A LOAD-BALANCING TOOL FOR STRUCTURED MULTI-BLOCK CFD APPLICATIONS APPLIED TO A PARALLEL NEWTON-KRYLOV ALGORITHM

by

Kwesi Parry Apponsah

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

A LOAD-BALANCING TOOL FOR STRUCTURED MULTI-BLOCK CFD APPLICATIONS APPLIED TO A PARALLEL NEWTON-KRYLOV ALGORITHM

Kwesi Parry Apponsah

Master of Applied Science

Graduate Department of Aerospace Science and Engineering

University of Toronto

2012

Advances in parallel computers have made the simulation of large scale computational fluid dynamics (CFD) problems feasible. For high-fidelity parallel CFD simulations, multi-block grid methodology makes it possible to simulate flows around complex geometries. An automatic load-balancing tool is developed for a parallel Newton-Krylov algorithm that uses multi-block grids. The load-balancing tool uses a recursive edge bisection tool for splitting blocks in order to enforce load-balancing constraints. The tool can be started off with a smaller number of blocks than processors. When homogeneous multi-block grids are used, an optional constraint is introduced to control block splitting. In the case of heterogeneous multi-block grids, a block size constraint is introduced to prevent smaller blocks from being split when the tool is started of with a smaller number of blocks than processors. The load-balancing tool is applied to three-dimensional multi-block grids for a Newton-Krylov solution process applied to the Euler and Reynolds-Averaged Navier-Stokes equations. For heterogeneous grids, significant turnaround time reductions are obtained using the load-balancing tool with a smaller number of processors than without a load-balancing tool. In the case of a homogeneous grid, the tool can be used to run a simulation on a larger number of processors than the problem was initially set up for. Finally, using the automatic tool, the scaling properties of the parallel Newton-Krylov algorithm are investigated.

Acknowledgements

I would like to thank Dr. David Zingg for being an excellent teacher, supervisor and a mentor. I am very grateful for giving me such a wonderful opportunity to work with you. I would also like to thank my Research Assessment Committee (RAC) for their insights and questions during my research assessment meetings.

To my colleagues at UTIAS, especially the CFD group, thank you for all your support and assistance during my studies. To Joan, Gail and Dr. Gottlieb, thank you for making my stay in UTIAS a pleasant one.

I am also grateful to my parents and siblings for all their support and encouragement. To Lala, thank you for all your patience and understanding.

I also wish to acknowledge financial support from the MITACS, Canada Research Chairs Program and the University of Toronto towards my studies.

KWESI PARRY APPONSAH

University of Toronto Institute for Aerospace Studies

September 18, 2012

Contents

List of Tables **vii**

List of Figures **ix**

List of Symbols and Abbreviations **xi**

1 Introduction **1**

1.1 CFD, and the Future of Aircraft Design 1

1.2 Structured Multi-block CFD Applications 3

1.2.1 Parallel Considerations for Multi-block CFD Applications 5

1.2.2 A Case for Load-Balancing 6

1.2.3 Load Balancing for CFD Applications 7

1.3 Thesis Objectives 8

1.4 Thesis Outline 9

2 Newton-Krylov Framework **11**

2.1 Governing Equations 11

2.2 Coordinate Transformation 12

2.3 Spatial Discretization 13

2.3.1 Inviscid Fluxes 14

2.3.2 Artificial Dissipation 14

2.3.3 Block Interfaces and Boundary Conditions 14

2.4 Newton-Krylov Flow Solver 15
2.4.1 Preconditioning 16

3 Load Balancing Strategy **21**
3.1 Coarse-grain Parallelism Approach 21
3.2 Load Balancing Algorithm 23
3.3 Application to a 2D Structured Multi-block Grid 27
3.3.1 10 Processors 28
3.3.2 12 Processors 30
3.3.3 15 Processors 31
3.4 Remarks on Results 33

4 Scaling Studies for the Newton-Krylov Algorithm **35**
4.1 Parallel Scaling 36
4.2 Homogeneous Multi-block Grids 37
4.2.1 Case I : $n_b^{(o)} > n_p^{(o)}$ 41
4.2.2 Case II : $n_b^{(o)} < n_p^{(o)}$ 47
4.3 Heterogeneous Multi-block Grids 48
4.3.1 ONERA M6 Wing 48
4.3.2 CRM Wing-body Configuration 55

5 Conclusions and Recommendations **63**
5.1 Conclusions 63
5.2 Recommendations 65

References **67**

List of Tables

3.1 Values of c and $f_{t,e}$ obtained for the cases considered 28

4.1 Mesh properties and flow conditions for the homogeneous case 38
4.2 Percentage reductions obtained after the introduction of b_r=2.0 relative to f_t=1.10 in Figures 4.6 and 4.7 46
4.3 Mesh properties and flow conditions for the heterogeneous cases 48
4.4 Speedup relative to 128 processors without load balancing 56
4.5 Speedup relative to 569 processors without load balancing 61

List of Figures

1.1 A 13-block grid for a multi-element airfoil 3

3.1 Initial C-mesh for RAE 2822 airfoil 28
3.2 Workload distribution for 10 processors 29
3.3 Domain decomposition for 10 processors 30
3.4 Workload distribution for 12 processors 31
3.5 Domain decomposition for 12 processors 32
3.6 Workload distribution for 15 processors 32
3.7 Domain decomposition for 15 processors 33

4.1 Coefficient of pressure distribution on the ONERA M6 wing using 96 processors with $f_t = 1.10$ 39
4.2 Execution times obtained for 16, 30, 48, 72, 128, 180, 240, 250, 320, 460, 500, 580, 640, 720, 760, 840, 960 and 1024 processors. 39
4.3 Relative efficiencies, η_p 40
4.4 A comparison of the Schur and Schwarz preconditioning techniques for the homogeneous multi-block grid for the ONERA M6 wing 42
4.5 Execution times obtained comparing the Schur and Schwarz preconditioners with varying number of blocks on 16 processors 43
4.6 Execution times and mesh-size factor c obtained for $n_b^{(o)} > n_p^{(o)}$ 44
4.7 Preconditioning time and Krylov iterations obtained for $n_b^{(o)} > n_p^{(o)}$ . . . 46
4.8 Execution times obtained for 180, 240, 320, 460, 580, 640, 760, 840 and 960 processors when $n_b^{(o)} < n_p^{(o)}$ with varying f_t. 47
4.9 Coefficient of pressure distribution on the Common Research Model wing-geometry using 560 processors with $f_t = 1.40$ 49
4.10 Parallel performance properties for the heterogeneous multi-block grid for the ONERA M6 wing 50
4.11 Newton-Krylov algorithm performance for the heterogeneous multi-block grid for the ONERA M6 wing 52
4.12 A comparison of the Schur and Schwarz preconditioning techniques for the heterogeneous multi-block grid for the ONERA M6 wing 53
4.13 C_L and C_D values for the heterogeneous multi-block grid for the ONERA M6 wing 54
4.14 Parallel performance properties for the heterogeneous multi-block grid for the CRM wing-body geometry 57

4.15 Newton-Krylov algorithm performance for the heterogeneous multi-block grid for the CRM wing-body geometry 59
4.16 C_L and C_D values for the heterogeneous multi-block grid for the CRM wing-body geometry 60

List of Symbols and Abbreviations

Alphanumeric

$\hat{\mathbf{E}}$, $\hat{\mathbf{F}}$, $\hat{\mathbf{G}}$	Discrete inviscid convective fluxes in the computational domain
b_r	Blocks-to-processor ratio
c	Mesh size factor
e	Total energy
E, F, G	Inviscid convective fluxes in physical space
f_b	Block size constraint
$f_{t,e}$	Effective load balance threshold
f_t	Load balance threshold
J	Metric Jacobian
n_b	Number of blocks in a multi-block grid
n_p	Number of processors
p	Pressure
Q	Flow variables in physical space
U, V, W	Contravariant velocities

Greek

γ	Ratio of specific heats
ρ	Density
ξ, η, ζ	Coordinates in computational space

Abbreviations

CRM	Common Research Model

DDR	Double Data Rate
DPW	Drag Prediction Workshop
ILU	Incomplete Lower-Upper
JAXA	Japanese Aerospace Exploration Agency
NK	Newton-Krylov
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
QDR	Quad Data Rate
RAM	Random Access Memory
RANS	Reynolds-Averaged Navier-Stokes
REB	Recursive Edge Bisection
SAT	Simultaneous Approximation Terms
SBP	Summation by Parts

Chapter 1

Introduction

1.1 CFD, and the Future of Aircraft Design

Computational fluid dynamics (CFD) has grown from a purely scientific discipline to become an important tool for solving fluid dynamics problems in industry, especially in the aerospace sector. The development of robust algorithms and the rapid advancements in computing facilities, particularly high performance computers (HPCs), have been responsible for the role CFD currently plays in the design and analysis of systems which interact with fluids.

Prior to the 1990s, full potential codes and panel methods were predominantly used in industry for solving flow problems during design stages. This was because these classical methods were computationally less expensive and more robust. The ability of these methods to employ a large number of grid points or panels also made them more accurate than the Euler and Navier-Stokes codes. The Euler and Navier-Stokes codes required more computational work per solution and storage requirements. Therefore, the development of Euler and Navier-Stokes codes was considered purely at the research level because of limited computing capabilities.

The advances in computing capabilities in the last two decades have contributed

significantly to the maturity of CFD technology as a design tool. As of now, many super-computers can perform 10^{12} *floating-operations per second* (giga-FLOPS), with some of the fastest in the peta-FLOPS regime. The pace of algorithm development for the Euler and Navier-Stokes equations has also caught up with the advances in computing and in effect many large-scale problems which would have taken years to solve three decades ago are being solved in minutes to hours. CFD codes such as ARC2D/ARC3D [46, 45], CFL3D and OVERFLOW, all of which are contributions from National Aeronautics and Space Administration (NASA), have been the basis for developing and comparing CFD codes for external flows.

As the computing power of HPCs continue to improve, high-fidelity CFD algorithms that can effectively solve turbulent flows, accurately predict flow transition, and also help design and optimize geometries will increasingly become the focus of CFD engineers. The efficient implementation of these algorithms in a parallel computing environment will be critical if we are to derive full benefit from the high-end computing platforms that will become available. In view of this, CFD and computational science engineers will consistently be faced with the challenge to develop efficient numerical methods for parallel computers, and ideally, these algorithms will have to perform as well as their serial counterparts.

