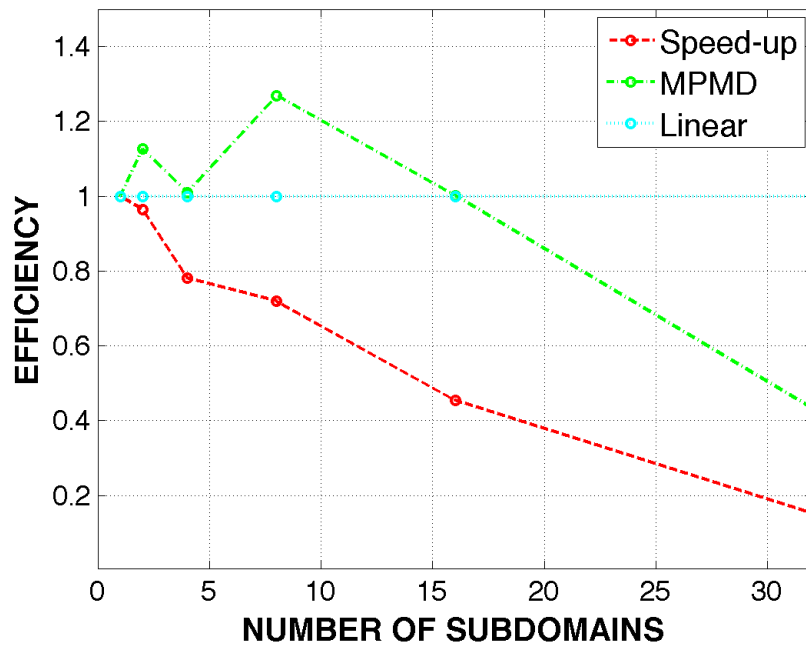
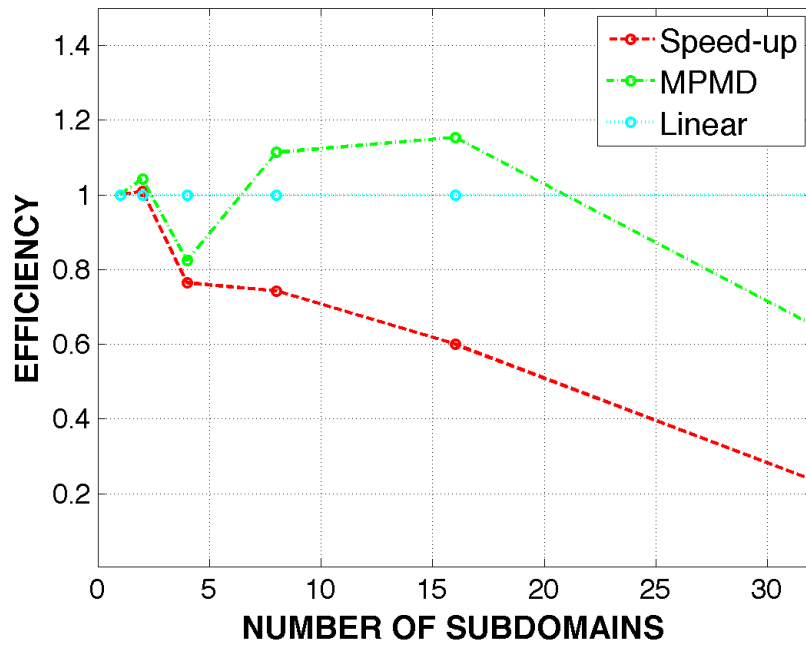


Figure 5.10: Grid of beams: Efficiency for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

performance increase faster than the one iteration case with factorization.

5.4.5 Performance for LU factorization phase

In the finite element solution procedure for typical structural problems, the LU factorization phase takes the largest portion of the execution times. Thus it will be helpful to estimate and measure the computational cost and the CPU time, respectively, of the factorization phase with various domain decomposition cases.

Table 5.13: Grid of beams: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for a subdomain problem

Number of subdomains	Degrees of freedom		Required size of memory [kB]		Mean bandwidth		LU factorization cost index	
	Max	Avg	Max	Avg	Max	Avg	Max	Avg
1	3216	←	2202	←	87	←	24341904	←
2	1680	1662	848	824	64	63	6881280	6600408
4	912	885	311	293	43	42	1686288	1548870
8	468	454	168	151	46	42	990288	811478
16	252	237	78	73	41	39	403440	357454
32	132	128	32	30	30	29	118800	111512

Table 5.14: Grid of beams: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for an interface problem

Number of subdomains	Degrees of freedom	Required size of memory [kB]	Mean bandwidth	LU factorization cost index
2	36	10	36	46656
4	108	71	84	762048
8	114	62	69	542754
16	132	65	63	523908
32	138	67	61	513498

The estimation of the factorization cost is organized in Tables 5.13 and 5.14 for the subdomain and interface problems, respectively. One can see that the load balancing for this problem is not bad when the maximum and averaged dofs are compared for each domain decomposition case. As explained in Section 5.2.4, the factorization cost is proportional to the dofs while it is so to the square of the mean bandwidth of the tangent stiffness matrix. This means that the mean bandwidth might be a

more dominant factor when the orders of magnitudes of both the dofs and the mean bandwidth are similar.

As the number of subdomains increases, one might expect that the factorization cost index of the subdomain matrix tends to decrease while the counterpart of the interface matrix tends to increase. But the estimation and the actual experiment show that the cost varying aspect for the interface matrix is not what it is expected to be. This unexpected behavior can be explained by the mean bandwidths in Table 5.13. It is understood that the mean bandwidth is the dominant factor for the factorization cost in this example model. The mean bandwidth jump from the two-subdomain case to the four-subdomain case is normal, however, for eight or more subdomain cases the mean bandwidth does not increase but even slightly decreases because the additional interface nodes are not necessary any more for the cases with more than four subdomains. This behavior is mainly due to the geometry of this example model and can be also affected by different partitionings.

Table 5.15: Grid of beams (MPMD, for the entire simulation): LU factorization CPU time and its percentage of the total simulation time

Number of subdomains	Total	Subdomain		Interface	
	simulation time [sec]	Factorization time [sec]	Percentage [%]	Factorization time [sec]	Percentage [%]
1	52.099	17.583	33.75	N/A	N/A
2	23.124	5.191	22.45	0.057	0.25
4	12.890	1.436	11.14	0.595	4.62
8	5.133	0.824	16.05	0.429	8.36
16	3.250	0.360	11.08	0.403	12.40
32	3.757	0.106	2.82	0.405	10.78

The factorization times of the actual numerical experiments are organized in Table 5.15 only for the entire simulation case. The close correlation of the factorization cost between the estimation and the experiment can be seen in Figure 5.11.

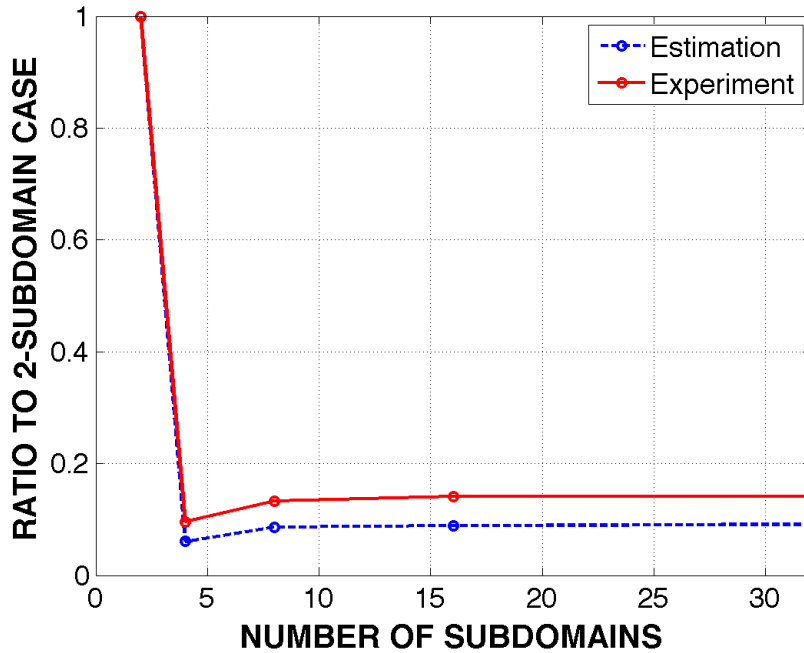
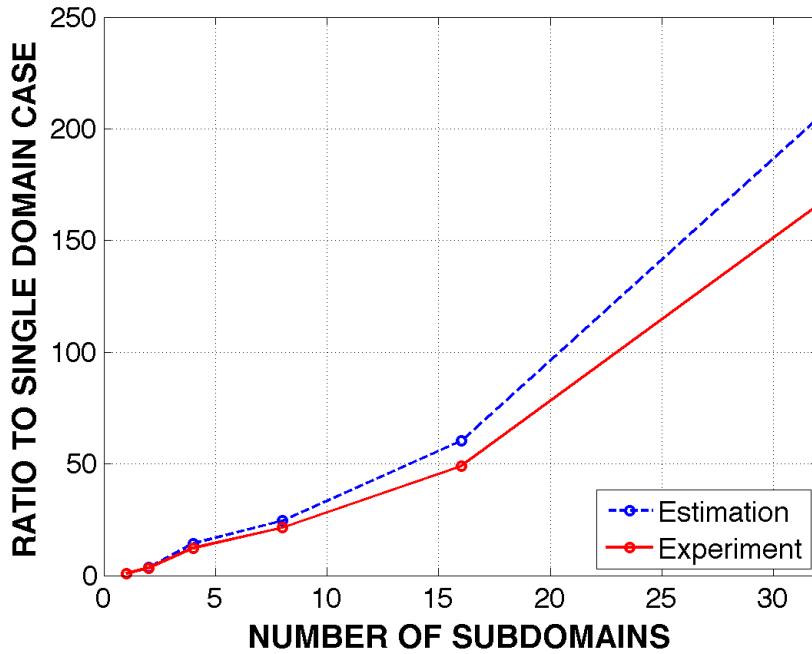


Figure 5.11: Grid of beams (MPMD): LU factorization performance (Top: Ratio of performance to single domain case for a subdomain problem; Bottom: Ratio of performance to 2-subdomain case for an interface problem)

5.4.6 Inter-processor communication time

Inter-processor communication time is another important portion of the parallel solution procedure. The size of the transferred data through MPI can be calculated by the

number of LLM elements which are generated in the domain decomposition process. The total size of the transferred data and the actual inter-processor communication time through MPI are tabulated, see Table 5.16.

Table 5.16: Grid of beams (MPMD, for the entire simulation): Dofs for all LLM elements, total MPI data size and inter-processor communication time

Number of subdomains	Dofs for all LLM elements	Total MPI data size [kB]	Total simulation time [sec]	Total MPI communication time [sec]	Time percentage [%]
2	72	22	23.124	0.022	0.10
4	216	106	12.890	0.101	0.78
8	264	80	5.133	0.180	3.51
16	354	73	3.250	0.654	20.12
32	516	80	3.757	2.206	58.72

As mentioned in Section 5.2.5, the overhead of the sender and receiver of the data, is not easily measured and, in turn, is not counted in the table. The correlation between the total MPI data size and the MPI communication time is not close. The abnormal jumps in the MPI communication time, compared to the transferred data size, might also be due to the fact that the used cluster for the parallel simulations has four processors per node so that the much slow network between nodes significantly impacts the MPI communication time when more than four processors are used for the cases with more than four subdomains.

Figure 5.12 shows the increase of the inter-processor communication time relative to the CPU time as the number of processors increases. One can easily see that the inter-processor communication time takes the significant portion of the total execution time especially in the case of 32 subdomains. There is a slight difference of the proportions of the inter-processor communication time between one iteration case and the entire simulation case. Since the factorization phase doesn't take part in every iteration of the entire simulation, the proportion of inter-processor communication time for the entire simulation case, becomes higher than the counterpart for one iteration case with factorization.

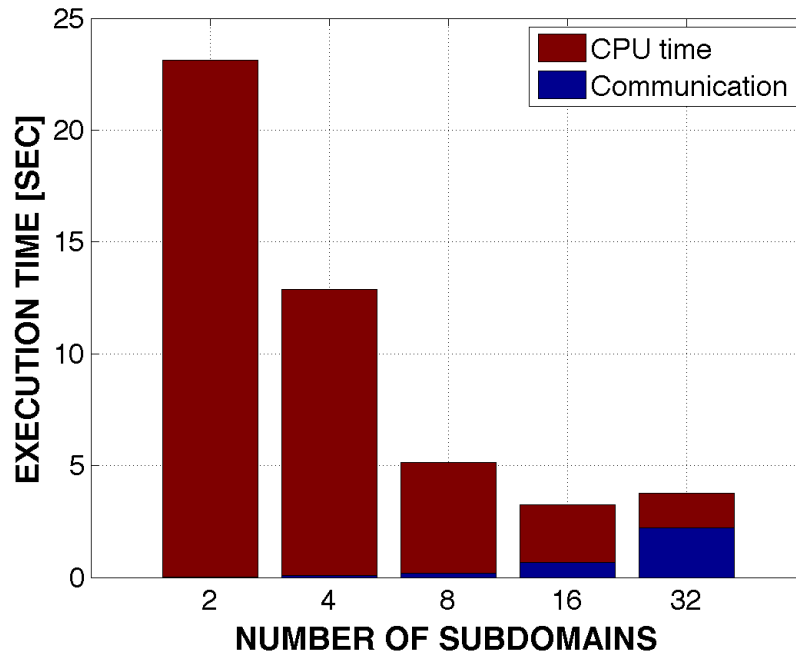
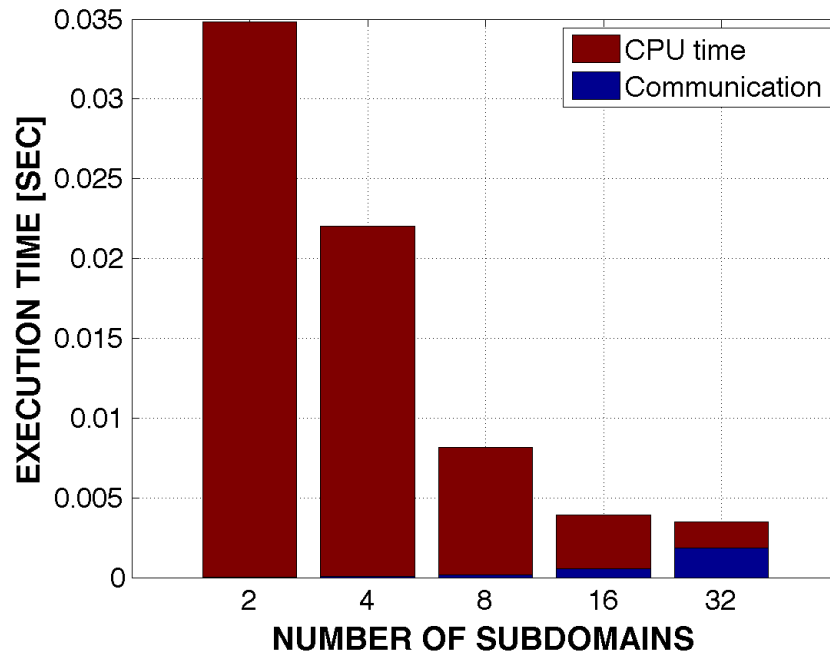


Figure 5.12: Grid of beams (MPMD): Inter-processor communication time (Top: for one iteration with factorization; Bottom: for the entire simulation)

5.4.7 Contributions of sub-phases

Contributions of the sub-phases of the solution procedure are visualized in Figures 5.13 and 5.14. Figure 5.13 is for the SPMD case while Figure 5.14 is for the

MPMD case. As stated in Section 5.2.6, note that some of the bar graphs don't reach 100% in total because minor sub-phases less than 2% have been excluded from the graphs.

In the SPSPD case (the first bar graph), this typical structural problem shows the typical distribution of the execution time proportions. For one iteration with factorization case, computations and assemblies of the element matrices and arrays take about 10% of the total execution time, LU factorization does about 60% and forward-back-substitution does about 30%. On the other hand, for the entire simulation case, the modified Newton-Raphson process remarkably affects the distribution as 15%, 35% and 50% in the same order as the previous. The execution time for the factorization considerably reduces while the counterpart for the forward-back-substitution boosts up.

In the SPMD case, it doesn't exhibit inter-processor communication time and a single processor processes all sub-phases. It is clearly shown that the execution time proportion for the computations and assemblies of the subdomain element matrices and arrays, steadily increases while the counterpart for the forward-back-substitution phase steadily decreases as the number of subdomains increases. This graphs also clearly indicate the decrease in the execution time proportion for the LU factorization of the interface matrix, with the four subdomain case as mentioned in Section 5.4.5.

In the MPMD case, the most significant change can be found from the inter-processor communication time. In the case of 32 subdomains, the execution time proportion for the inter-processor communication is dominant taking about 60% of the total execution time. The inter-processor communication time is the main bottleneck of the parallel solution procedure.

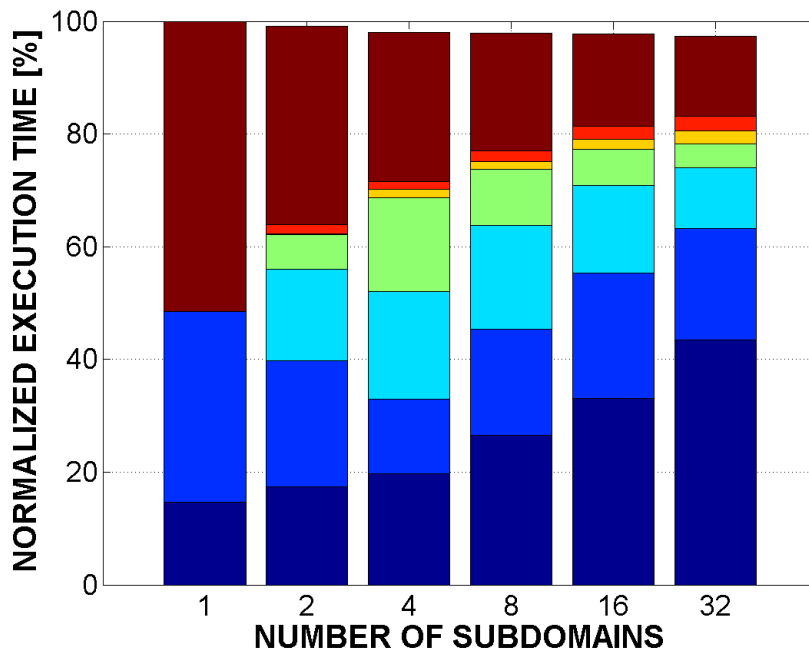
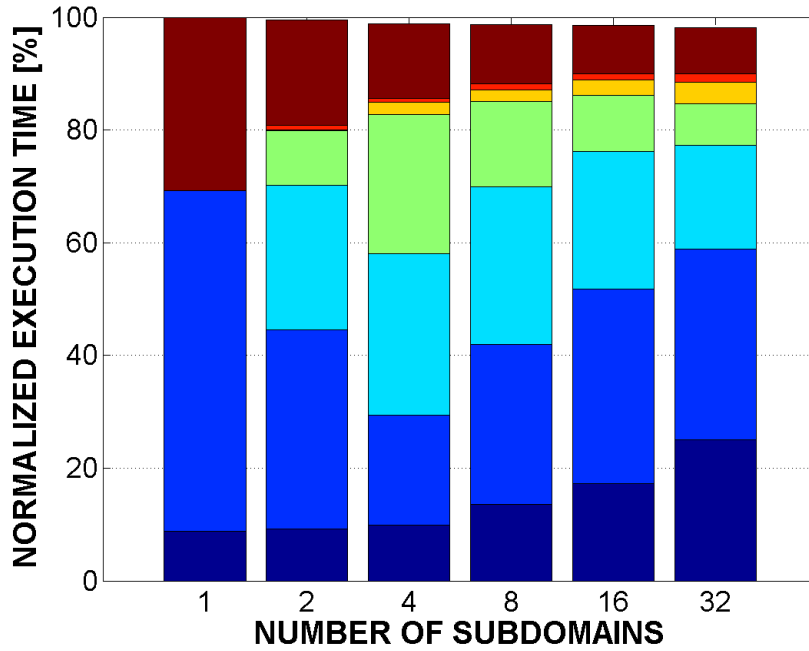
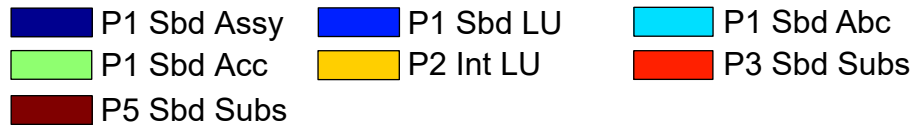


Figure 5.13: Grid of beams (SPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution)

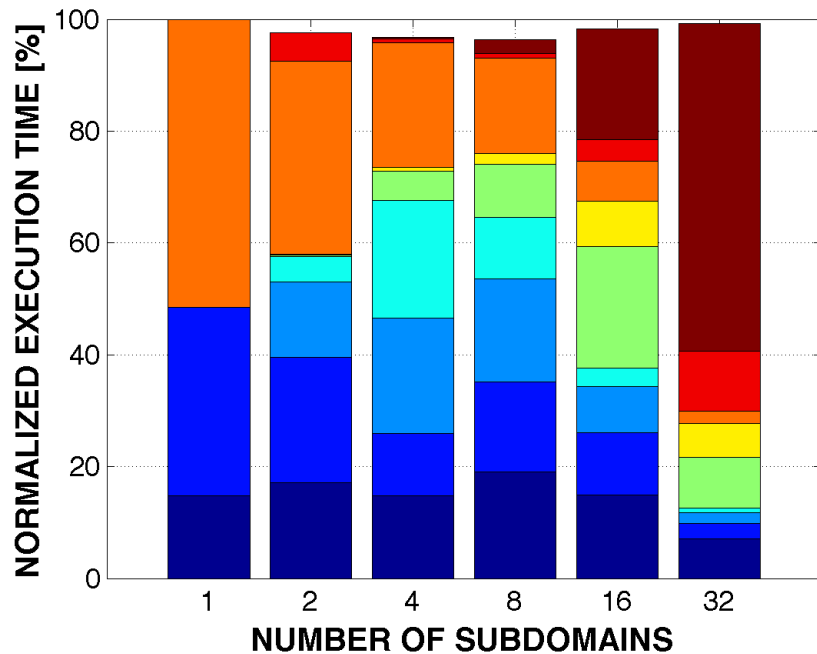
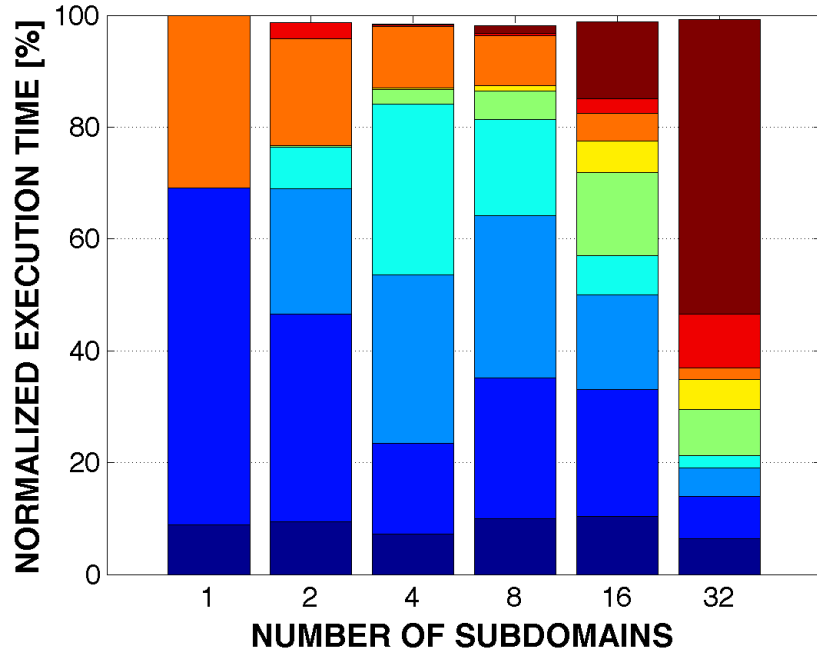


Figure 5.14: Grid of beams (MPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution; MPI Wait: Processor synchronization; MPI Comm: Inter-processor communication)

5.5 Rotor System

The third example is a helicopter *rotor system* which is one of the most representative and complex multibody systems. The rotor system is usually composed of various mechanical components such as rotor blades, a rotor hub, swash plates, pitch links, inner blade parts, etc. The mechanical components are modeled with rigid elements, flexible elements such as beams, plates or shells as well as constraint elements which represent mechanical joints such as revolute joints, prismatic joints, spherical joints, universal joints, etc.

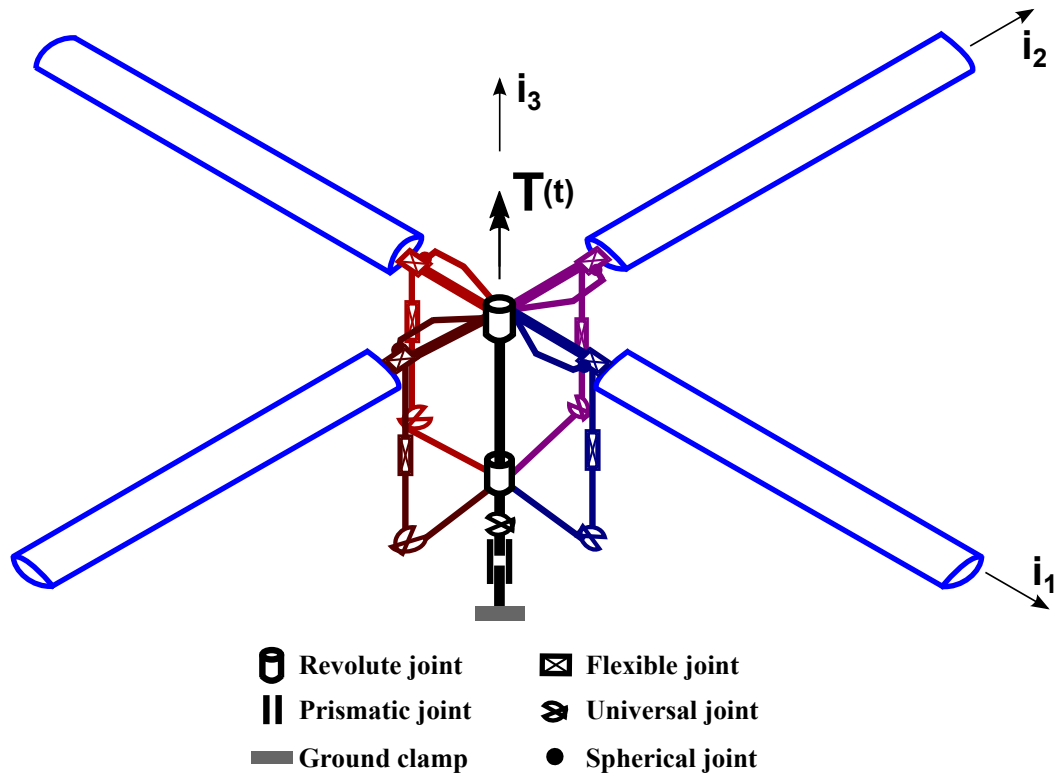


Figure 5.15: Rotor system: Four-bladed hingeless flexbeam rotor system under a driving torque at the hub

The parallel algorithm with the proposed domain decomposition method has been developed for and focused on flexible multibody systems which are distinguished from typical structures without mechanical joints. If the proposed parallel algorithm works well for this complex multibody system, it is expected that the algorithm can be

further developed for and extended to advanced application areas.

5.5.1 Model description

The rotor system under consideration is a four-bladed hingeless flexbeam rotor system as seen in Figure 5.15. Thus the inner blade configuration of this rotor system is simpler than that of the fully articulated rotor system. Each rotor blade is 17.5 ft long. The cross sectional properties of the rotor blade vary along the blade span and they are depicted in Figure 5.16; mass per unit span, torsional stiffness and two bending stiffnesses for flap and lead-lag, are plotted in the figure set. The total dofs from the finite element meshing of the unpartitioned rotor system is 10,288 and each rotor blade has 128 cubic (four-node) beam elements.