CFD will also play a bigger role during design cycles of future aircraft because of the increasing demand on manufacturers to develop and build fuel-efficient, low-drag and quieter vehicles. The rising cost of fuel and concerns regarding the effect of green-house gases on the environment have been the main reasons for the shift towards efficient aircraft designs. The expectation is that CFD will provide an inexpensive and efficient approach to assess the performance of different designs simultaneously. In effect, high-fidelity CFD codes can also be used in a multi-disciplinary optimization framework, which has the potential to improve performance qualities such as range and lift-drag ratio, and could lead to the discovery of unconventional aircraft configurations as well.

1.2 Structured Multi-block CFD Applications

CFD algorithms that use finite differences to approximate the partial differential equations (PDEs) that describe fluid flow require a structured discretization of the problem domain in order to obtain numerical solutions to these equations. Structured grids employ curvilinear coordinates to produce body-fitted meshes. This grid methodology has the advantage that boundaries can be exactly described and hence boundary conditions can be accurately modelled. As the complexity of the geometry increases, single-block structured grids become insufficient for most practical problems. To overcome the shortcomings of single-block structured grids, structured multi-block grids are used. Figure 1.1 shows are a 13-block grid for a multi-element airfoil.

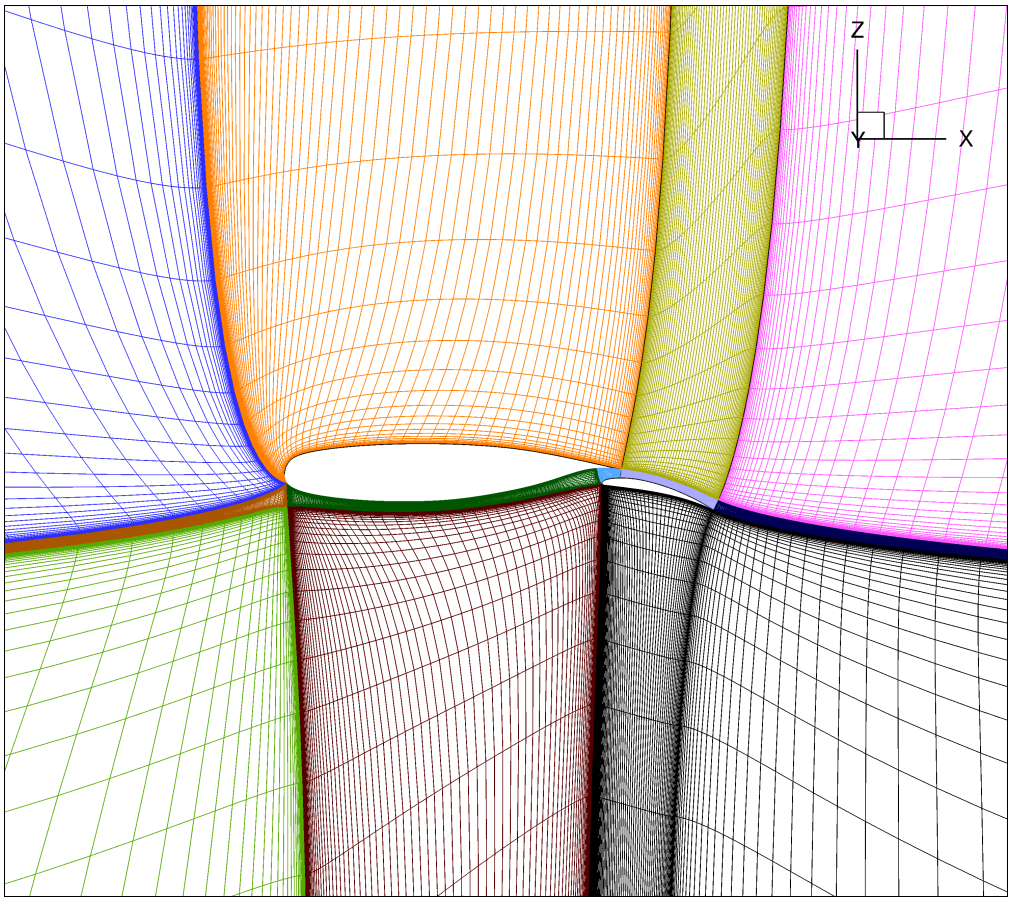


Figure 1.1: A 13-block grid for a multi-element airfoil

In the multi-block approach, the solution domain is divided into blocks or subdomains. At the block level, multi-block grids can be heterogeneous, i.e. composed of blocks with different sizes, or homogeneous i.e. blocks with equal sizes. A structured grid is generated for each block accordingly. In addition, the connectivity information for all six faces of individual blocks needs to be determined. As an additional requirement on multi-block grids suited for finite-differences, the two faces of blocks sharing an interface should typically have the same number of tangential nodes. Depending on how the solution algorithm treats block interfaces, grid lines may be required to be smooth at the interfaces. Used as a domain decomposition strategy, the multi-block methodology allows for direct parallelization of both grid generation and flow codes on massively parallel computers [20].

In order to compute the flow solution for multi-block grids, each block in the physical domain is mapped into a computational domain where the mesh is uniform. The governing equations are therefore solved on a set of connected blocks in the computational domain. Since each block is assigned its own local coordinate system, spatial discretization of each block can be carried out independently.

The main difficulty with the use of multi-block finite-difference schemes is to couple blocks together in a stable and accurate manner. To resolve this problem Summation-by-Parts (SBP) operators can be used to discretize each block and treat both the boundary (external) and interface (internal) conditions using the Simultaneous Approximation Terms (SAT) boundary procedure. This particular approach was adopted in [7] and [8] and to construct stable and conservative boundary and interface conditions for the (1D) constant-coefficient Euler and Navier-Stokes equations on multiple domains. In [23], Hicken and Zingg extended this approach to multi-dimensional (3D) multi-block grids using the Euler equations. Osusky and Zingg [42] have also applied the SBP-SAT approach to three-dimensional multi-block grids using the Reynolds-Averaged Navier-Stokes (RANS) equations. Chapter 2 provides some details of the use of SBP operators

and SATs for the CFD solver used for all the test cases considered in this thesis.

1.2.1 Parallel Considerations for Multi-block CFD Applications

Multi-block grids allow for direct parallelization of CFD solution codes. The common approach to parallelizing multi-block CFD codes is through domain decomposition or grid partitioning. Blocks are required to be equal or greater than the number of processors requested for each computation. In this approach, one or more blocks are combined to form a partition, each processor is then assigned a partition. Therefore, the same set of computer instructions for the solution process is executed on each partition independently. For example to compute a global inner product, the local inner product for each process is calculated first and then the components local to each process are summed. The Message Passing Interface (MPI) API function *MPI_Allreduce()* can be used for the summation process.

Generally, any form of domain decomposition introduces a new type of boundary condition. This new boundary type exists at domain interfaces. For parallel computations adjacent blocks may reside on different processors, therefore a general strategy is to construct halo or ghost blocks that surround the block boundaries. The halo block attached to a face of a block has the full tangential indices of the face and one or multiple layers depending on the number of nodes required to enforce the inter-domain boundary condition. Message passing between processors is used to update the halo block data required to enforce this new boundary condition. The use of message passing introduces an overhead during data exchange. Non-blocking message passing "*send*" and "*receive*" can be used to partially hide the overhead incurred during communication since instructions that do not require the data being exchanged can be executed as well while communication between processors proceeds.

For simple geometries, homogeneous grids can easily be generated for CFD computations. Hence obtaining a perfect static load balance, and the choice of the number of

processors become a natural consequence of grid generation. As the complexity of the geometry increases, it becomes unrealistic to assume that all blocks will have the same size throughout the physical domain. Therefore, load balancing becomes a major concern under such circumstances. Distributing workload "evenly" across processors based on some algorithm helps improve turnaround times for computations involving heterogeneous or arbitrary multi-block grids.

1.2.2 A Case for Load-Balancing

Assigning blocks with different sizes to processors can cause serious load balancing problems. One way to solve this problem is to restrict multi-block grids to the same block sizes during mesh generation. Such an approach is not realistic for all applications since blocks should be generated on the basis of the geometry and refined according to numerical and aerodynamic requirements. Therefore, an automatic tool that splits blocks and distributes workload as evenly as possible can enable more freedom in grid generation and the number of processors to be used. The inherent coarse-grain parallelism that multi-block grids exhibit can be exploited to achieve this goal.

Considering the distribution of blocks to processors, there is a maximum achievable parallel efficiency, since the flow solution process proceeds at the pace of the slowest processor, i.e. the one with the maximum number of grid points per processor [20]. This peak efficiency, which is a ratio of the maximum number to the average number of grid points per processor, can be improved significantly with an automatic load balancing tool, since some of the workload on overloaded processors can be passed on to processors with smaller amounts of work.

An automatic tool for load balancing also improves the portability of a CFD application, since a CFD code with such an implementation can run on any kind of distributed or shared memory system.

1.2.3 Load Balancing for CFD Applications

Many load-balancing algorithms, such as METIS [28] and ParMETIS [29], have been developed for unstructured grids. Prior to the mid 1990s, work on structured multi-block meshes focused on distributing original blocks without necessarily splitting blocks. In 1997, Ytterström [57] used the Recursive Edge Bisection (REB) algorithm to load-balance multi-block grids. REB was used to increase the number of initial blocks in order to run on a larger number of processors. Ytterström [57] also proposed the Greedy and GreedyXtra load-balancing techniques. These two algorithms are better suited for heterogeneous machines since smaller problems are used to test the execution speed of each processor before load-balancing proceeds. The processor information gained is then factored into the load-balancing algorithms. In the case of the GreedyXtra technique, workload in ghost cells and boundary nodes is also accounted for.

Similarly, work by Rantakokko [49] and Sermeus *et al.* [26] also proposed various load-balancing techniques for structured multi-block meshes. In [49], a load-balancing tool that exploits both fine-grain and coarse-grain parallelisms exhibited by multi-block grids is proposed. The technique can assign subdomains to a processor or assume subdomains to be single grids and then use single-grid partitioning techniques to partition a subdomain across all processors. Clustering processors into a group in the case of a large number of processors compared to blocks is also proposed to ensure a good load balance as well.