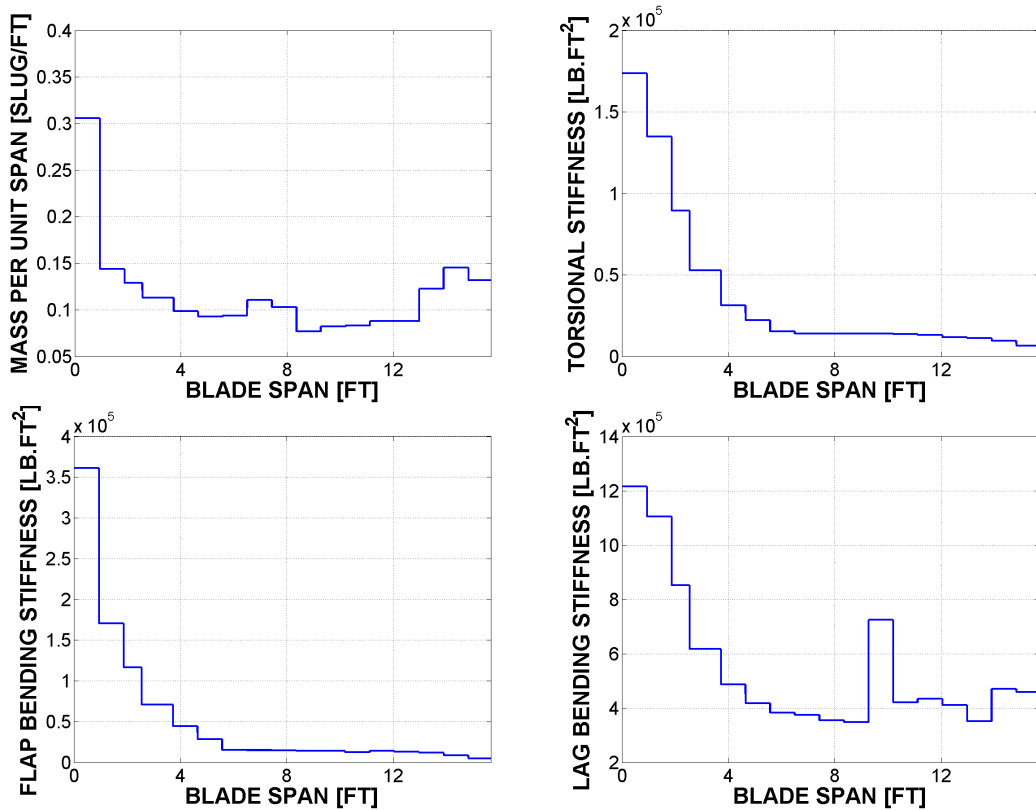


Figure 5.16: Rotor system: Blade sectional properties

5.5.2 Domain decomposition

The proposed domain decomposition method applies to the rotor system to partition the system into non-overlapping subdomains. Due to the special geometry of the rotor system, the partitioning has been performed as seen in Figure 5.17. Both subsystems of the rotor hub and the swash plates are grouped as *Subdomain 1*. Because of the four blades, there are four sets of three subsystems which are the inner blade and the pitch link and the rotor blade. Out of the three subsystems, a pair of the inner blade and the pitch link is assigned to a subdomain so that the four sets of the pair are assigned as *Subdomain 2*, *Subdomain 3*, *Subdomain 4* and *Subdomain 5*. The rest four rotor blades are assigned as *Subdomain 6*, *Subdomain 7*, *Subdomain 8* and *Subdomain 9*.

The further domain decomposition is, however, applied only to the four blades by 2^N partitioning, see Figure 5.18, where $N = 0, 1, \dots, 4$ while the first 5 subdomains (*Subdomains 1 ~ 5*) are fixed and not further partitioned for more than 9-subdomain cases. Thus, except for the unpartitioned case, the numbers of subdomains (N_s) are calculated as

$$N_s = 5 + 4 \cdot 2^N. \quad (5.11)$$

Therefore, there are six domain decomposition cases in total and the problem sizes in dofs for all cases are organized in Table 5.17 and depicted in Figure 5.19. Because there are multiple subdomains for each domain decomposition case, each subdomain dofs is an averaged value for all subdomains. It should be noted that the averaged dofs for every case are rounded for easier reading in the table. The total dofs are the sum of the dofs of an interface problem and all subdomain problems.

For the domain decompositions of the rotor system, as the number of subdomains increases, the averaged dofs of a subdomain problem decreases at a constant rate while the interface problem size increases at the similar constant rate, as seen in Figure 5.4.

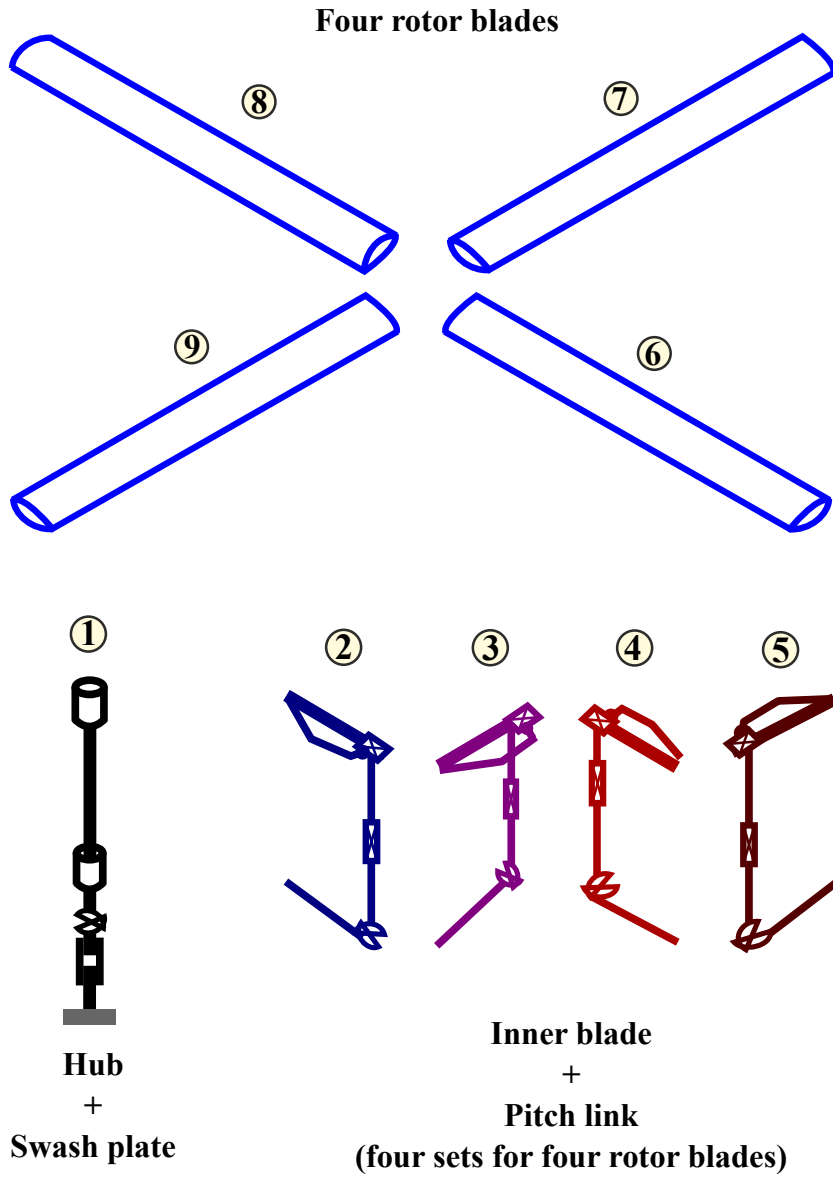


Figure 5.17: Rotor system: Domain decomposition

Table 5.17: Rotor system: Problem sizes for 6 domain decomposition cases

Number of subdomains	Subdomain dofs (Avg.)	Interface dofs	Total dofs
1	10288	N/A	10288
9	1171	60	10600
13	816	84	10696
21	512	132	10888
37	298	228	11272
69	168	420	12040

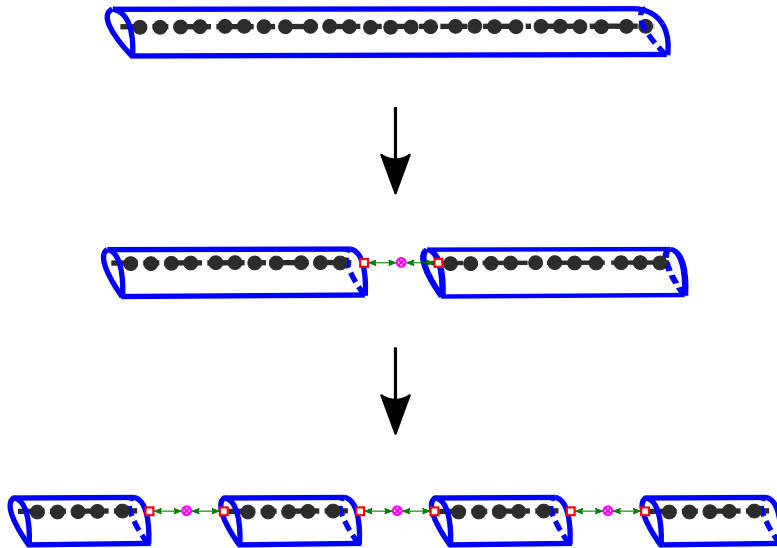


Figure 5.18: Rotor system: 2^N partitioning of each rotor blade

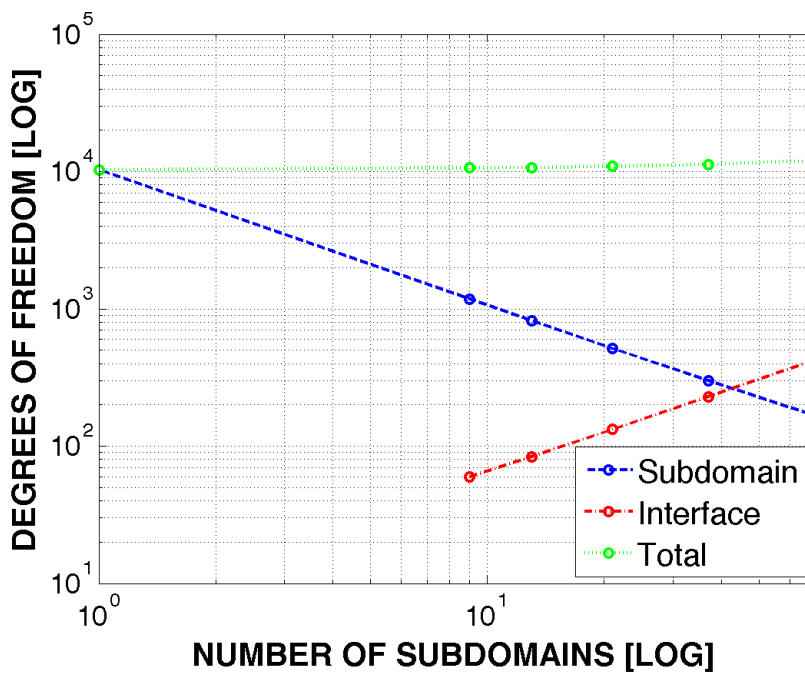


Figure 5.19: Rotor system: Problem sizes for 6 domain decomposition cases

Similarly to the grid of beams example, as the subdomain size gets smaller, the total dofs slightly increases because of the additional independent interface nodes and localized Lagrange multiplier nodes. It can be easily found from Table 5.17 that the interface dofs and the subdomain (averaged) counterpart cross over between the

37-subdomain and 69-subdomain cases. Note that, for the multi-processors for multi-domain (MPMD) simulation cases, only one processor is assigned to each subdomain.

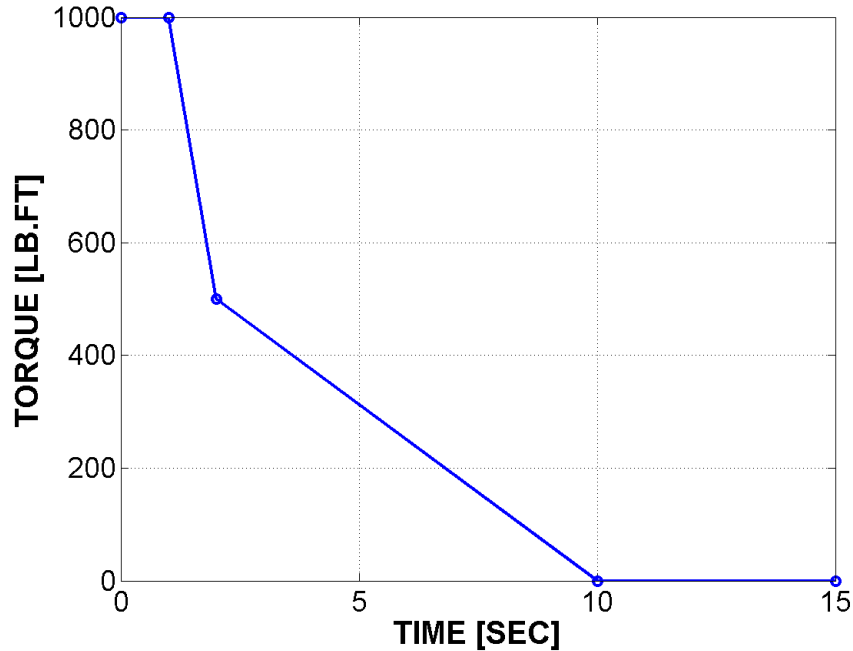


Figure 5.20: Rotor system: Time history of driving torque

5.5.3 Predictions of dynamic response

The dynamic simulations of the Rotor system have been performed for 3,000 steps with a constant step size of $\Delta t = 5.0 \times 10^{-3}$ sec. The rotor hub is subjected to a driving torque $T(t)$ which changes in time as seen in Figure 5.20. The driving torque generates about three revolutions of the rotor during the dynamic simulations. The convergence tolerance for the iteration termination has been set to 10^{-6} .

The dynamic responses of the rotor system are investigated at the tip of a blade for the validation of the proposed parallel algorithm. The single processor for the single domain case (SPSD) and the multi-processors for the 69-subdomain case (MPMD) are compared as shown in Figures 5.21, 5.22, 5.23 and 5.24.