Sermeus *et al.* proposed two algorithms. The first algorithm distributes workload without splitting blocks. The second algorithm is an improved version of the first algorithm. The algorithm is iterative and enforces a load balance constraint through splitting blocks. In this algorithm, the largest block is split at each iteration until the load balance constraint is met. They showed that the improved algorithm gave a better workload distribution on processors compared to the first algorithm. In their iterative procedure, the largest block at the n^{th} iteration is split using REB.

The underlying goal of all the above algorithms is to distribute workload as evenly as possible. Usually, load-balancing algorithms are developed to suit the needs of the CFD algorithms used. Partitioning meshes through the use of a block splitting tool is usually common to all. The difference in algorithms come from how constraints are defined and the assumptions made in order to determine how partitioning is done.

The work presented in this thesis modifies the second algorithm proposed by Sermeus *et al.* specifically for the parallel CFD algorithm used. Modifications are made to improve the number of load-balancing iterations performed, while new constraints are implemented as well.

Using such an automatic load-balancing tool, weak and strong scaling studies can be performed easily whenever modifications are made to the parallel CFD algorithm. In weak scaling, the size of the problem and the number of processors are varied but the workload per processor is kept constant. For strong scaling, the grid size remains the same while the number of processors is varied. These performance parameters can give an insight into parts of the algorithm that can be improved to speed up the overall solution process.

1.3 Thesis Objectives

The objectives of this thesis are:

- To develop an automatic tool for splitting blocks and also partitioning multi-block grids as evenly as possible across processors, in order to derive maximum benefit from a parallel computing platform.

- To integrate the automatic load balancing tool into a Newton-Krylov (NK) CFD framework developed by Hicken, Osusky and Zingg [21, 23, 41, 42] for the Euler and RANS equations to enable the NK framework to handle any arbitrary multi-block grid for 3D flow problems. This objective will also make the NK framework

portable to any parallel computing platform.

- To investigate the scalability of the Newton-Krylov CFD code on homogeneous and heterogeneous multi-block grids using the load balancing tool. This investigation will also include a comparison of the performance of the additive-Schwarz and approximate-Schur parallel preconditioners depending on the number of blocks assigned per process during the solution.

1.4 Thesis Outline

The thesis is divided into 5 chapters. Following this introductory chapter, Chapter 2 presents the Newton-Krylov CFD solution process. The load balancing algorithm implementation and results for tests performed on a two-dimensional multi-block grid are presented in Chapter 3. Chapter 4 presents and discusses the parallel scaling results obtained for both homogeneous and heterogeneous multi-block grids in three dimensions. A comparison of the additive-Schwarz and approximate-Schur parallel preconditioners is also presented in Chapter 4. Finally, based on the results presented in Chapter 4, concluding remarks and recommendations for further research are presented in Chapter 5.

Chapter 2

Newton-Krylov Framework

A Newton-Krylov strategy is used to solve the the Euler and RANS equations. The flow solver is largely based on work in [11, 23, 38, 39, 42, 44, 46]. A simple presentation of the Newton-Krylov framework as applied to the Euler equations follows.

2.1 Governing Equations

The governing equations considered in this chapter are the Euler equations. The Euler equations neglect fluid viscosity and as such retain only the convection fluxes of the Navier-Stokes equations. The Euler equations are expressed in conservative variables as:

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} + \frac{\partial G}{\partial z} = 0 \tag{2.1}$$

where Q is the vector of conservative variables, and E, F and G are the inviscid convective fluxes:

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{bmatrix}, \quad E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(e+p) \end{bmatrix}, \quad F = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(e+p) \end{bmatrix}, \quad \text{and} \quad G = \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(e+p) \end{bmatrix} \tag{2.2}$$

Here, ρ is the density of the fluid, u, v and w are the components of the fluid velocity vector, and e is the total energy of the fluid. The vectors Q, E, F and G are functions of the spatial coordinates x, y and z. To close the Euler equations the fluid is assumed to be a thermally and calorically perfect gas. Therefore, the pressure, p and the flow variables appearing in the conservative variable vector, Q are related as follows:

$$p = (\gamma - 1)\left[e - \frac{\rho(u^2 + v^2 + w^2)}{2}\right] \tag{2.3}$$

where the ratio of specific heats γ is 1.4 for a diatomic gas such as air.

2.2 Coordinate Transformation

A generalized coordinate transformation is used to map the physical domain with coordinates x, y and z into a computational domain with coordinates ξ, η, and ζ based on Pulliam [46] and Pulliam and Steger [47]:

$$\begin{aligned} \xi &= \xi(x, y, z) \\ \eta &= \eta(x, y, z) \\ \zeta &= \zeta(x, y, z) \end{aligned} \tag{2.4}$$

In the computational domain, the grid is uniform and the value of the grid spacing between nodes is unity. The use of the coordinate transformation simplifies the spatial discretization significantly.

In the computational space, the Euler equations can be written as:

$$\frac{\partial \hat{Q}}{\partial t} + \frac{\partial \hat{E}}{\partial \xi} + \frac{\partial \hat{F}}{\partial \eta} + \frac{\partial \hat{G}}{\partial \zeta} = 0 \tag{2.5}$$

where

$$\hat{Q} = J^{-1}Q, \quad \hat{E} = J^{-1}\begin{bmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ \rho w U + \xi_z p \\ (e+p)U \end{bmatrix}, \quad \hat{F} = J^{-1}\begin{bmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ \rho w V + \eta_z p \\ (e+p)V \end{bmatrix}, \quad \text{and } \hat{G} = J^{-1}\begin{bmatrix} \rho W \\ \rho u W + \zeta_x p \\ \rho v W + \zeta_y p \\ \rho w W + \zeta_z p \\ (e+p)W \end{bmatrix} \tag{2.6}$$

where U, V and W are contravariant velocities, defined as:

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \xi_x & \eta_x & \zeta_x \\ \xi_y & \eta_y & \zeta_y \\ \xi_z & \eta_z & \zeta_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.7}$$

Details on how to compute the partial derivative terms in (2.7) can be found in Pulliam and Zingg [48].

2.3 Spatial Discretization

The Euler equations (2.5) can be cast in the following form:

$$\frac{1}{J}\frac{\partial Q}{\partial t} + R(Q) = 0 \tag{2.8}$$

where

$$R(Q) = \frac{\partial \hat{E}}{\partial \xi} + \frac{\partial \hat{F}}{\partial \eta} + \frac{\partial \hat{G}}{\partial \zeta} \tag{2.9}$$

is the flow residual. For spatial discretization, discrete flow variables, i.e. $\mathbf{Q}_{j,k,m}$ associated with each computational node (j,k,m), are used to construct a finite-difference approximation to the spatial partial derivatives in the Euler equations. The spatial discretization of the PDEs lead to a coupled system of ODEs [39, 46]. The semi-discrete form of the Euler equations can be written as:

$$\frac{1}{J}\frac{d\mathbf{Q}}{dt} + \mathbf{R}(\mathbf{Q}) = \mathbf{0} \tag{2.10}$$

where $\mathbf{Q} = [\mathbf{Q}_{1,1,1}, \mathbf{Q}_{1,1,2}, \ldots, \mathbf{Q}_{j,k,m}]^T$ is the vector of discrete flow variables, and $\mathbf{R}$ is the discrete residual, which also contains the appropriate boundary conditions.

2.3.1 Inviscid Fluxes

A second-order centered-difference scheme is used to discretize the inviscid fluxes in space for the interior nodes within each block. An artificial dissipation term is added to the discretized terms as well. The discrete residual $\mathbf{R}$ at each interior node can be expressed as:

$$\begin{aligned} \mathbf{R}(\mathbf{Q})_{j,k,m} &= \frac{\hat{\mathbf{E}}_{j+1,k,m} - \hat{\mathbf{E}}_{j-1,k,m}}{2} - \mathbf{D}^{(\xi)}_{j,k,m} + \\ &\quad \frac{\hat{\mathbf{F}}_{j,k+1,m} - \hat{\mathbf{F}}_{j,k-1,m}}{2} - \mathbf{D}^{(\eta)}_{j,k,m} + \qquad (2.11) \\ &\quad \frac{\hat{\mathbf{G}}_{j,k,m+1} - \hat{\mathbf{G}}_{j,k,m-1}}{2} - \mathbf{D}^{(\zeta)}_{j,k,m} \qquad (2.12) \end{aligned}$$

where $\mathbf{D}^{(\xi)}_{j,k,m}$, $\mathbf{D}^{(\eta)}_{j,k,m}$ and $\mathbf{D}^{(\zeta)}_{j,k,m}$ are the artificial dissipation terms in the ξ, η and ζ directions.

2.3.2 Artificial Dissipation

An artificial dissipation term is introduced to limit the production of high-frequency modes due to the nonlinear effects associated with the inviscid fluxes. These high frequency modes may introduce errors in our solution. The introduction of artificial dissipation also enhances the stability of our numerical scheme and prevents oscillations at discontinuities such as shock waves. Details on the dissipation model used can be found in [21, 48].

2.3.3 Block Interfaces and Boundary Conditions

For a multi-block finite difference scheme that uses the same interior scheme for discretization at block interfaces, layers of ghost or halo blocks are required to enable discretization

at interface nodes [23]. For example, to evaluate the fourth-difference dissipation term in the ξ direction at node J, information is required from nodes $J-1$ and $J-2$ in the local block, and from two other nodes from the adjacent block. A two-layer ghost block needs to be attached to that interface to store the appropriate values required for discretization. In a parallel implementation, this adjacent block is likely to be stored in a memory location that is only accessible by another processor. In this case, flow variables near block interfaces must be passed to the appropriate processors via message passing, affecting the scalability of the parallel method. In addition, mesh lines connecting adjacent blocks will be required to be sufficiently smooth in order to retain the desired accuracy. In three dimensions it is difficult to achieve the desired mesh smoothing requirements.

This difficulty associated with multi-block finite difference schemes is addressed through the application of summation by parts (SBP) operators and simultaneous approximation terms (SATs). The SBP operators simply retain the interior centered difference scheme, while at interfaces one-sided schemes are used for discretization. Conditions associated with block interfaces are enforced using SATs. The method has been successfully used to enforce boundary conditions [6] and block interfaces [8]. For details on the application of SATs to structured multi-block grids using the Euler and the RANS equations, see Hicken and Zingg [23] and Osusky and Zingg [42].