All figure sets are configured as the following: the time histories of translational or rotational response predictions from both sequential and parallel computations are

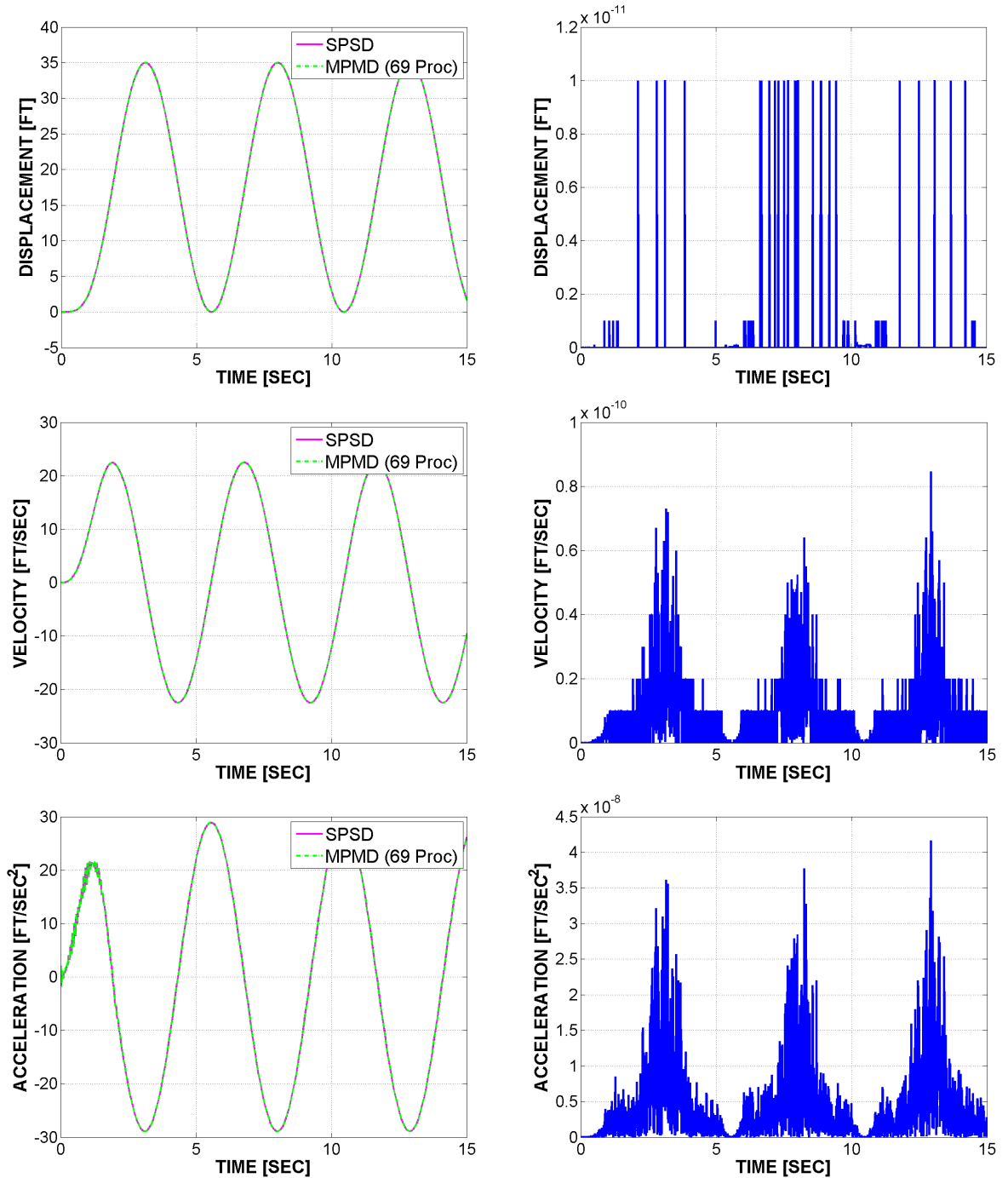


Figure 5.21: Rotor system: Predictions of in-plane translational (along \bar{v}_2) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains

on the left of the figure sets while the response differences between sequential and parallel computations are on the right. The first two figure sets, Figures 5.21 and 5.22,

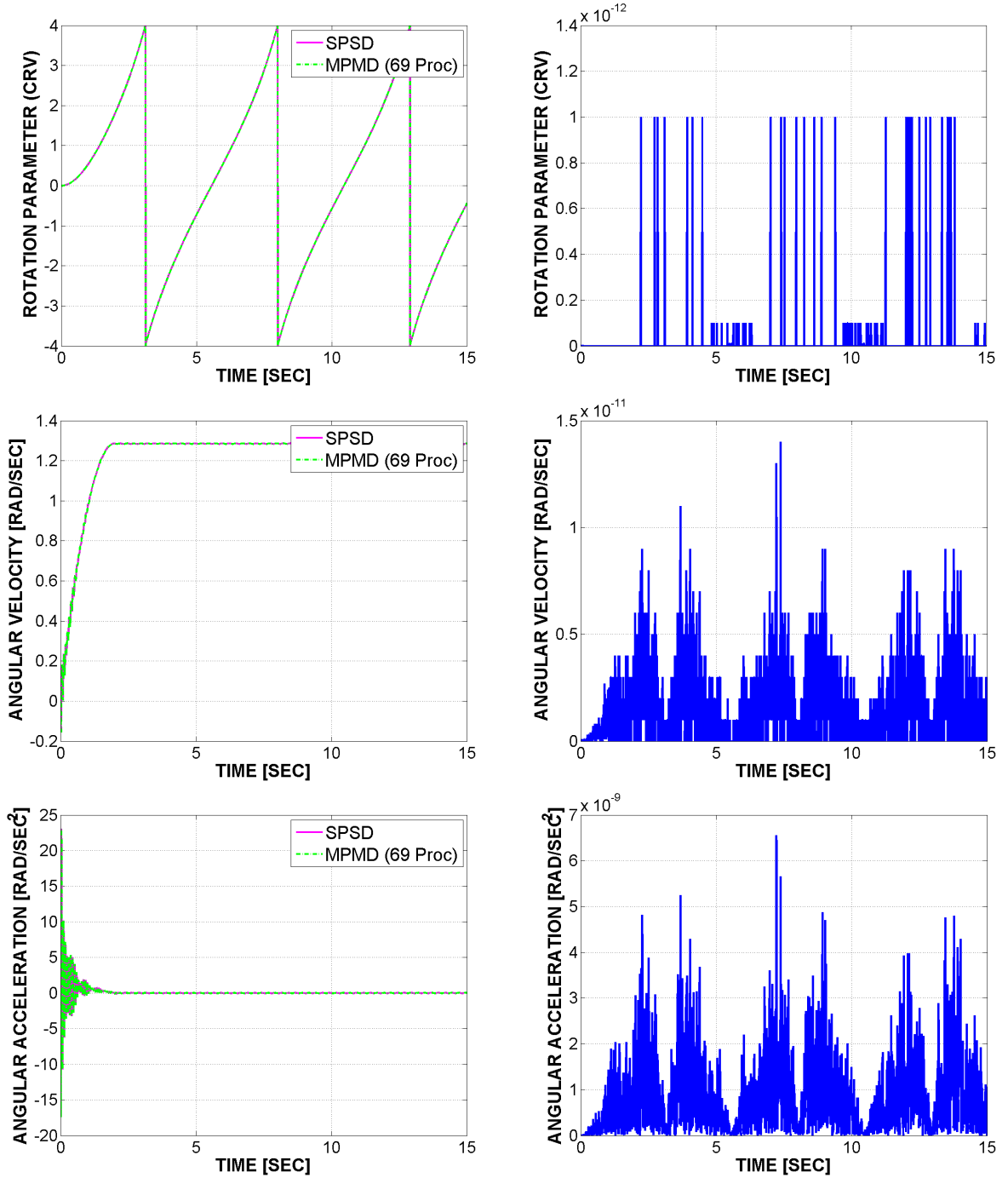


Figure 5.22: Rotor system: Predictions of in-plane rotational (about \bar{v}_3) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains

are about the rotor in-plane responses and the last two figure sets, Figures 5.23 and 5.24, are about the rotor out-of-plane counterparts. Here again, both sequential

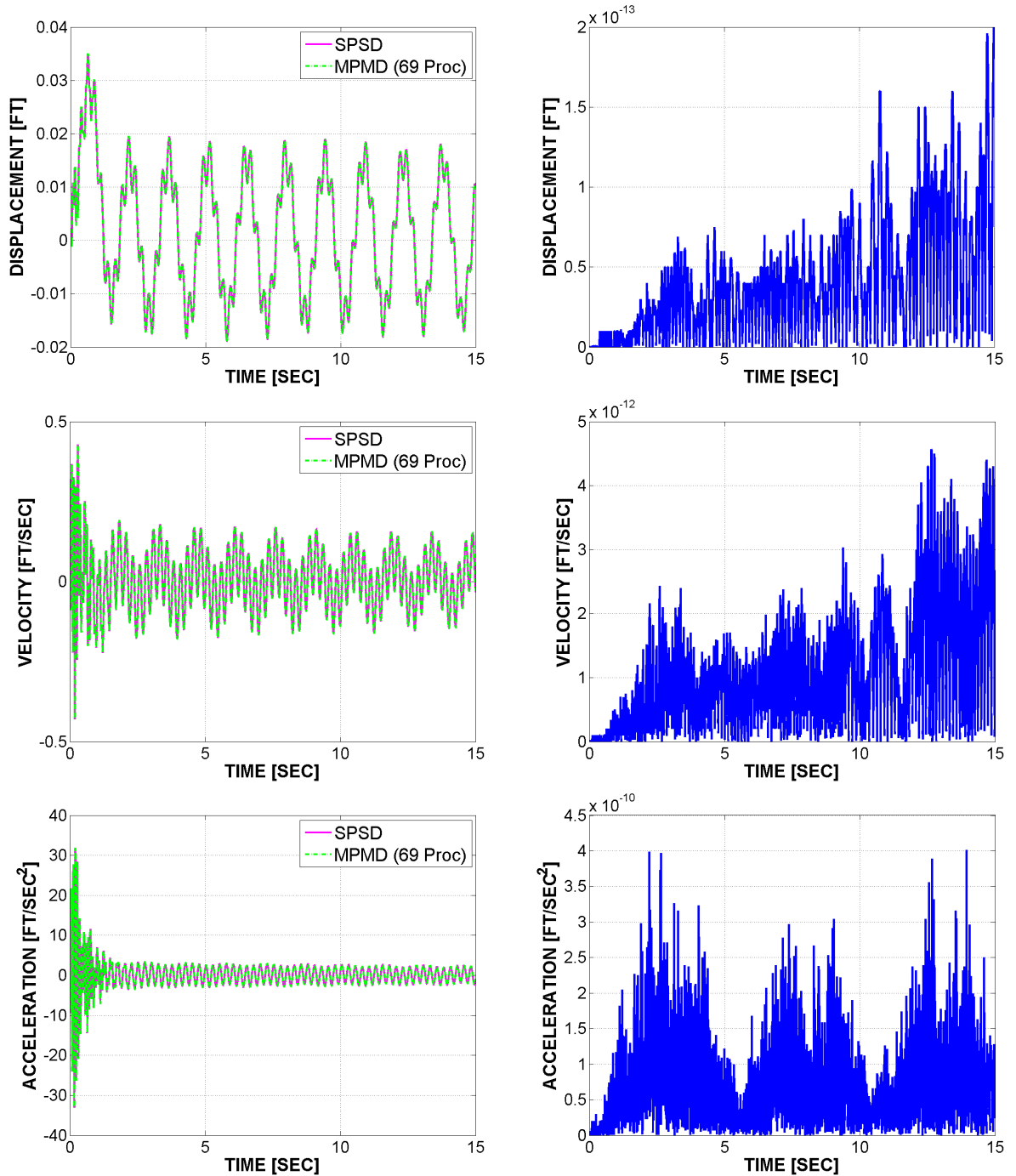


Figure 5.23: Rotor system: Predictions of out-of-plane translational (along \bar{l}_3) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains

and parallel predictions are kept identical within machine accuracy during all the dynamic simulations.

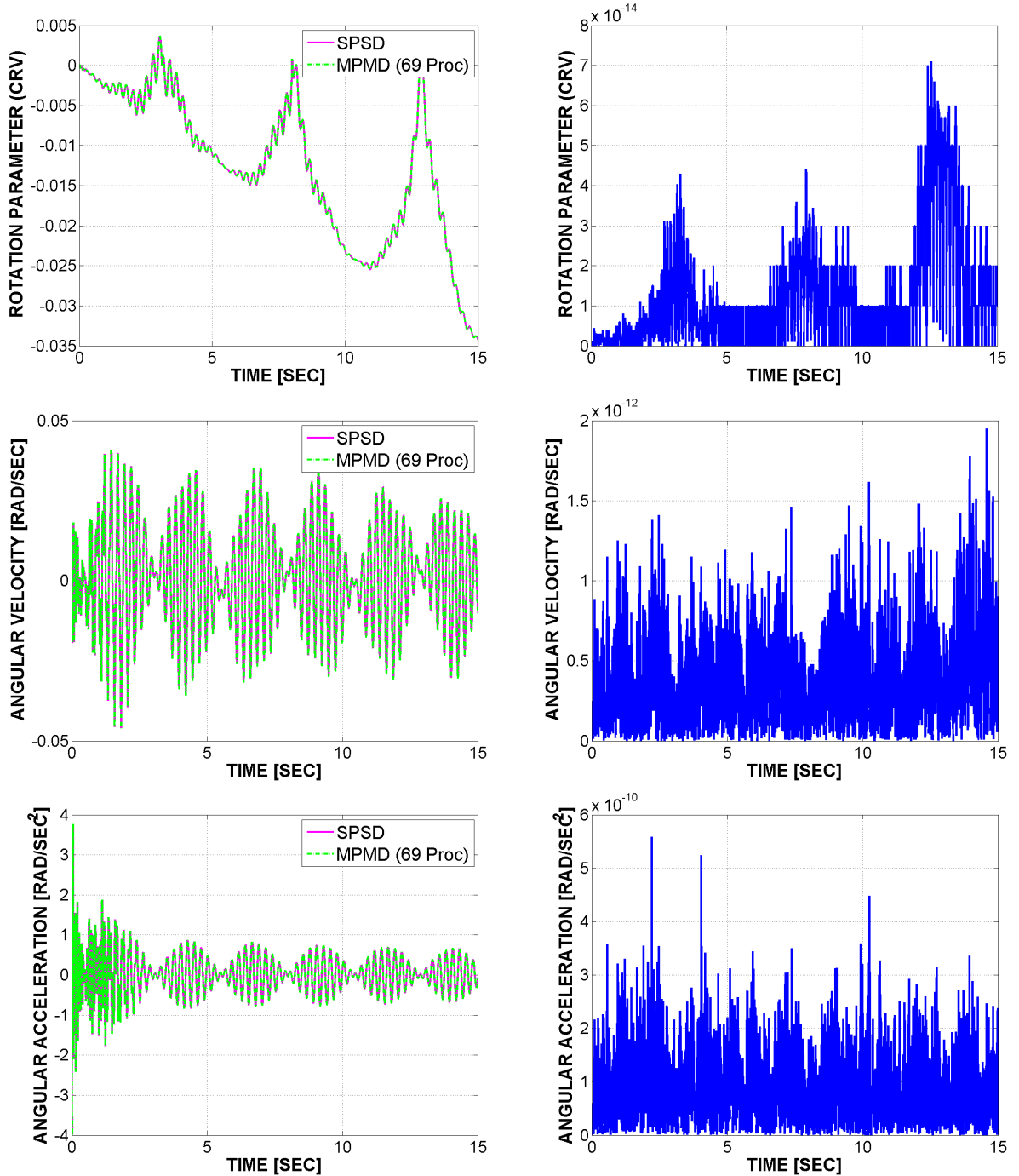


Figure 5.24: Rotor system: Predictions of out-of-plane rotational (about \bar{v}_2) responses at blade tip (Left) and their differences (Right) between two cases: Single domain vs. 69 subdomains

5.5.4 Performance for solution procedure

Through the dynamic simulations with the rotor system, the parallel performance has been evaluated. The base metrics, CPU times have been measured for the various

domain decomposition cases and the derived metrics, speedup and efficiency, are then calculated. The CPU times or the execution times are measured for the duration for the solution procedure only, which is explained in Section 3.5. The CPU time and the speed-up are listed in Tables 5.18 and 5.19 for the two different periods: for one iteration with factorization and for the entire simulation. As stated earlier, the difference between the two different periods are characterized by the modified Newton-Raphson iteration process. For one iteration with factorization, the domain decomposition with 37 subdomains has achieved the shortest CPU time for both SPMD and MPMD cases with the maximum speed-up of 10.08. On the other hand, for the entire simulation, the SPMD and MPMD cases have obtained the shortest CPU times with 69 and 21 subdomain cases, respectively. But it should be noted that, it is unavoidable that the speed-up starts to drop at a performance limit for this example problem as well.

Table 5.18: Rotor system (for one iteration with factorization): CPU time, speed-up and efficiency for solution procedure

Number of subdomains	CPU time [sec]		Speed-up	Efficiency
	SPMD	MPMD		
1	0.3519	0.3519	1.00	1.00
9	0.0949	0.0215	4.40	0.49
13	0.0879	0.0129	6.80	0.52
21	0.0772	0.0080	9.60	0.46
37	0.0734	0.0073	10.08	0.27
69	0.0750	0.0098	7.65	0.11

The measured CPU times for various cases are plotted in Figure 5.25. For both one iteration and the entire simulation cases, the CPU times decrease at first but tend to be going back up for more than 37 subdomains.

The ratio of CPU time to the SPMD case and the speed-up are plotted in Figure 5.26 and tabulated in Tables 5.20 and 5.21. The performance by the effect of domain decomposition only (SPMD, solid line in blue) slowly increases at first and stays almost flat while the speed-up (dashed line in red) starts to noticeably decrease

Table 5.19: Rotor system (for the entire simulation): CPU time, speed-up and efficiency for solution procedure

Number of subdomains	CPU time [sec]		Speed-up	Efficiency
	SPMD	MPMD		
1	2646.90	2646.90	1.00	1.00
9	749.04	177.68	4.22	0.47
13	593.64	88.60	6.70	0.52
21	489.41	58.91	8.31	0.40
37	445.26	60.59	7.35	0.20
69	441.83	89.93	4.91	0.07

at some performance limit for both period cases. The overall performance with the domain decomposition and the multi-processors (MPMD, dash-dot line in green), is remarkable until the domain decomposition with 37 subdomains. Again, the overall performance is degraded mainly by the parallel processing after some performance limit.

Table 5.20: Rotor system (for one iteration with factorization): Ratio of CPU time to SPMD case and speed-up

Number of subdomains	Ratio to SPMD		Speed-up	Efficiency
	SPMD	MPMD		
1	1.00	1.00	1.00	1.00
9	3.71	16.33	4.40	0.49
13	4.00	27.22	6.80	0.52
21	4.56	43.74	9.60	0.46
37	4.79	48.32	10.08	0.27
69	4.69	35.91	7.65	0.11

As observed in the plots of the performance ratio to the SPMD case or the speed-up, the overall performance (MPMD) provides efficiency over one up to the 37-subdomain cases, as seen in Figure 5.27, because of the remarkable effect combining both the domain decomposition and the parallel processing. Since this metric applies only to multi-processor cases, the SPMD case cannot be shown here.

As for the comparison between one iteration and the entire simulation cases, the saturation or degradation of the speed-up for one iteration with factorization case is

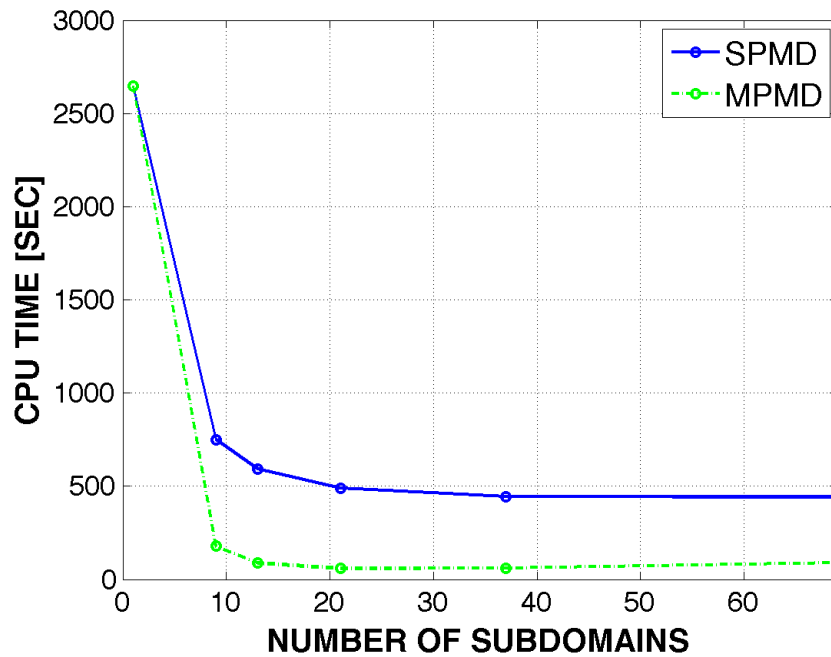
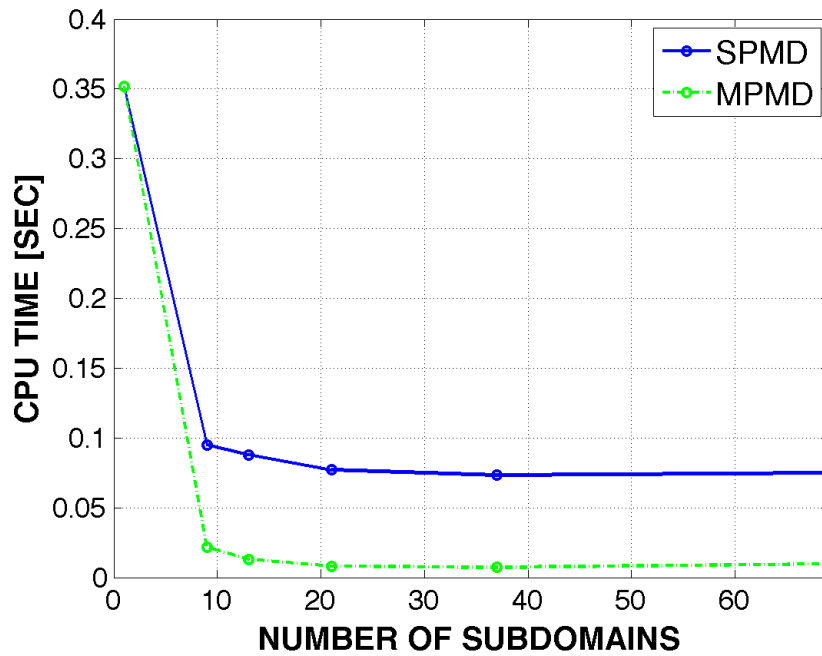


Figure 5.25: Rotor system: CPU time for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

relatively delayed compared to the counterpart for the entire simulation case because of the modified Newton-Raphson iteration process.

The parallel efficiency for this problem appears to be poor for multiple subdomain

Table 5.21: Rotor system (for the entire simulation): Ratio of CPU time to SPSP case and speed-up

Number of subdomains	Ratio to SPSP		Speed-up	Efficiency
	SPMD	MPMD		
1	1.00	1.00	1.00	1.00
9	3.71	16.33	4.22	0.47
13	4.00	27.22	6.70	0.52
21	4.56	43.74	8.31	0.40
37	4.79	48.32	7.35	0.20
69	4.69	35.91	4.91	0.07

cases. Again, the size of the problem is not large and this leads to the low parallel efficiency. The efficiency is expected to be better for the problems with larger dofs.

5.5.5 Performance for LU factorization phase

In order to study the computational cost of the LU factorization phase with parallel simulations, both the cost estimation and the actual execution time for the factorization phase are compared and correlated with various domain decomposition cases. The estimates of the factorization cost are organized in Tables 5.22 and 5.23 for the subdomain and interface problems, respectively. Unlike the grid of beams example,

Table 5.22: Rotor system: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for a subdomain problem

Number of subdomains	Degrees of freedom		Required size of memory [kB]		Mean bandwidth		LU factorization cost index	
	Max	Avg	Max	Avg	Max	Avg	Max	Avg
	1	10288	←	5220	←	64	←	42139648
9	2358	1171	552	274	29	28	1983078	981470
13	1212	816	285	191	30	28	1090800	706139
21	636	512	150	119	30	29	572400	436428
37	348	298	82	69	30	29	313200	252411
69	263	168	62	39	30	29	221183	141364

the load balancing for this problem is poor when the maximum and averaged dofs are compared for each domain decomposition case. This is because the geometry of the rotor system is special and no optimal partitioner was used for domain decomposition.

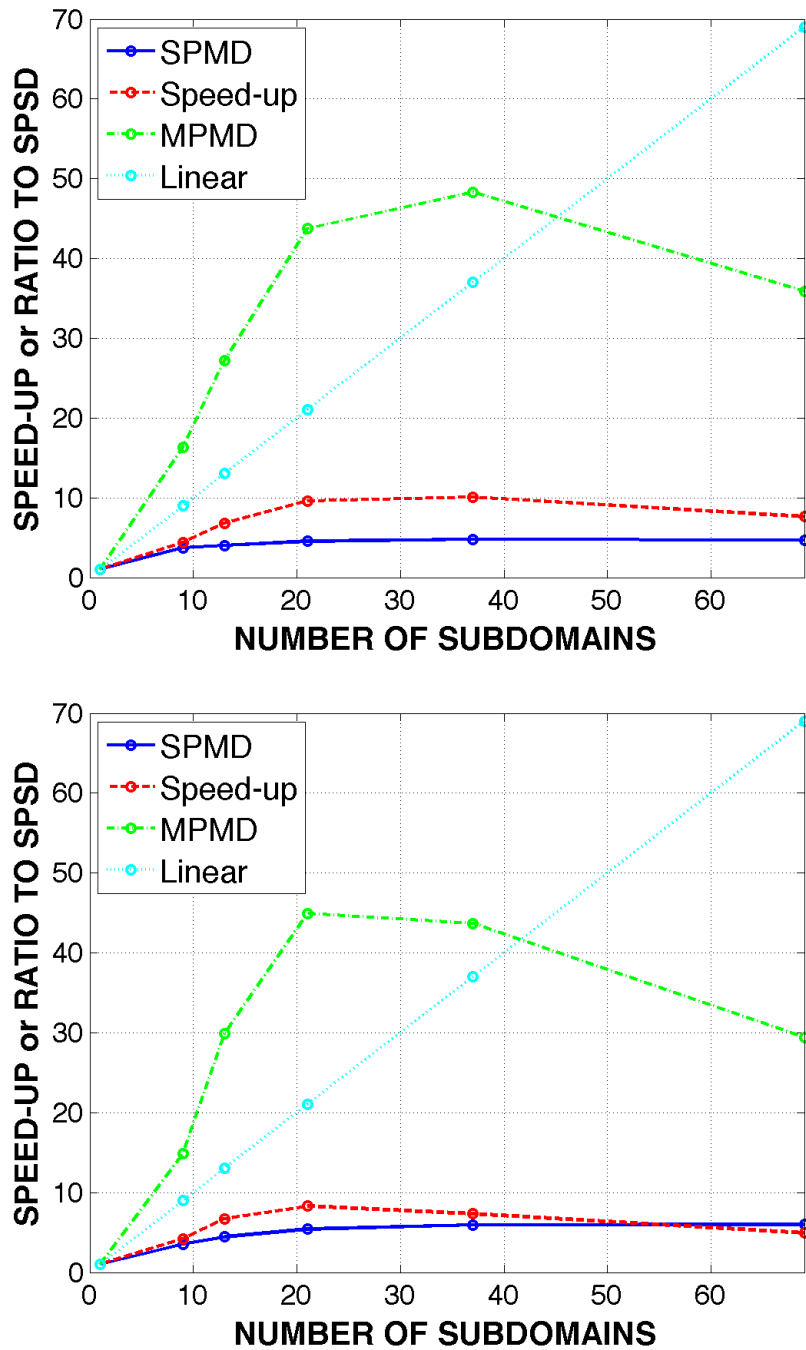


Figure 5.26: Rotor system: Speed-up or ratio of CPU time to SPSD case for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

As explained in Section 5.2.4, the factorization cost is proportional to the dofs while it is so to the square of the mean bandwidth of the tangent stiffness matrix.