2.4 Newton-Krylov Flow Solver

For this work, all flow solutions are computed at steady-state conditions. At steady state, the flow variables do not change with time, therefore, $d\mathbf{Q}/dt = \mathbf{0}$, hence the flow residual must go to zero:

$$\mathbf{R}(\mathbf{Q}) = \mathbf{0} \tag{2.13}$$

Discretization of the steady flow equations results in a set of nonlinear equations. The set of nonlinear equations are solved using an inexact Newton method. Using the exact

Newton method as a basis, the following linear system is solved at each iteration:

$$\mathbf{A}^{(n)}\Delta\mathbf{Q}^{(n)} = -\mathbf{R}^{(n)} \tag{2.14}$$

where the matrix

$$\mathbf{A}^{(n)} = \left.\frac{\partial\mathbf{R}}{\partial\mathbf{Q}}\right|^{(n)} \tag{2.15}$$

and

$$\Delta\mathbf{Q}^{(n)} = \mathbf{Q}^{(n+1)} - \mathbf{Q}^{(n)} \tag{2.16}$$

Since the residual evaluation of the Euler or the RANS equations is based on the right-hand side of (2.14), the matrix $\mathbf{A}^{(n)}$ on the left-hand side of (2.14) can be modified to improve the convergence of the solution process without any effect on the evaluation of the flow residual. The solution process is therefore broken into two phases, an approximate-Newton phase which is used to find a suitable initial iterate and an inexact-Newton phase which uses the initial iterate and a modified $\mathbf{A}^{(n)}$ to compute the steady state solution. Specifics of the solution process as applied to the Euler and RANS equations can be found in [21, 23, 42].

The resulting linear system at each iteration of the approximate-Newton phase or inexact-Newton phase is solved inexactly using a Krylov subspace method. In particular the General Minimal Residual (GMRES) [50] method is found to be well-suited for CFD applications.

2.4.1 Preconditioning

To improve the convergence of GMRES, the linear system is preconditioned with a matrix $\mathbf{M}$. The matrix $\mathbf{M}$ is obtained through an incomplete lower-upper (ILU) factorization of the modified Jacobian using a fill level of p. This factorization is local to each process and does not involve any form of inter-processor communication. Two parallel preconditioners are available to the flow solver. One is based on the additive-Schwarz method and the other is based on the approximate-Schur method.

Additive-Schwarz Preconditioning

The additive-Schwarz preconditioning is basically a block-Jacobi iteration; see Saad [50]. For a given vector $\mathbf{v}$, the local component of the preconditioned vector $\mathbf{z}$ is given by the

$$\mathbf{z}_i = \mathbf{M}_i^{-1}\mathbf{v}_i \tag{2.17}$$

which is the exact or inexact solution to the system

$$\mathbf{M}_i\mathbf{z}_i = \mathbf{v}_i \tag{2.18}$$

where $\mathbf{M}_i$ is the local component of the ILU(p) factorization: $\mathbf{L}_i\mathbf{U}_i$, of the modified Jacobian. There is no inter-processor communication for the additive-Schwarz preconditioning since all computations remain local to each process. The linear system in (2.18) is solved approximately to obtain the preconditioned vector $\mathbf{v}_i$ through a forward and backward substitution of $\mathbf{L}_i$ and $\mathbf{U}_i$.

Approximate-Schur Preconditioning

The approximate-Schur preconditioner presented here follows the development by Saad and Sosonkina [51, 50]. In formulating the approximate-Schur preconditioner for GMRES, the matrix is divided among S sub-domains:

$$\begin{bmatrix} A_{11} & \cdots & A_{1s} \\ \vdots & & \vdots \\ A_{s1} & \cdots & A_{ss} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_s \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_s \end{bmatrix} \tag{2.19}$$

where $A_{i1} \ldots A_{is}$ are stored on processor i. The sub-domains are generally non-overlapping. With careful ordering of nodes, the diagonal blocks (A_{ii}) and off-diagonal blocks (A_{ij}) in each sub-domain can be written as:

$$A_{ii} = \begin{bmatrix} B_i & E_i \\ F_i & C_i \end{bmatrix}, \quad A_{ij} = \begin{bmatrix} 0 \\ E_{ij} \end{bmatrix} \tag{2.20}$$

where B_i are the internal entries, C_i are the interface entries, and E_i, E_{ij} and F_i represent the coupling between internal and interface entries. The solution vector $\mathbf{x}$ and the right-hand-side $\mathbf{b}$ can be similarly partitioned for internal and interface entries:

$$x_i = \begin{bmatrix} u_i \\ y_i \end{bmatrix}, \quad b_i = \begin{bmatrix} f_i \\ g_i \end{bmatrix} \tag{2.21}$$

The part of the linear system that is local to the sub-domain i can now be written as:

$$\begin{bmatrix} B_i & E_i \\ F_i & C_i \end{bmatrix} \begin{bmatrix} u_i \\ y_i \end{bmatrix} + \begin{bmatrix} 0 \\ \sum_{i \neq j} E_{ij} y_j \end{bmatrix} = \begin{bmatrix} f_i \\ g_i \end{bmatrix} \tag{2.22}$$

The local entries u_i can be solved once the interface entries y_i are known:

$$u_i = B_i^{-1}(f_i - F_i y_i) \tag{2.23}$$

Substituting u_i into the equation for y_i the following linear system for the interface can be obtained:

$$S_i y_i + \sum_{i \neq j} E_{ij} y_i = g_i - F_i B_i^{-1} f_i \tag{2.24}$$

where S_i is the "local Schur complement matrix", defined as:

$$S_i = C_i - F_i B_i^{-1} E_i \tag{2.25}$$

S_i is generally dense. When written out for all sub-domains, the overall linear system from (2.24) now becomes:

$$\begin{bmatrix} S_1 & E_{12} & \cdots & \cdots & E_{1s} \\ E_{21} & S_2 & E_{23} & \cdots & E_{2s} \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ E_{s1} & E_{s2} & E_{s3} & \cdots & S_s \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ \vdots \\ y_s \end{bmatrix} = \begin{bmatrix} g_1' \\ \vdots \\ \vdots \\ \vdots \\ g_s' \end{bmatrix} \tag{2.26}$$

The left-hand-side matrix is referred to as the "global Schur complement matrix". In theory, this Schur complement system can be assembled and solved for each y_i, then for

the internal unknowns u_i in (2.22). However, solving this linear system exactly is not competitive. Instead, a system that approximates (2.26) is used as a preconditioner for the global problem (2.14).

Suppose A_{ii} has been factored into $A_{ii} = L_i U_i$, where

$$L_i = \begin{bmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{bmatrix}; \quad U_i = \begin{bmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{bmatrix} \tag{2.27}$$

It can be shown that

$$S_i = L_{S_i} U_{S_i} \tag{2.28}$$

Therefore the LU decomposition of the local Schur complement matrix can be obtained through the LU decomposition of the local matrix A_{ii}. Similarly, an approximate factorization of S_i can be found by the ILU factorization of A_{ii}. The interface unknowns y_i can be solved approximately using this approximate local Schur complement system:

$$y_i = U_{S_i}^{-1} L_{S_i}^{-1} \left[g_i - F_i B_i^{-1} f_i - \sum_{i \neq j} E_{ij} y_j \right] \tag{2.29}$$

This equation is equivalent to a single block-Jacobi iteration, which can be solved using a Krylov-method such as GMRES. Once y_i is found and exchanged on all sub-domains, the local unknowns u_i can be solved approximately, again with ILU factorization.

A linear solver that uses the approximate Schur preconditioner must be "flexible", in that the preconditioning may change from one inner iteration to the next. In light of this, a flexible variant of GMRES ("FGMRES") [50] is used. The details of the approximate-Schur preconditioning technique with modifications by Hicken and Zingg can be in found [23].

Chapter 3

Load Balancing Strategy

Massively parallel computers have proven to improve turnaround times for large-scale CFD applications. These computer systems use a divide and conquer approach to solve the large-scale problems assigned to them. They also provide considerable amounts of memory hitherto unavailable on single processor machines. For large-scale CFD problems that use multi-block grids, parallelization becomes straightforward since the grid generation takes care of the domain decomposition process. This approach to parallelization is usually referred to as coarse-grain parallelism.

This chapter provides a detailed description of a static and iterative load balancing algorithm and results obtained for test cases on a 2D multi-block grid.

The algorithm and results presented in this chapter and part of Chapter 4 were presented in a recent paper [5] at the CFD Society of Canada 2012 conference.

3.1 Coarse-grain Parallelism Approach

In the coarse-grain parallelism approach, entire blocks are assigned to the processors. Only the grid interconnections are affected. The coarse-grain parallelism is usually referred to as domain decomposition and is a popular approach in CFD. This strategy is well suited for a multigrid approach [2] because all grid-levels on a block can be assigned

to one processor. The computational workload is balanced by combining or packing the blocks onto the processors with some load balancing algorithm, for example with any greedy-like method [57, 58] or bin-packing method [36] to minimize a cost function.

However, assigning these large entities of work can cause serious load balancing problems if the number of blocks and block sizes do not match the number of processors. A remedy is to use a pre-processing step to split larger blocks into smaller ones, introducing more internal boundaries [36, 52]. In [2], Ålund *et al.* presented a block-splitting algorithm with considerations for the numerical method and boundary layer thickness. When the boundary layer is not divided between two blocks in the direction normal to the wall in the original partitioning, then the same holds true in the new partitioning. The recursive edge bisection (REB) method for splitting blocks is also discussed and used in [57, 58]. This method does not take into consideration the numerical method and boundary layer constraints. Splitting blocks can have some drawbacks. Many small blocks are a better choice as regards load balance, but communication overhead between blocks is increased.

The coarse-grain load balancing strategy can be implemented either statically or dynamically. In the static sense, the load balancing tool is used as a pre-processing step without any consideration for changes in flow solution and processor characteristics. In dynamic load balancing, changes in flow solution and processor characteristics are considered, and the load balancing tool is executed in parallel. Static load balancing is ideal for homogeneous processors while dynamic load balancing is more suited for an adaptive solution process or a heterogeneous set of processors.

We retain the basic components of the load balancing algorithm in [26] and make additions in terms of our choice of blocks to split, a constraint for limiting block splitting, and the ability to start the tool off with a smaller number of blocks than processors.

3.2 Load Balancing Algorithm

It is impractical to assume that structured multi-block grids will always have the same block sizes. One can expect different block sizes for a multi-block grid generated around multi-element airfoils, wings, and wing-body configurations. The challenge with solving such problems on massively parallel computers is the ability to load balance in order to improve turnaround times. Since a CFD simulation will proceed at the rate of the process with the largest amount of work, it is important that a tool be made available to spread the work as evenly as possible.