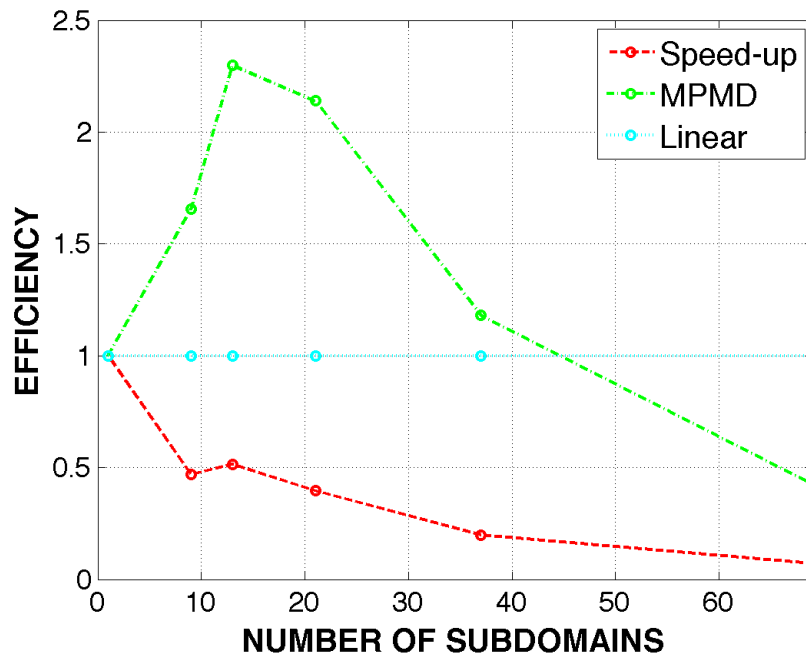
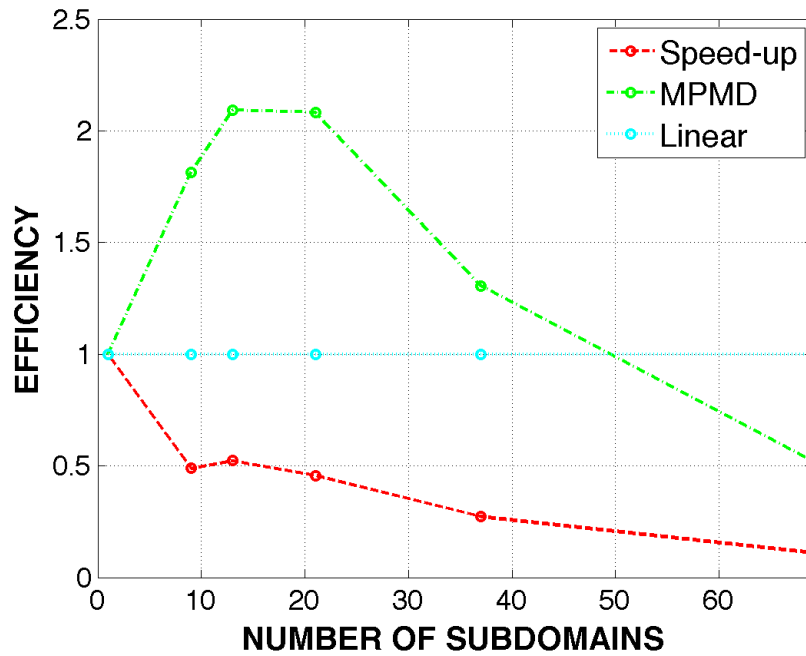


Figure 5.27: Rotor system: Efficiency for solution procedure (Top: for one iteration with factorization; Bottom: for the entire simulation)

As the number of subdomains increases, the factorization cost index of the subdomain matrix decreases while the counterpart of the interface matrix increases as expected. Therefore, the factorization speed-up and speed-down for the subdomain

Table 5.23: Rotor system: Degrees of freedom, required memory size, mean bandwidth and LU factorization cost index for an interface problem

Number of subdomains	Degrees of freedom	Required size of memory [kB]	Mean bandwidth	LU factorization cost index
9	60	28	60	216000
13	84	44	67	377076
21	132	58	56	413952
37	228	85	47	503652
69	420	139	42	740880

matrix and the interface matrix, respectively, are expected to monotonically increase. It is interesting to note that the mean bandwidth of the subdomain stiffness matrix remains flat as the number of subdomains increases. It should also be noted that the factorization cost index of the interface matrix first exceeds the counterpart of the subdomain matrix for the 37-subdomain case.

Table 5.24: Rotor system (MPMD, for the entire simulation): LU factorization CPU time and its percentage of the total simulation time

Number of subdomains	Total	Subdomain		Interface	
	simulation time [sec]	Factorization time [sec]	Percentage [%]	Factorization time [sec]	Percentage [%]
1	2646.90	355.11	13.42	N/A	N/A
9	177.68	14.69	8.27	1.45	0.82
13	88.60	7.59	8.57	2.34	2.64
21	58.91	4.02	6.82	2.77	4.70
37	60.59	2.24	3.70	3.57	5.89
69	89.93	1.29	1.43	5.27	5.86

Figure 5.28 shows the speed-up and the speed-down for the factorization of the subdomain matrix and the interface matrix, respectively. The relative performance evaluations are from both the cost estimations and the actual execution times for the factorization. As seen in the figure, The estimations and the actual experiments have a good correlation. Table 5.24 shows the actual execution time of the factorization phase and its percentage relative to the total simulation time. As expected in the previous paragraph, the factorization times of the subdomain matrix and the interface

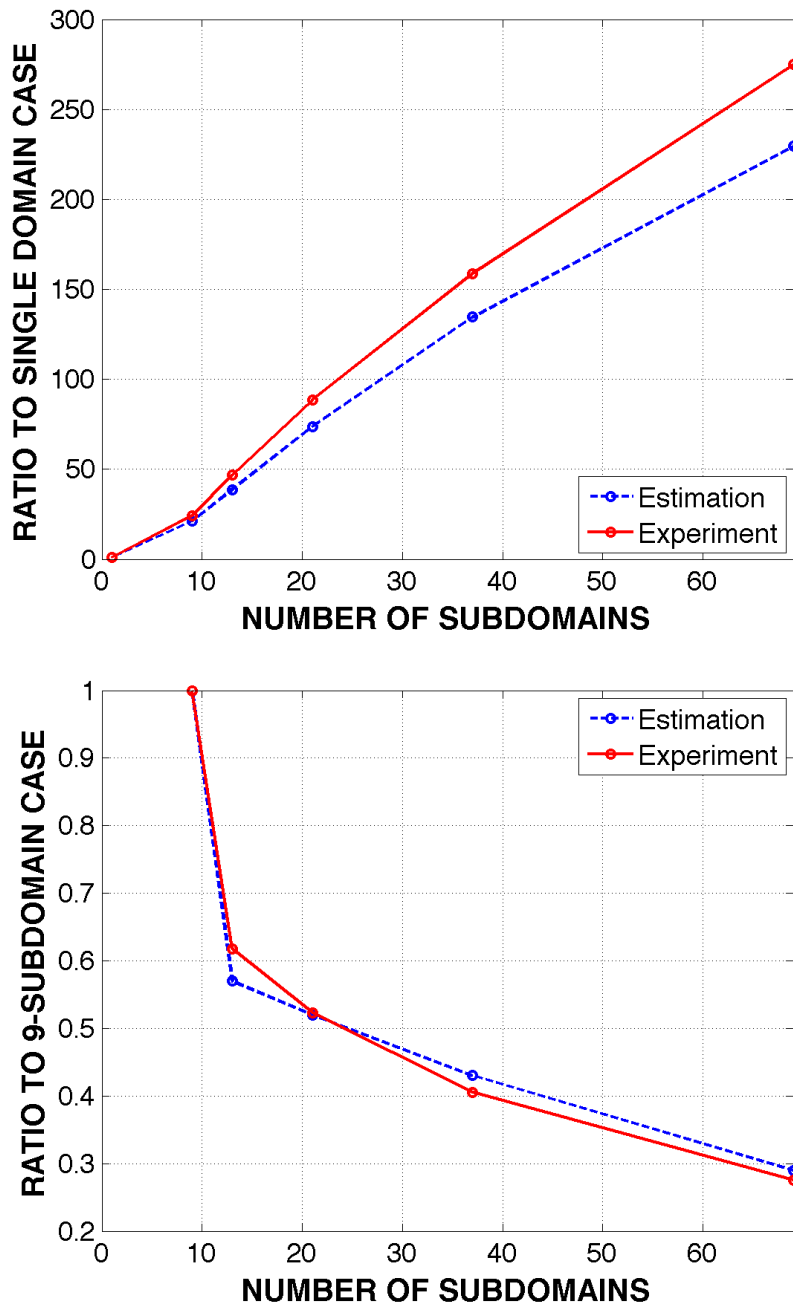


Figure 5.28: Rotor system (MPMD): LU factorization performance (Top: Ratio of performance to single domain case for a subdomain problem; Bottom: Ratio of performance to 9-subdomain case for an interface problem)

matrix cross over for the 37-subdomain case; the factorization of the interface matrix takes more CPU time than the counterpart of the subdomain matrix for the 37 or more subdomain cases.

5.5.6 Inter-processor communication time

For this rotor system problem, the inter-processor communication time remarkably increases as the number of subdomains or processors increases. The size of the transferred data through MPI can be calculated by the number of LLM elements. The total size of the transferred data and the actual inter-processor communication time through MPI are tabulated, see Table 5.25.

Table 5.25: Rotor system (MPMD, for the entire simulation): Dofs for all LLM elements, total MPI data size and inter-processor communication time

Number of subdomains	Dofs for all LLM elements	Total MPI data size [kB]	Total simulation time [sec]	Total MPI communication time [sec]	Time percentage [%]
9	156	27	177.68	4.00	2.25
13	204	35	88.60	9.42	10.63
21	300	46	58.91	17.62	29.91
37	492	69	60.59	29.52	48.72
69	876	114	89.93	54.42	60.51

As stated in Section 5.2.5, the information about the overhead time of the sender and receiver of the transferred data, is not easily available. When the total size of the transferred data and the actual inter-processor communication time are compared, they seem to have some correlation but not so close.

It is confirmed again that, the inter-processor communication time takes the significant portion of the total execution time as seen in Figure 5.29. But, unlike the other example problem, Grid of beams, there is a less difference between one iteration case and the entire simulation case. Because the factorization phase is not the most dominant phase, which is explained in Section 5.5.7, the modified Newton-Raphson process doesn't make a large difference between the two cases.

5.5.7 Contributions of sub-phases

Contributions of the sub-phases are plotted in Figures 5.30 and 5.31. Figure 5.30 is for the SPMD case while Figure 5.31 is for the MPMD case. Here again, as stated

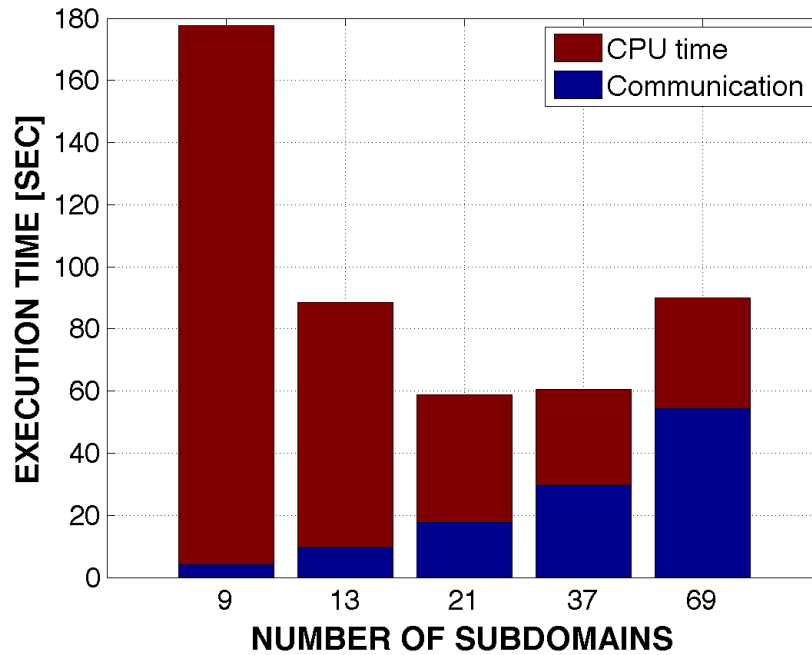
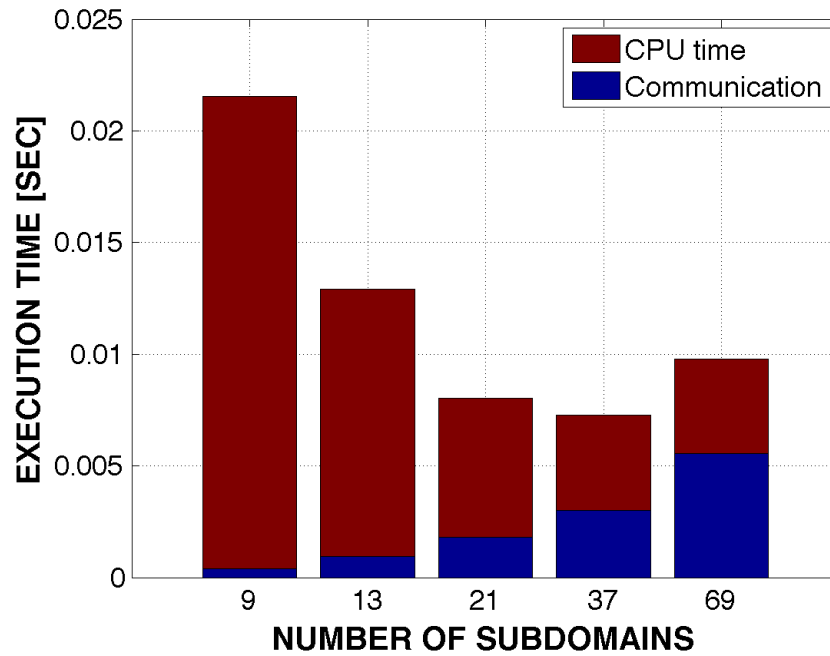


Figure 5.29: Rotor system (MPMD): Inter-processor communication time (Top: for one iteration with factorization; Bottom: for the entire simulation)

in Section 5.2.6, note that some of the bar graphs don't reach 100% in total because minor sub-phases less than 2% have been excluded from the graphs.

In the SPMD case (the first bar graph), the most expensive sub-phase is the

back-substitution unlike typical structural problems. This is because the system matrix of the unpartitioned Rotor system problem has a very narrow bandwidth. The very narrow bandwidth stems from the fact that each rotor blade is sliced into 128 elements which are many. Thus the most dofs are concentrated on the rotor blades and the finite element discretization of them renders the resulting stiffness matrix very sparse. This fact can be checked in Table 5.22. The dofs (n) of the problem is 10,288 while the mean bandwidth, m , is just 64. Considering the factorization index which is proportional to nm^2 , m^2 is just 4,096 which is less than the half of n . Thus the computational cost for the factorization phase is expected to be relatively small compared to the counterpart for the other phases. That is, the LU factorization won't take the largest portion of the execution times. Rather, the back-substitution phase becomes more dominant because the cost of the back-substitution phase is proportional to n^2 . A simple calculation of the back-substitution cost index is $n^2 = 105,842,944$ which is about 2.5 times of the factorization cost index $nm^2 = 42,139,648$.

In the SPMD case, it can be easily seen that the proportion of the back-substitution phase steadily decreases as the domain decomposition gets finer. This is because the domain decomposition process reduces the dofs of the subdomain problem and this directly affects the cost of the subdomain back-substitution phase (P5 Sbd Subs).

In the MPMD case, as the domain decomposition gets finer, while the proportion of the subdomain back-substitution phase (P5 Sbd Subs) dramatically decreases, the counterpart of the interface matrix assembly and factorization (P2 Int Acc LU) increases. This stems from the fact that, as the domain decomposition gets finer, the subdomain problem dofs decreases while the interface problem dofs increases. The inter-processor communication time grows fast for more and more subdomains and is still the main bottleneck of the parallel solution procedure.

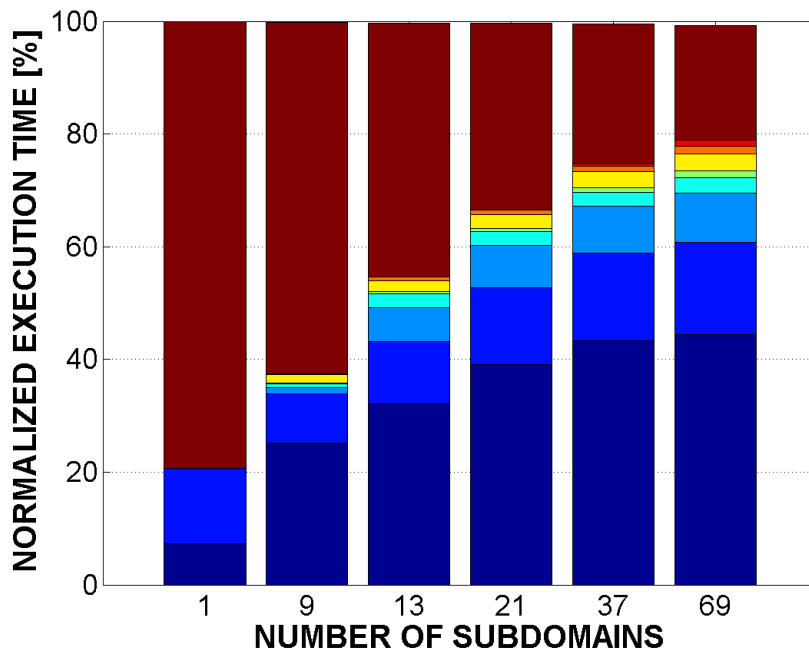
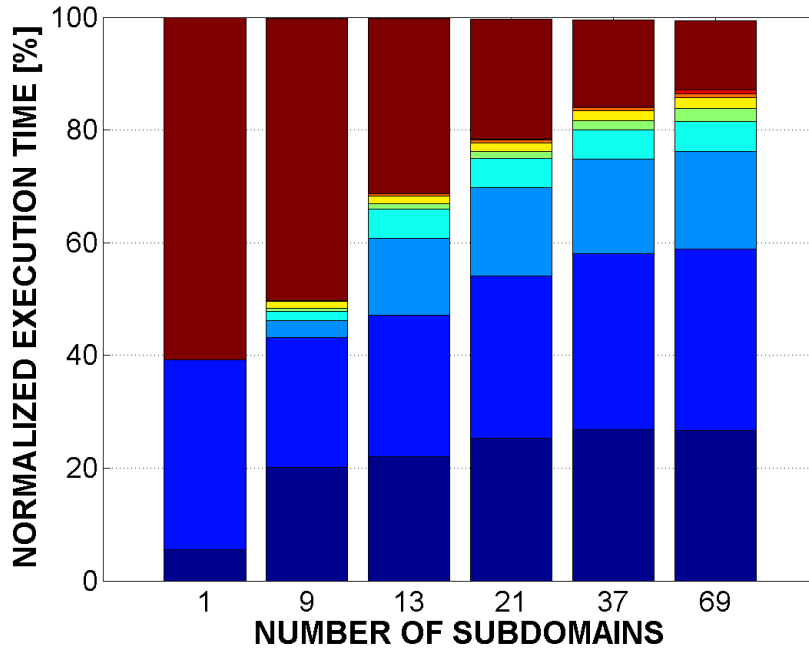
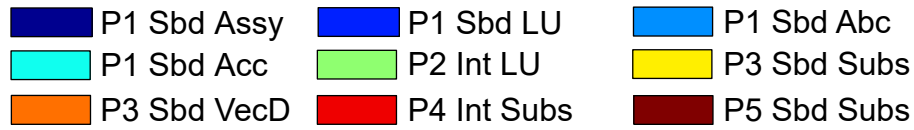


Figure 5.30: Rotor system (SPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution)

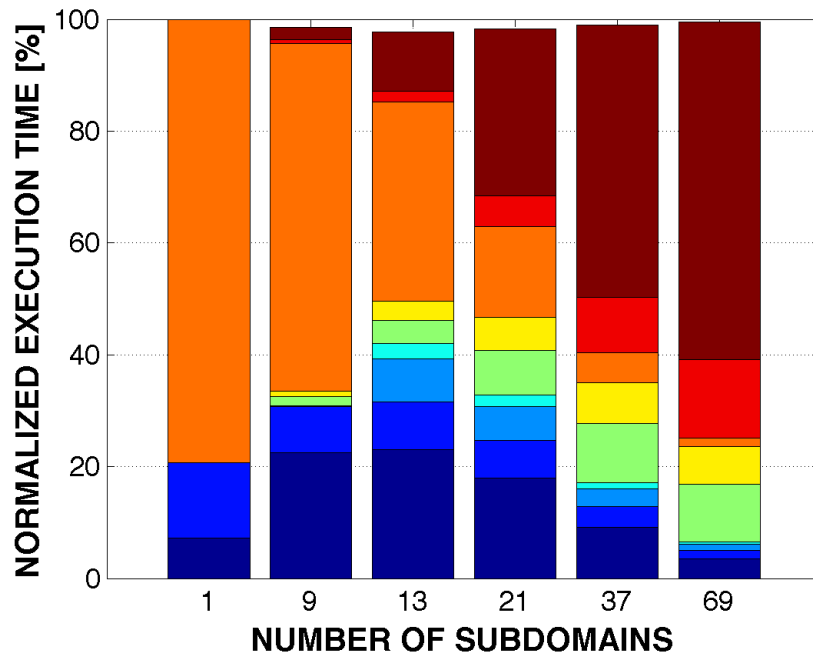
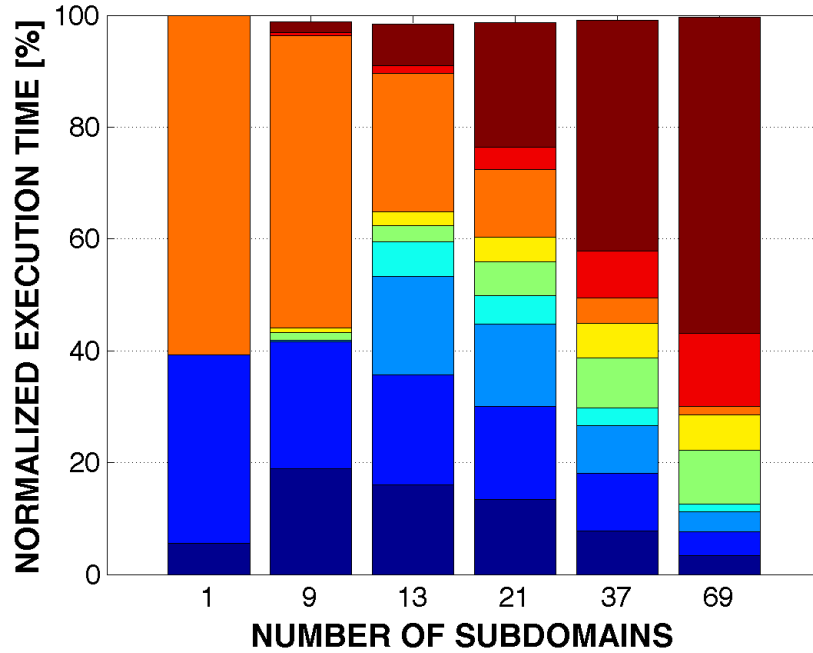


Figure 5.31: Rotor system (MPMD): Normalized execution times for sub-phases (Top: for one iteration with factorization; Bottom: for the entire simulation; P_i : Phase i ; Sbd: Subdomain; Int: Interface; Assy: Element matrix computation and assembly; LU: Factorization; Abc: Partial Sbd-Int coupling matrix computation; Acc: Partial Int matrix computation/assembly; Subs: Forward/Back substitution; MPI Wait: Processor synchronization; MPI Comm: Inter-processor communication)

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

A novel non-overlapping domain decomposition method was developed and implemented to perform flexible multibody dynamics simulations in parallel. The proposed domain decomposition approach partitions a computational domain into non-overlapping subdomains and kinematic constraints are enforced via a localized version of Lagrange multiplier technique to ensure both continuity between the subdomains and better convergence behavior. The localized Lagrange multiplier (LLM) technique introduces independent interface nodes which belong to a global interface problem while the Lagrange multipliers belong to local subdomain problems. Moreover, an augmented Lagrangian formulation is used in conjunction with the localized Lagrange multipliers for robust multibody dynamics simulations.

The finite element tearing and interconnecting (FETI) method has been widely used for parallel finite element analysis. The FETI method is also a non-overlapping domain decomposition method and it uses Lagrange multipliers to enforce the kinematic constraints between subdomains. The interface problem generated by the FETI domain decomposition is solved by an iterative solver. The performance and convergence of iterative solvers, however, critically depend on the condition number of the system matrix. Thus, it is not desirable to use the FETI method for multibody problems because multibody systems are inherently ill-conditioned due to the presence of rigid body modes and kinematic constraints. The proposed domain decomposition method solves multibody dynamics problems using the direct solvers that have been developed and validated by the multibody community to overcome these problems.

Within the framework of direct solvers, the solution process can be divided into two parts: factorization of the stiffness matrix, followed by forward and backward substitutions. These two parts are further subdivided into five phases for parallel computations. In the first two phases, the subdomain stiffness matrices are factorized in parallel and an interface problem is factorized sequentially. In the last three phases, the linear equations for local subdomain problems are solved in parallel to obtain the intermediate subdomain solutions by forward substitutions and the linear equations for the global interface problem are solved sequentially for interface solutions by forward and backward substitutions. Finally, the solutions for the local subdomain problems are obtained by backward substitutions based on the interface solution. This parallel algorithm solution procedure requires data transfer between each subdomain and the interface.

Because the parallel algorithm with the proposed domain decomposition approach can be applied to standard finite element problems, the algorithm has been implemented in a finite-element-based nonlinear multibody dynamics simulation program (Dymore) to perform numerical experiments. During the domain decomposition process, an LLM element is defined for each localized Lagrange multiplier and is implemented as a finite element to enforce the kinematic constraints between interface and subdomain boundary nodes. After the domain decomposition, the original problem is split into subproblems, which are distributed to multiple processors for parallel computations. The interface problem is assigned to a master processor while subdomain problems are assigned to slave processors. The message passing interface (MPI) library has been used to implement inter-processor communication in the parallel solution procedure.

Several numerical experiments have been performed to assess the performance of the proposed domain decomposition method and to study the scalability of the parallel solution procedure. First, the static analysis of a cantilever beam was performed

to ascertain that both sequential and parallel algorithms yield identical solutions and convergence behavior. Next, dynamic simulations for a grid of beams, a typical structural dynamics problem, were performed to assess the performance of both domain decomposition and parallel implementation. Finally, a four-bladed helicopter rotor system, treated as a complex flexible multibody system, was considered. It was verified numerically that for both statics and dynamics simulations, the proposed domain decomposition approach yields solutions identical, within machine accuracy, to those obtained using serial processing and a single domain. This should be expected because the proposed domain decomposition method uses numerical techniques that have been fully validated for the solution of multibody problems.

To assess the performance of the proposed approach, three cases were contrasted. The SPSD (Single Processor for Single Domain) case is the baseline problem used as reference to assess performance. The SPMD (Single Processor for Multi-Domains) case introduces the proposed domain decomposition, which already improves performance significantly, without using parallel hardware. Finally, the MPMD (Multi-Processor for Multi-Domains) case implements the proposed approach in parallel hardware; this implementation is enabled by the proposed domain decomposition. The computational performance of these three cases has been contrasted.

From the SPMD case, it has been confirmed that the proposed domain decomposition method, by itself, can reduce CPU time dramatically by decreasing the bandwidth of the system matrices thereby reducing the computational cost of the factorization phase. The proposed domain decomposition approach lends itself naturally to a parallel implementation as demonstrated by the performance of the MPMD case. For the MPMD cases, the computational bottlenecks were expected to be the solution of the interface problem and the inter-processor communication time; the latter was observed to be the dominant bottleneck.

The numerical experiments presented here show poor scalability. This can be

attributed to the fact the number of degrees of freedom used for the modeling of multibody systems is rather modest and hence, it is not possible to use a large number of processors. Furthermore, as the number of subdomains increases, the size of the interface problem increases, creating a bottleneck in the solution process. Note that the FETI method uses iterative solvers to solve the interface problem, leading to better scalability. Finally, load balancing was not implemented in the proposed approach, leading to uneven subdomain sizes and computational requirements.

In summary, the proposed domain decomposition method enables parallel computations for flexible multibody dynamics simulations. The localized Lagrange multiplier technique leads to a robust solution procedure that relies on direct solvers only. The combined effects of domain decomposition and parallel processing shows great potential for better parallel performance.

6.2 Main Contributions of the Thesis

The main contributions of this thesis are:

- Developing a non-overlapping domain decomposition method which can deal with the differential algebraic equations nature of flexible multibody dynamics problems
- Employing localized version of Lagrange multipliers to enforce kinematic constraints between subdomains for better localization of subproblems and better convergence behavior
- Designing a parallel solution procedure which uses direct solvers only for both subdomain and interface problems
- Implementing parallel algorithms in a finite-element-based nonlinear multibody dynamics simulation program (Dymore) to perform numerical experiments on parallel hardware

6.3 Future Work

This work has introduced two techniques: domain decomposition and parallel processing. Future work should focus on further parallelization of all the phases of the solution procedure.

For instance, in Phase 1 of the solution procedure, element matrices and arrays are computed for each element. The computations of element matrices and arrays are usually done by numerical integration using Gauss integration, which implements the integration as a loop over the Gauss points. In the present implementation, this loop over the Gauss points is performed sequentially. Clearly, the same task can be implemented in parallel. Because this type of task is a fine-grained size, GPGPU can be used for parallel computations. If this parallelization is implemented with the proposed algorithm, the entire implementation falls into the category of a hybrid parallel approach that implements both fine- and coarse-grained parallelism simultaneously.

The factorization of the interface stiffness matrix is a bottleneck for the parallel implementation. To further speed up the factorization of the interface stiffness matrix, multifrontal solvers could be used to bring some level of parallelism into this phase of the solution process. Multifrontal solvers are widely available and have shown good scalability for large-scale problems. The use of multifrontal solvers should be explored for the solution of the interface problem.

As stated earlier in Section 6.1, optimal partitioning of the system is necessary for optimal load balancing. When dealing with multibody systems, load balancing is particularly difficult because the system cannot be partitioned along arbitrary boundaries. Indeed, the Lagrange multipliers used to enforce kinematic constraints and the constrained degrees of freedom must all belong to the same subdomain, imposing stringent requirements on partitioning of the system and hindering optimal load balancing.