For a CFD solution process that executes the same instructions per node, it is safe to assume that a measure of workload on a process is the number of computational nodes assigned to that process. In addition, a block splitting strategy can be implemented for the load balancing strategy employed since the numerical method and boundary layer constraints need not be considered because of our constant CPU time assumption.

Our Newton-Krylov solution framework uses multi-block grids and executes the same instructions per node on a homogeneous distributed system; therefore a static load balancing tool is sufficient. Such a tool is fast to develop and implement without having to make considerable changes to the CFD solution algorithm.

For the algorithm presented here we seek to reduce a cost function (i.e. the ratio of the maximum number of grid nodes on any processor to the average number of grid nodes per processor). Assuming a constant CPU time per computational node, the workload for a given block, w_b is defined as:

$$w_b = n_i \times n_j \times n_k \tag{3.1}$$

where n_i, n_j and n_k are the number of nodes in the i, j and k index directions of the block. We define the workload for a partition (i.e. total workload on a processor) as:

$$w_p = \sum_{s=1}^{n_{b,p}} w_{b,s} \tag{3.2}$$

where $n_{b,p}$ is number of blocks assigned to the processor. Blocks are sorted in descending order of size using the insertion sort algorithm [12]. The sorted list of blocks is assigned to processors in a loop, starting with the largest block and then assigning subsequent blocks to the processor with the minimum workload at each instance of assignment. We also define a set of blocks, $\mathbf{W_b}$ and a set of partitions, $\mathbf{W_p}$ as follows:

$$\mathbf{W_b} = \{w_{b,1}, w_{b,2}, ..., w_{b,n_b}\} \tag{3.3}$$

$$\mathbf{W_p} = \{w_{p,1}, w_{p,2}, ..., w_{p,n_p}\} \tag{3.4}$$

For a set of blocks, $\mathbf{W_b}$, n_p processors and a set of partitions, $\mathbf{W_p}$, an average workload, $\bar{w}$ and workload ratio, $w_{r,m}$ are defined as:

$$\bar{w} = \frac{\sum_{m=1}^{n_p} w_{p,m}}{n_p} \tag{3.5}$$

$$w_{r,m} = \frac{w_{p,m}}{\bar{w}},\ m = 1, 2, 3, ..., n_p \tag{3.6}$$

The user specifies a load-balance threshold factor, f_t, and blocks are split and assigned until the workload ratio, $w_{r,m}$, on each processor lies at or below the threshold.

After a block assignment iteration, the processors with their workload ratio, $w_{r,m}$ exceeding the workload threshold, f_t have their largest block split halfway (for an odd number of nodes) or near halfway (for an even number of nodes) along the index with the maximum number of nodes. For example a user workload threshold specification of 1.05 ensures that the workload on any processor is kept to less than or equal to 1.05 times the average workload. The process of assigning and enforcing workload threshold is repeated until $\max(w_{r,m}) \leq f_t$.

Two additional constraints are introduced specific to the nature of the multi-block grid (homogeneous or heterogeneous) under consideration. The block splitting factor, b_r is set by the user to enforce the following constraint at the n^{th} iteration:

$$\frac{n_b}{n_p} \leq b_r \tag{3.7}$$

When invoked, this constraint overrides the workload balance. It ensures that for homogeneous blocks the ratio of the number of blocks to the number of processors does not exceed b_r. The lower bound on the ratio of the number of blocks to the number of processors is unity. Therefore b_r controls the upper limit of this ratio to prevent excessive block splitting, which has the disadvantage of increasing the problem size significantly.

When the tool is started with a smaller number of blocks than processors the average workload per process can be smaller than the smallest block in the grid file. Based on the size of the smallest block the user can specify a factor which allows only blocks greater than the smallest by this factor to be split. Once there are enough blocks to form a partition on all processors, the main load-balancing loop is started since the new average workload will either be equal or greater than the smallest block in the mesh. The block size factor, f_b is used to determine which blocks need to be split by enforcing the following constraint:

$$\frac{w_{b,s}}{\min(w_b)} \leq f_b, \quad s = 1, 2, 3, ..., n_b \tag{3.8}$$

where n_b is the number of blocks.

The algorithm is presented below:

Algorithm

n_b = number of blocks in grid file

n_p = number of processors specified

$\mathbf{W_b}$ = is the set of initial block sizes sorted in descending order

$\mathbf{W_{b,new}}$ = is the set of new block sizes sorted in descending order

w_b = block workload

w_p = processor workload

w_r = processor workload ratio

p_x = number of processors with workload exceeding load balance threshold

f_t = load balance threshold specified

b_r = blocks-to-processor ratio

$\min(j)$ = index of processor with the minimum workload

$\mathbf{W_b} = (w_{b,1}, w_{b,2}, ..., w_{b,n_b})$ // the set of blocks in descending order of size

$p_x = 0$ // set the number of processors with $w_r > f_t$ to zero

repeat

$\bar{w} = \frac{\sum_{i=1}^{n_b} w_{b,i}}{n_p}$ // calculate the average workload

$w_{p,j} = 0$, $j = 1, n_p$ // set all processor workloads to zero

for $i = 1$ to n_b **do**

$w_{p,\min(j)} = w_{p,\min(j)} + w_{b,i}$ // assign block to processor with minimum work

end for

for $i = 1$ to n_p **do**

$w_{r,j} = \frac{w_{p,j}}{\bar{w}}$ // calculate workload ratio for processor

if $w_{r,j} > f_t$ **then**

$p_x = p_x + 1$ // keep count of number of processors in violation of workload threshold constraint

end if

end for

if $p_x > 0$ **then**

Split blocks associated with $w_{b,i}$, $i = 1, p_x$

$n(\mathbf{W_{b,new}}) = n(\mathbf{W_{b+p_x}})$ // set the number of new blocks equal to number of old blocks plus p_x

$\mathbf{W_b} = \mathbf{W_{b,new}}$ // set blocks for nth+1 iteration equal to new blocks

end if

until $p_x = 0$ **or** $\frac{n_b}{n_p} >= b_r$

3.3 Application to a 2D Structured Multi-block Grid

Some preliminary tests were performed and the results are presented below. For the purpose of clarity and visualisation, these tests were performed on a 2D grid. A C-mesh was generated for the RAE 2822 airfoil using AMBER2d[1]. The starting mesh had a total of 18915 nodes decomposed into 3 blocks. Load balance thresholds of 1.05, 1.10, 1.15, and 1.20 were tested using 10, 12 and 15 processors. To provide a basis for comparing the workload balance achieved for the different thresholds set on the different number of processors used, two additional parameters are defined. The first is a mesh-size factor, c, defined as follows:

$$c = \frac{\bar{w}_f}{\bar{w}_o}$$

where $\bar{w}_o$ is the average workload in the initial mesh and $\bar{w}_f$ is the average workload in the final mesh The second is the effective load balance threshold, $f_{t,e}$, defined as:

$$f_{t,e} = f_t \times c$$

For every iteration performed the requirement of the load balance threshold is enforced based on the new mesh generated. The mesh size factor is a measure of the grid nodes added as a result of introducing new interfaces during block splitting. Hence $f_{t,e}$, is the effective load balance threshold allowed on each processor based on the initial mesh. Therefore for any two or more load balancing cases started off from the same initial mesh and number of processors but different workload thresholds, the case with a lowest effective workload threshold will complete its simulation first. Also, the effective workload ratio enables us to determine the actual cost of using different thresholds on a specific number of processors.

Table 3.1 shows the mesh size factors and the effective load balance thresholds obtained for the cases tested. Figure 3.1 shows the initial mesh with the edges of the blocks

[1] 2D Automated Multi-Block Elliptic gRid generator developed in UTIAS by A.R. Wilkinson & D. W. Zingg

	$p = 10$		$p = 12$		$p = 15$	
f_t	c	$f_{t,e}$	c	$f_{t,e}$	c	$f_{t,e}$
1.05	1.04	1.09	1.07	1.13	1.11	1.17
1.10	1.03	1.13	1.07	1.18	1.07	1.18
1.15	1.02	1.17	1.07	1.24	1.05	1.20
1.20	1.02	1.23	1.03	1.23	1.05	1.26

Table 3.1: Values of c and $f_{t,e}$ obtained for the cases considered

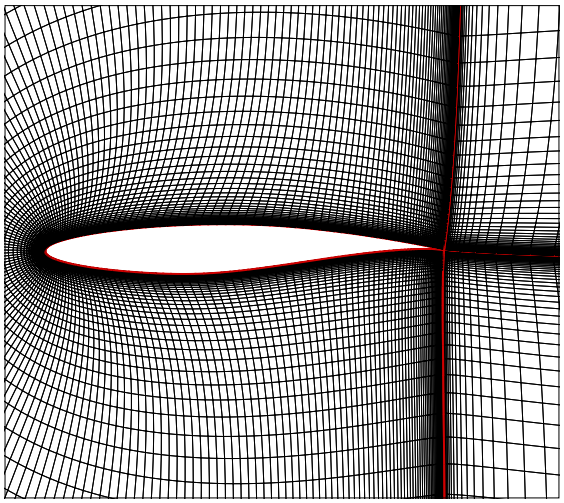


Figure 3.1: Initial C-mesh for RAE 2822 airfoil

coloured red.

3.3.1 10 Processors

Starting with a load balance threshold of 1.05, the load balancing tool was tested using the initial mesh for the RAE 2822 airfoil. A single iteration of the load balancing tool

involves block splitting and assignment of workload toward the enforcement of the load balance threshold constraint. With $f_t = 1.05$, 5 iterations were required to meet the specified load balance threshold. The decomposed mesh has 18 blocks with a total of 19,602 nodes, giving $c = 1.04$. In the second case, f_t was set to 1.10. Four iterations were performed in order to satisfy this requirement. The new mesh has 14 blocks with a total of 19,470 nodes, giving $c = 1.03$. Finally, with f_t set to 1.15 and 1.20, 3 iterations were performed, and the new mesh has 10 blocks and a total of 19,338 nodes or $c = 1.02$. The reduction in the additional nodes due to decomposing the domain is due to a reduction in the number of block splittings as a result of increasing the load balance threshold. Figures 3.2 and 3.3 show the workload distribution on each processor and the new block decompositions generated, respectively.

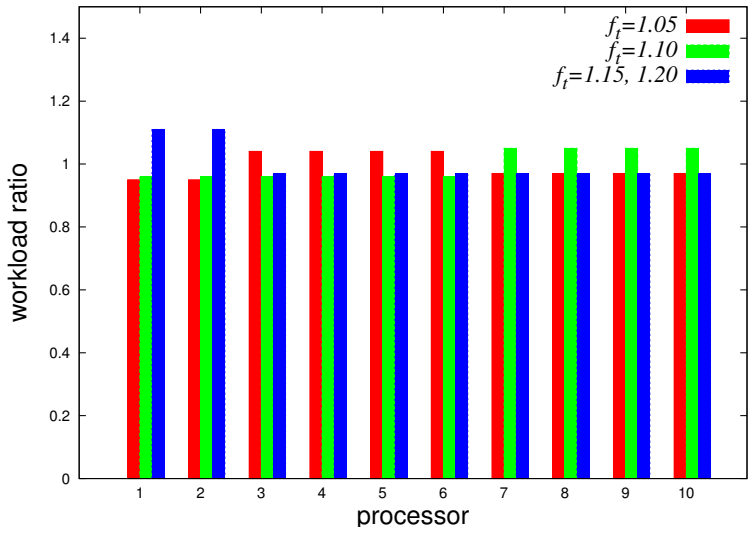
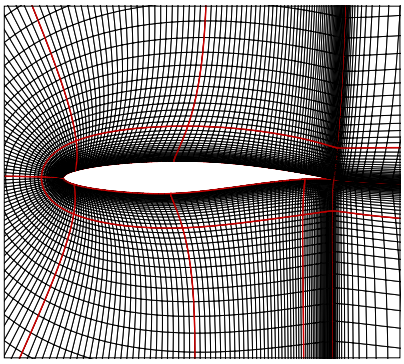
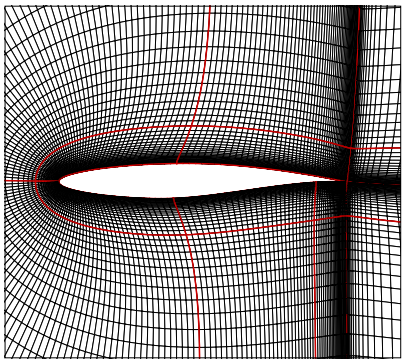


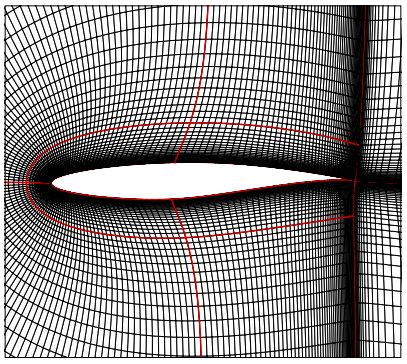
Figure 3.2: Workload distribution for 10 processors



(a) $f_t = 1.05$, 18 blocks, 19,602 nodes, 5 iterations



(b) $f_t = 1.10$, 14 blocks, 19,470 nodes, 4 iterations



(c) $f_t = 1.15$ and $f_t = 1.20$, 10 blocks, 19,338 nodes, 3 iterations

Figure 3.3: Domain decomposition for 10 processors

3.3.2 12 Processors

Similar to the first case considered, an initial load balance threshold of 1.05 was tested using 12 processors. Nine iterations were performed to meet the requirement set by the load balance threshold. The decomposed mesh has 44 blocks with a total of 20,332 nodes,

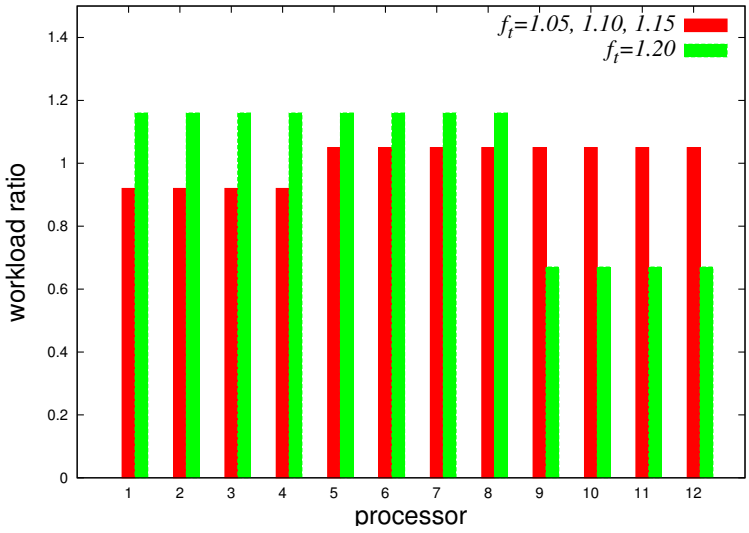
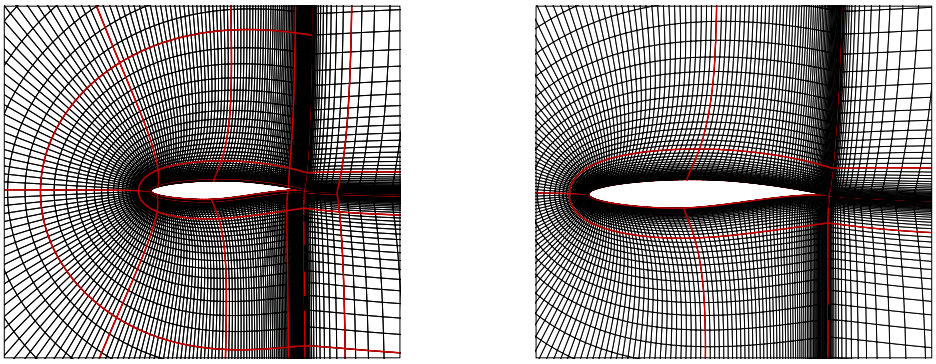


Figure 3.4: Workload distribution for 12 processors

giving $c = 1.07$. With load balance thresholds of 1.10 and 1.15, the results obtained are the same as using a threshold of 1.05. Finally, f_t was set to 1.20. In this case, 3 iterations were performed, and the new mesh has 12 blocks and a total of 19,404 nodes, giving $c = 1.03$. In comparison with the 10-processor case, a larger number of iterations was required for the 12-processor case for a specific threshold. However, a single load-balancing iteration is not expensive. Figure 3.4 shows the workload distribution on the processors. Figure 3.5 shows the decomposed domains for the 12 processors considered.

3.3.3 15 Processors

Finally, all the load balance thresholds considered in the previous cases were tested using 15 processors. With f_t set to 1.05, 15 iterations were performed. The resulting mesh has 85 blocks and a total of 21,029 nodes, giving $c = 1.11$. Using a threshold of 1.10, 8 iterations were performed, and the new mesh had 42 blocks and a total of 20,298 nodes or $c = 1.07$. The last two cases considered have f_t set to 1.15 and 1.20 respectively. Five iterations were performed for both cases, and the resulting meshes have the same properties: 25 blocks and a total of 19,829 nodes, or $c = 1.05$. Figure 3.6 shows the



(a) $f_t = 1.05$, $f_t = 1.10$, and $f_t = 1.15$, 44 blocks, 20,332 nodes, 9 iterations

(b) $f_t = 1.20$, 12 blocks, 19,404 nodes, 3 iterations

Figure 3.5: Domain decomposition for 12 processors

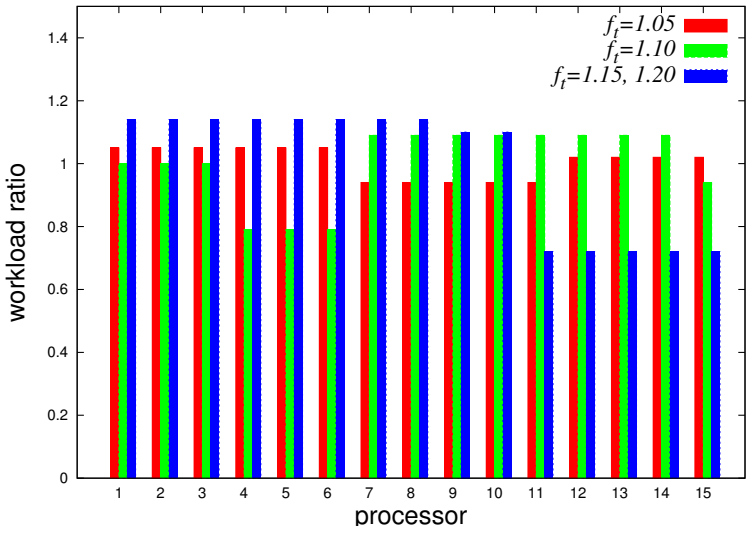
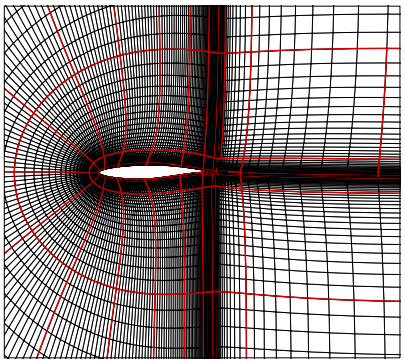
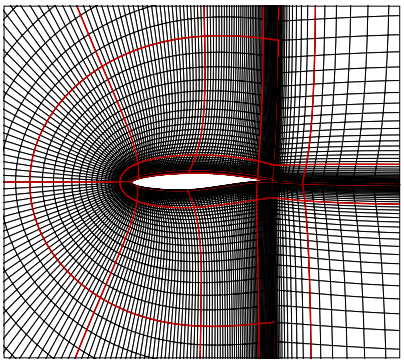


Figure 3.6: Workload distribution for 15 processors

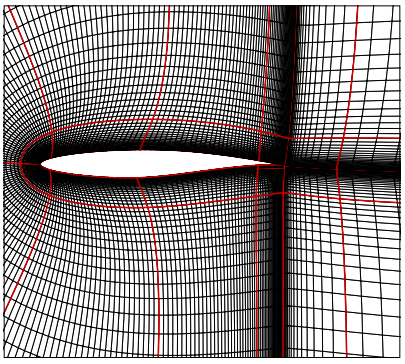
workload distribution on the processors. Figure 3.7 shows the re-decomposed domains for the 15 processors considered.