REFERENCES

- [1] ANDERSON, K. S., “An order n formulation for the motion simulation of general multi-rigid-body tree systems,” *Computers and Structures*, vol. 46, no. 3, pp. 547–559, 1993.
- [2] ANDERSON, K. S. and DUAN, S., “A hybrid parallelizable low-order algorithm for dynamics of multi-rigid-body systems: Part I, chain systems,” *Mathematical and Computer Modelling*, vol. 30, no. 9-10, pp. 193–215, 1999.
- [3] BATHE, K. J., *Finite Element Procedures*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1996.
- [4] BAUCHAU, O. A. and AHMAD, J. U., “Advanced CFD and CSD methods for multidisciplinary applications in rotorcraft problems,” in *Proceedings of NASA/USAF Multidisciplinary Analysis and Optimization Symposium*, pp. 1441–1451, 1996.
- [5] BAUCHAU, O., “Parallel computation approaches for flexible multibody dynamics simulations,” *Journal of the Franklin Institute*, vol. 347, pp. 53–68, February 2010.
- [6] BAUCHAU, O., EPPLE, A., and BOTTASSO, C., “Scaling of constraints and augmented Lagrangian formulations in multibody dynamics simulations,” *Journal of Computational and Nonlinear Dynamics*, vol. 4, pp. 021007 1–9, April 2009.
- [7] BAUCHAU, O. A., *Dymore User Manual*. PhD thesis, Georgia Institute of Technology, Atlanta, USA.
- [8] BAUCHAU, O. A., “On the choice of appropriate bases for nonlinear dynamic modal analysis,” *Journal of the American Helicopter Society*, vol. 38, pp. 28–36, 1993.
- [9] BAUCHAU, O. A., “Computational Schemes for Flexible , Nonlinear Multi-Body Systems,” *Multibody System Dynamics*, vol. 2, no. 2, pp. 169–225, 1998.
- [10] BAUCHAU, O. A. and HODGES, D. H., “Analysis of Nonlinear Multibody Systems with Elastic Couplings,” *Multibody System Dynamics*, vol. 3, pp. 163–188, 1999.
- [11] BAUCHAU, O. A. and LAULUSA, A., “Review of contemporary approaches for constraint enforcement in multibody systems,” *Journal of Computational and Nonlinear Dynamics*, vol. 3, no. 1, pp. 011005 1–8, 2008.

- [12] BAVESTRELLO, H., AVERY, P., and FARHAT, C., “Incorporation of linear multipoint constraints in domain-decomposition-based iterative solvers - Part II: Blending FETI-DP and mortar methods and assembling floating substructures,” *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 8, pp. 1347–1368, 2007.
- [13] BAYO, E., DE JALÓN, J. G., AVELLO, A., and CUADRADO, J., “An efficient computational method for real time multibody dynamic simulation in fully cartesian coordinates,” *Computer Methods in Applied Mechanics and Engineering*, vol. 92, pp. 377–395, 1991.
- [14] BAYO, E., DE JALÓN, J. G., and SERNA, M., “A modified lagrangian formulation for the dynamic analysis of constrained mechanical systems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 71, pp. 183–195, November 1988.
- [15] BOTTASSO, C., BAUCHAU, O., and CARDONA, A., “Time-step-size-independent conditioning and sensitivity to perturbations in the numerical solution of index three differential algebraic equations,” *SIAM Journal on Scientific Computing*, vol. 29, no. 1, pp. 397–414, 2007.
- [16] CESNIK, C. and HODGES, D., “VABS: a new concept for composite rotor blade cross-sectional modeling,” *Journal of the American Helicopter Society*, vol. 42, pp. 27–38, 1997.
- [17] CHIOU, J., PARK, K., and FARHAT, C., “A natural partitioning scheme for parallel simulation of multibody systems,” *International Journal for Numerical Methods in Engineering*, vol. 36, pp. 945–967, 1993.
- [18] COULON, D., *Paradigmes Parallèles Appliqués au Calcul de la Dynamique non Linéaire des Mécanismes Flexibles*. PhD thesis, Université de Liège, Belgium, 1998.
- [19] COULON, D., GÉRADIN, M., and FARHAT, C., “Adaptation of a finite element solver for the analysis of flexible mechanisms to parallel processing systems,” *Second International Conference on Computational Structures Technology*, Athens, Greece, 30 August - 1 September 1994.
- [20] CRITCHLEY, J. H. and ANDERSON, K. S., “A parallel logarithmic order algorithm for general multibody system dynamics,” *Multibody System Dynamics*, vol. 12, no. 1, pp. 75–93, 2004.
- [21] DUFF, I. and REID, J., “The multifrontal solution of indefinite sparse symmetric linear equations,” *ACM Transactions on Mathematical Software*, vol. 9, pp. 302–325, 1973.
- [22] EPPLE, A., *Methods for increased computational efficiency of multibody simulations*. PhD thesis, Georgia Institute of Technology, 2008.

- [23] FARHAT, C., CHEN, P.-S., RISLER, F., and F.-X., R., “A unified framework for accelerating the convergence of iterative substructuring methods with Lagrange multipliers,” *International Journal for Numerical Methods in Engineering*, vol. 42, no. January 1997, pp. 257–288, 1998.
- [24] FARHAT, C., CRIVELLI, L., and ROUX, F.-X., “Extending substructure based iterative solvers to multiple load and repeated analyses,” *Computer Methods and Applied Mechanics and Engineering*, vol. 117, pp. 195–209, 1994.
- [25] FARHAT, C., CRIVELLI, L., and ROUX, F.-X., “A transient FETI methodology for large-scale implicit computations in structural mechanics,” *International Journal for Numerical Methods in Engineering*, vol. 37, no. 11, pp. 1945–1975, 1994.
- [26] FARHAT, C. and GERADIN, M., “On the general solution by a direct method of a large-scale singular system of linear equations: Application to the analysis of floating structures,” *International Journal for Numerical Methods in Engineering*, vol. 41, no. 4, pp. 675–696, 1998.
- [27] FARHAT, C., LESOINNE, M., and PIERSON, K., “A scalable dual-primal domain decomposition method,” *Numerical Linear Algebra with Applications*, vol. 7, pp. 687–714, 2000.
- [28] FARHAT, C., PIERSON, K., and LESOINNE, M., “The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems,” *Computer Methods and Applied Mechanics and Engineering*, vol. 184, pp. 333–374, 2000.
- [29] FARHAT, C. and ROUX, F.-X., “A method of finite element tearing and interconnecting and its parallel solution algorithm,” *International Journal for Numerical Methods in Engineering*, vol. 32, pp. 1205–1227, 1991.
- [30] FARHAT, C. and ROUX, F.-X., “An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems,” *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 1, pp. 379–396, 1992.
- [31] FARHAT, C., CHEN, P.-S., MANDEL, J., and ROUX, F.-X., “The two-level FETI method for static and dynamic plate problems Part I: An optimal iterative solver for biharmonic systems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 155, no. 1-2, pp. 129–151, 1998.
- [32] FARHAT, C., CHEN, P.-S., MANDEL, J., and ROUX, F.-X., “The two-level FETI method Part II: Extension to shell problems, parallel implementation and performance results,” *Computer Methods in Applied Mechanics and Engineering*, vol. 155, no. 1-2, pp. 153–179, 1998.

- [33] FARHAT, C., LESOINNE, M., LETALLEC, P., PIERSON, K., and RIXEN, D., “FETI-DP: A dual-primal unified FETI method part I: A faster alternative to the two-level FETI method,” *International Journal for Numerical Methods in Engineering*, vol. 50, no. 7, pp. 1523–1544, 2001.
- [34] FEATHERSTONE, R., “A Divide-and-Conquer Articulated-Body Algorithm for Parallel $O(\log(n))$ Calculation of Rigid-Body Dynamics. Part 1: Basic Algorithm,” *The International Journal of Robotics Research*, vol. 18, no. 9, pp. 876–892, 1999.
- [35] FEATHERSTONE, R., “A Divide-and Conquer Articulated-Body Algorithm for Parallel $O(\log(n))$ Calculation of Rigid-Body Dynamics. Part 2: Part 2: Trees, Loops, and Accuracy,” *International Journal of Robotics Research*, vol. 18, no. 9, pp. 876–892, 1999.
- [36] FELIPPA, C. A., “Superelements and global-local analysis,” *Class notes*, 2016.
- [37] FIJANY, A., SHARF, I., and DELEUTERIO, G. M. T., “Parallel $O(\log N)$ Algorithms for Computation of Manipulator Forward Dynamics,” *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 389–400, 1995.
- [38] FORTIN, M. and GLOWINSKI, R., *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*. Amsterdam, the Netherlands: North-Holland, 1983.
- [39] GÉRADIN, M. and CARDONA, A., *Flexible Multibody System: A Finite Element Approach*. New York: John Wiley & Sons, 2001.
- [40] GILL, P., MURRAY, W., SAUNDERS, M., and WRIGHT, M., “Sequential quadratic programming methods for nonlinear programming,” *Computer-Aided Analysis and Optimization of Mechanical System Dynamics*, vol. 9, pp. 679–700, 1984.
- [41] GOLUB, G. H. and VAN LOAN, C. F., *Matrix Computations*. Baltimore and London: The Johns Hopkins University Press, third ed., 1996.
- [42] GOSSELET, P. and REY, C., “Non-overlapping domain decomposition methods in structural mechanics,” *Archives of computational methods in engineering*, vol. 13, pp. 515–572, 2006.
- [43] GUNNEY, M. E., *High-performance direct solution of finite element problems on multi-core processors*. PhD thesis, Georgia Institute of Technology, 2010.
- [44] HODGES, D. H., *Nonlinear Composite Beam Theory*. Reston, Virginia: AIAA, 2006.
- [45] IRONS, B. M., “A frontal solution scheme for finite element analysis,” *International Journal for Numerical Methods in Engineering*, vol. 2, pp. 5–32, 1970.

- [46] KIM, J. H. and KIM, S. J., “Multifrontal Solver Combined with Graph Partitioner,” *AIAA Journal*, vol. 37, no. 8, pp. 964–970, 1999.
- [47] KIM, K.-C. and CHOPRA, I., “Aeroelastic analysis of helicopter rotor blades with advanced tip shapes,” in *31st Structures, Structural Dynamics and Materials Conference*, 1990.
- [48] KORNEEV, V. G., LANGER, U., and KEPLER, J., “Domain Decomposition Methods and Preconditioning,” *Encyclopedia of Computational Mechanics*, vol. 1, no. 22, 2004.
- [49] LAULUSA, A. and BAUCHAU, O. A., “Review of Classical Approaches for Constraint Enforcement in Multibody Systems,” *Journal of Computational and Nonlinear Dynamics*, vol. 3, no. 1, pp. 011004 1–8, 2008.
- [50] LESOINNE, M. and PIERSON, K., “FETI-DP: An Efficient, Scalable, and Unified Dual-Primal FETI Method,” in *International conference on Domain Decomposition Methods*, pp. 421–428, 1999.
- [51] MESSAGE PASSING INTERFACE FORUM, “MPI: A message-passing interface standard,” tech. rep., University of Tennessee, Knoxville, 2012.
- [52] MUMPS, “MULTifrontal Massively Parallel Solver: Users’ guide,” 2017.
- [53] MURTY, H., BOTTASSO, C., DINDAR, M., SHEPHARD, M., and BAUCHAU, O., “Aeroelastic analysis of rotor blades using nonlinear fluid/structure coupling,” in *Proceedings of American Helicopter Society 53rd Annual Forum Proceedings*.
- [54] NEGRUT, D., TASORA, A., MAZHAR, H., HEYN, T., and HAHN, P., “Leveraging parallel computing in multibody dynamics,” *Multibody System Dynamics*, vol. 27, no. 1, pp. 95–117, 2012.
- [55] PARK, K. C. and FELIPPA, C. A., “A Variational Framework for Solution Method Developments in Structural Mechanics,” *Journal of Applied Mechanics*, vol. 65, no. March, pp. 242–249, 1998.
- [56] PARK, K., FELIPPA, C., and GUMASTE, U., “A localized version of the method of Lagrange multipliers and its applications,” *Computational Mechanics*, vol. 24, pp. 476–490, 2000.
- [57] PRZEMIENIECKI, J. S., “Matrix Structural Analysis of Substructures,” *AIAA Journal*, vol. 1, no. 1, pp. 138–147, 1963.
- [58] QUARANTA, G., MASARATI, P., and MANTEGAZZA, P., “Multibody analysis of controlled aeroelastic systems on parallel computers,” *Multibody System Dynamics*, vol. 8, no. 1, pp. 71–102, 2002.
- [59] ROUX, F.-X., “Tests on parallel machines of a domain decomposition method for a composite three-dimensional structural analysis problem,” *Association for Computing Machinery*, pp. 273–283, 1988.

- [60] ROUX, F.-X., “Acceleration of the outer conjugate gradient by reorthogonalization for a domain decomposition method for structural analysis problems,” *Association for Computing Machinery*, pp. 471–476, 1989.
- [61] SAAD, Y., *Iterative Methods for Sparse Linear Systems*. Philadelphia: Society for Industrial and Applied Mathematics, second ed., 2003.
- [62] SAAD, Y. and SCHULTZ, M. H., “GMRES: a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems,” *SIAM J. Sci. STAT. COMPUT. Vol.—*, vol. 7, no. 3, pp. 856–869, 1986.
- [63] SCHWARZ, H. A., “Über einige Abbildungsaufgaben,” *J. Reine Angew. Math.*, vol. 70, pp. 105–120, 1869.
- [64] SIMO, J. C. and FOX, D. D., “On a stress resultant geometrically exact shell model. Part I formulation and optimal parametrization,” *Computer Methods in Applied Mechanics and Engineering*, vol. 72, pp. 267–304, 1989.
- [65] SIMO, J. C., FOX, D. D., and RIFAI, M. S., “On a stress resultant geometrically exact shell model. Part II: The linear theory; Computational aspects,” *Computer Methods in Applied Mechanics and Engineering*, vol. 73, no. 1, pp. 53–92, 1989.
- [66] TONG, P. and PIAN, T., “A hybrid-element approach to crack problems in plane elasticity,” *International Journal for Numerical Methods in Engineering*, no. 7, pp. 297–308, 1973.
- [67] WAKE, B. and SANKAR, L., “Solution of Navier-Stokes equations for the flow over a rotor blade,” *Journal of American Helicopter Society*, 1989.
- [68] YANG, Z., SANKAR, L., SMITH, M., and BAUCHAU, O., “Recent improvements to a hybrid method for rotors in forward flight,” *Journal of Aircraft*, vol. 39, no. 2, pp. 804–812, 2002.

VITA

SeunDo Heo was born on February 3, 1981 in Masan, South Korea. He obtained his Bachelors degree in Aerospace and Mechanical Engineering from Korea Aerospace University in February 2006. In August 2006, he started his graduate studies in the School of Aerospace Engineering at Georgia Institute of Technology, Atlanta, GA. He received his Masters degree in August 2008 completing a Special Problem. SeunDo Heo passed the qualifying examinations in April 2010 and became a PhD candidate after his thesis proposal in December 2013. As a main part of his thesis, he developed, implemented, tested and evaluated domain-decomposition-based parallel algorithms for flexible multibody dynamics on parallel hardware.