(a) $f_t = 1.05$, 85 blocks, 21,029 nodes, 15 iterations



(b) $f_t = 1.10$, 42 blocks, 20,298 nodes, 8 iterations



(c) $f_t = 1.15$ and $f_t = 1.20$, 25 blocks, 19,829 nodes, 5 iterations

Figure 3.7: Domain decomposition for 15 processors

3.4 Remarks on Results

For all the cases considered the workload threshold constraint set was met. For load-balance thresholds closer to unity, the mesh size factors increase considerably. Although smaller block sizes are ideal for the purposes of load balancing, the increased block

splitting associated with smaller thresholds introduces an extra overhead in terms of communication and additional grid nodes.

As the number of processors is increased, load balancing becomes more difficult, since a number of block splittings is required to initially form a partition before load balancing constraints are enforced. This difficulty also increases the number of iterations required to complete a single load balancing run.

In the scaling study results presented in the next chapter we use load-balance thresholds greater than 1.05 but less than 1.50. The expectation is that the load balance thresholds chosen will provide a compromise between obtaining a "near" perfect load balance and keeping mesh size factors low. An example of the application of the block size constraint and the improvements gained through its enforcement is also shown in the next chapter.

It is worth mentioning that this tool takes a few seconds to minutes to decompose domains and output the necessary files required to partition blocks across processors to be used. Thus the cost of using this tool is nearly insignificant compared to the turnaround times for flow solutions.

Chapter 4

Scaling Studies for the Newton-Krylov Algorithm

Prior to this work, the number of blocks required to simulate flows using our parallel Newton-Krylov solution framework was always a multiple of the number of processors. When the number of blocks are more than the number of processors, blocks are combined to form a partition on each processor. In this thesis, a *partition* will refer to a block or a combination of blocks assigned to a processor during the solution process of our Newton-Krylov algorithm.

One of the objectives of this work is to integrate the load-balancing tool into our Newton-Krylov solution process. To achieve this goal, the Fortran 90 *Partition module* which contains all the subroutines for forming partitions and performing all partition level computations was extended to accommodate an option to use the load-balancing tool if desired by the user. The load-balancing tool returns new grid-related files as well as a new input file (*input.proc*) after execution. The new input file contains the rule for combining blocks to form a partition based on the decomposition determined by the load-balancing tool.

For the purposes of scaling studies, new subroutines were added to the Fortran 90

Common module for the solver in order to provide execution time information for various parts of the computation process. Using *MPI_Wtime()*, the time taken to complete the ILU(k) factorization, preconditioning, communication and the overall computation is recorded.

Two types of multi-block grids are considered for our scaling studies. The first is a homogeneous multi-block grid, the second is a heterogeneous multi-block grid. The geometries considered are the ONERA M6 wing and the Common Research Model (CRM) wing-body geometry from the Drag Prediction Workshop (DPW) -IV. The JAXA set of grids were used in the case of the CRM wing-body geometry. Results and discussion for these grids are presented in the subsequent sections.

For this study we consider two different governing equations, the Euler equations and the Reynolds-Averaged Navier-Stokes (RANS) equations with the Spalart-Allmaras one-equation turbulence model.

All results obtained in this chapter were performed on an IBM iDataPlex DX360M2 cluster based on Intel's Nehalem architecture. Each computing node on the cluster has 8 cores of Intel Xeon E5540 processors sharing 16GB of RAM and a clock speed of 2.53GHz. Results were obtained on the nodes interconnected with non-blocking double data rate (DDR) InfiniBand.

4.1 Parallel Scaling

An understanding of an algorithm's parallel scaling is important, since it enables us to determine how many processors may be effectively utilized for a given problem size. Such a study also helps determine the approximate speedups that can be achieved while increasing the number of processors. There are two main scaling studies performed for parallel applications. They are strong and weak scaling studies respectively.

For strong scaling studies, the problem size is kept fixed while the number of processors

used for computations is varied. Such scaling studies have been performed in [22] for the present Newton-Krylov algorithm, and the results show a good strong scaling for the algorithm.

Weak scaling performance is determined by fixing the workload per processor and varying the problem size with the number of processors. Results that have been obtained so far have not been as impressive as the strong scaling, and work is ongoing to help improve this performance indicator for the algorithm

To measure scaling properties such as speedup and efficiency we use the following relations. An ideal time, $t_{p,\ ideal}$ is defined as:

$$t_{p,\ ideal} = \frac{t_1}{n_p} \tag{4.1}$$

where t_1 is the computational time if the problem is run on a single processor (serial case), and n_p is the number of processors. Since none of the cases under consideration can be run on a single processor, (4.1) is modified slightly to:

$$t_{p,\ ideal} = \frac{t_p^{(o)} \times n_p^{(o)}}{n_p} \tag{4.2}$$

where $t_p^{(o)}$ and $n_p^{(o)}$ are the execution time and number of processors for the smallest number of processors in the scaling study group.

Finally, the speedup, S_p and efficiency, η_p for a number of processors is given as:

$$S_p = \frac{t_p^{(o)}}{t_p} \tag{4.3}$$

$$\eta_p = \frac{t_{p,\ ideal}}{t_p} \tag{4.4}$$

where t_p is the actual execution time recorded.

4.2 Homogeneous Multi-block Grids

We refer to homogeneous multi-block grids as grids with blocks of the same size throughout the domain. We use a 64-block grid generated around the ONERA M6 wing for all

Properties	ONERA M6
Blocks	64
Nodes	3,241,792
$\text{block}_{\text{min}}$	$37 \times 37 \times 37$
$\text{block}_{\text{max}}$	$37 \times 37 \times 37$
Equations	Euler
α	3.06°
Re	-
M	0.699

Table 4.1: Mesh properties and flow conditions for the homogeneous case

of the cases in this section. Table 4.1 gives an overview of the grid properties and flow conditions used for the cases considered here. Figure 4.1 shows the coefficient of pressure (C_p) obtained. We consider two cases based on the initial number of processors started with for the scaling studies since we extrapolate the ideal time based on these starting points.

In Figure 4.2, we show a combined scaling diagram for the two cases considered for homogeneous multi-block grids. In addition we include the ideal cases based on the two extrapolation points, 16 and 180 processors. From Figure 4.2 we observe that with 128, 320, 640, 720 and 960 and 1024 processors we obtain lower execution times. This occurs because 128, 320, 640, 720, 960 and 1024 are multiples of the initial number of blocks (64). If the ratio of the number of processors to the number of initial blocks is equal to a power of 2, then we obtain a perfect load balance considering our block splitting strategy. Therefore for 128 and 1024 processors we obtain a perfect load balance. In the cases (320, 640, 720 and 960 processors) where the number of processors to number of blocks ratio is not equal to a power of 2, we still obtain a near perfect load balance. Therefore, for

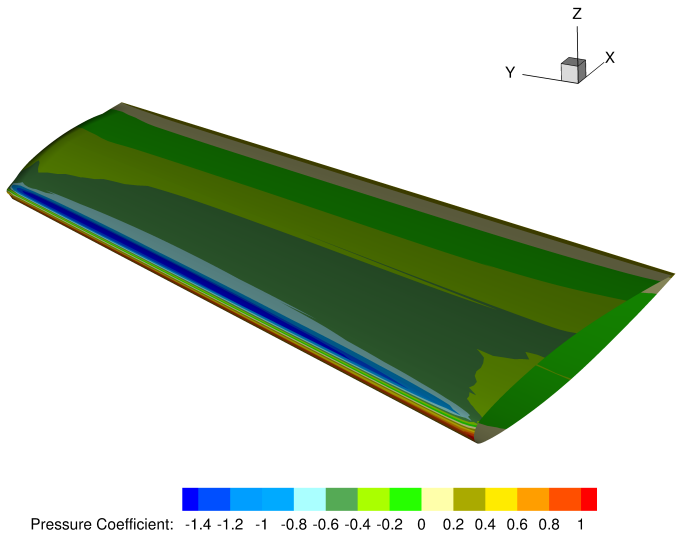


Figure 4.1: Coefficient of pressure distribution on the ONERA M6 wing using 96 processors with $f_t = 1.10$

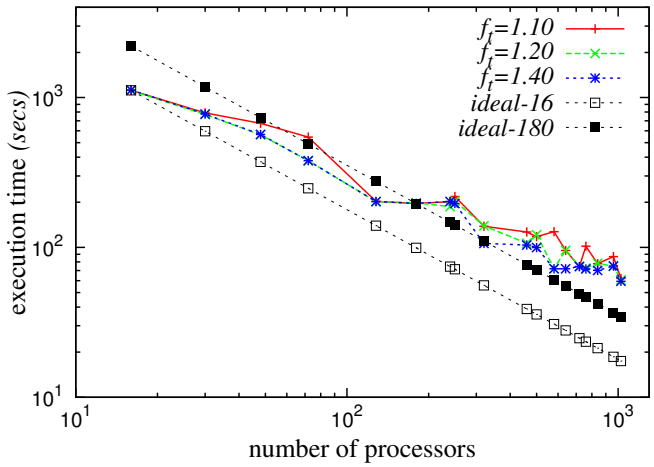
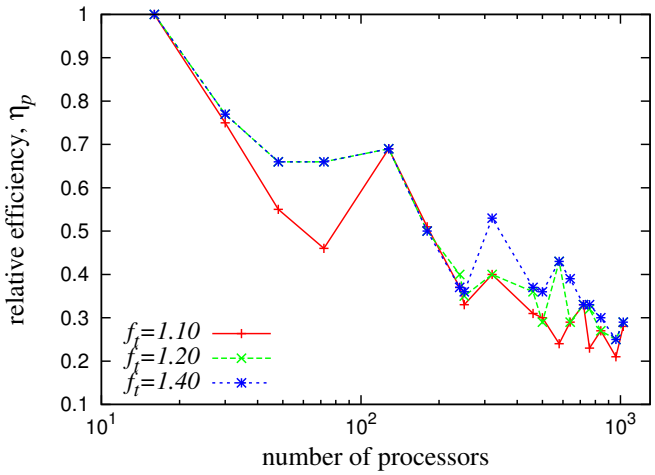


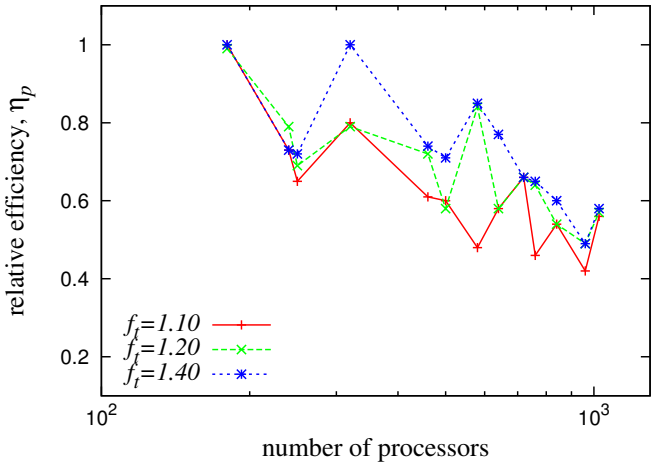
Figure 4.2: Execution times obtained for 16, 30, 48, 72, 128, 180, 240, 250, 320, 460, 500, 580, 640, 720, 760, 840, 960 and 1024 processors.

homogeneous multi-block grids lower turnaround times and better performance can be achieved when the number of processors is a multiple of the number of blocks.

The relative parallel efficiencies achieved using 16 and 180 processors as extrapolation points are shown in Figures 4.3a and 4.3b respectively. Further, the results in Figure 4.2



(a) Relative efficiencies based on 16 processors



(b) Relative efficiencies based on 180 processors

Figure 4.3: Relative efficiencies, η_p

are broken down into two sets of data in Subsections 4.2.1 and 4.2.2. In the first instance, we assume a scaling study where the initial number of blocks is greater than the initial number of processors. The second group starts off with a smaller number of blocks than the initial number of processors. The second instance is possible by splitting the initial set of blocks (64) into enough blocks to form 180 partitions while for the first case no block splitting is required.

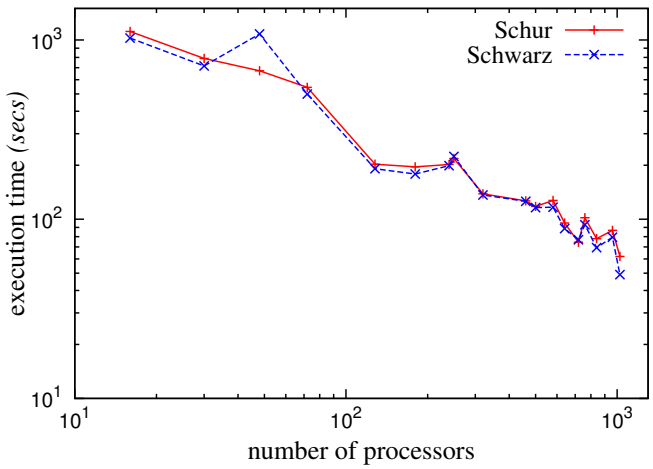
The execution times recorded using Schur and Schwarz parallel preconditioning techniques are shown Figures 4.4a, 4.4b, 4.4c. A comparison of the two preconditioning

techniques was prompted by an initial investigation (see Figure 4.5) into how the number of blocks assigned to a processor affects the execution time for a simulation using the Schur and Schwarz parallel preconditioners. In this study, a 16-block mesh is split into 32, 64, 128, 256, 512 and 1024 blocks. The resulting blocks are assigned to 16 processors leading to 1, 2, 4, 8, 16, 32 and 64 blocks per processor. As the number of blocks per processor increases the Schwarz preconditioner performs better than the Schur preconditioner.

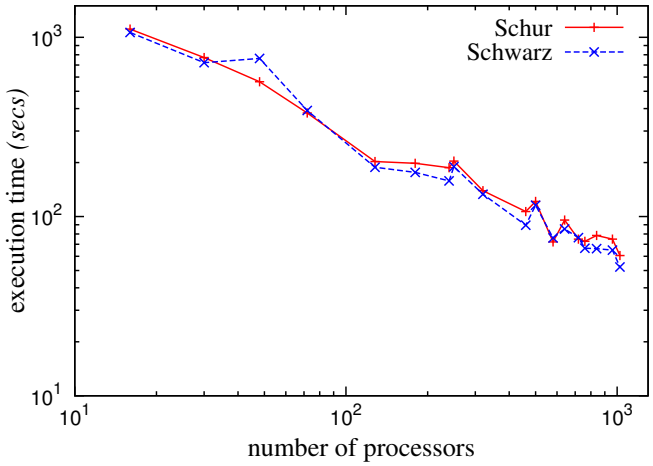
In Figure 4.4 the preconditioning techniques considered show a similar trend in execution times although slight variations in the values are observed. When $f_t = 1.10$ and $f_t = 1.20$, the execution time recorded using the Schwarz preconditioning technique is lower than the Schur technique as the number of processors is increased. A combination of a smaller load-balance factor and an increase in the number of processors leads to several block splits which in turn increases the number of interface nodes. An increase in the interface nodes increases the time taken to precondition the linear system using the Schur technique as compared to the Schwarz technique which is just a block-Jacobi iteration. When $f_t = 1.40$, an increase in the number of processors does not increase the number of block edge-cuts compared to $f_t = 1.10$ and $f_t = 1.20$, leading to a lower turnaround time for the Schur technique for 300 processors upwards. Using the Schwarz technique, more Krylov iterations are required to solve the same problem although the time taken to precondition the system is lower compared to the Schur. The fewer Krylov iterations required to solve the linear system in the case of the Schur technique compensates for the time taken to precondition the system.

4.2.1 Case I : $n_b^{(o)} > n_p^{(o)}$

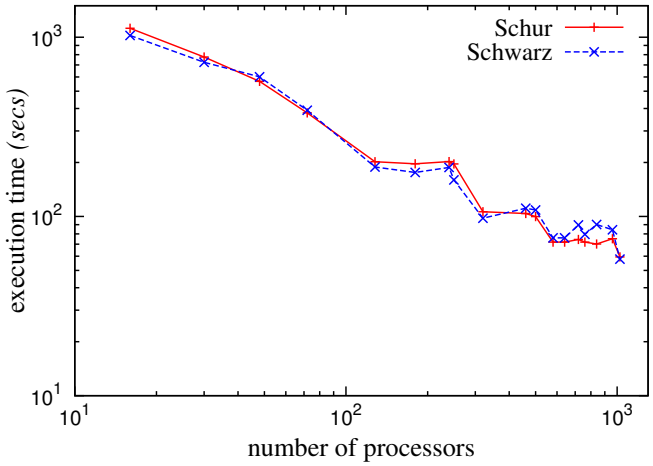
This case and the subsequent one break the results in Figure 4.2 into two cases. The first case considers a starting point where the number of blocks exceed the number of processors. Figure 4.6a shows how the Newton-Krylov algorithm described in Chapter 3



(a) Schur vs. Schwarz preconditioning for $f_t = 1.10$



(b) Schur vs. Schwarz preconditioning for $f_t = 1.20$



(c) Schur vs. Schwarz preconditioning for $f_t = 1.40$

Figure 4.4: A comparison of the Schur and Schwarz preconditioning techniques for the homogeneous multi-block grid for the ONERA M6 wing

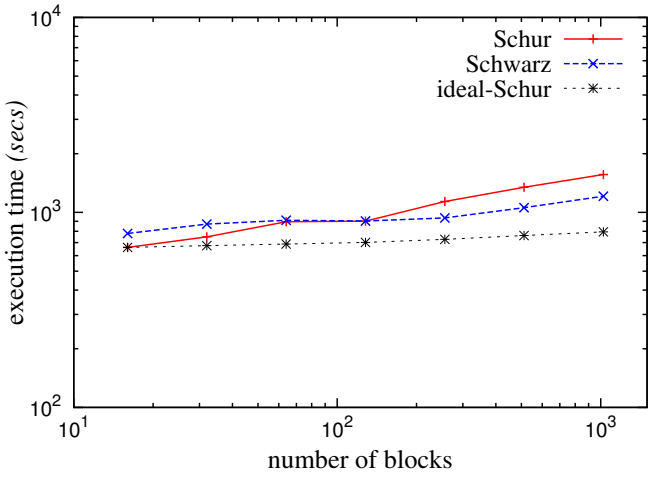
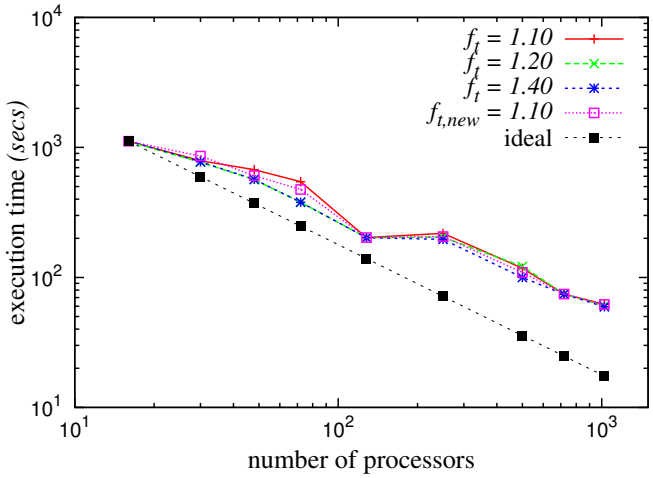


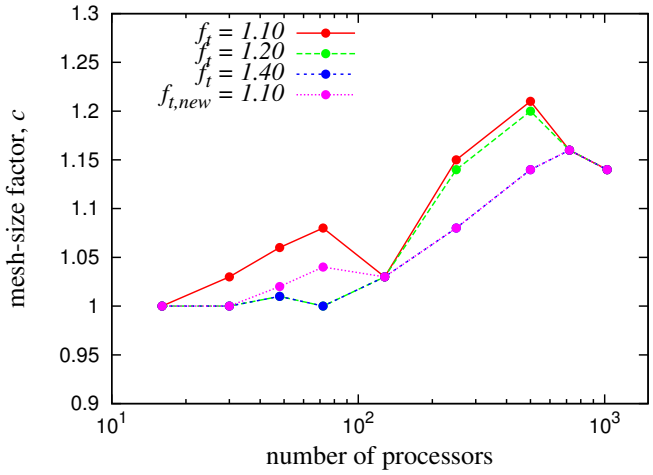
Figure 4.5: Execution times obtained comparing the Schur and Schwarz preconditioners with varying number of blocks on 16 processors

scales as we increase the number of processors. As we increase the number of processors beyond the number of initial blocks, blocks have to be split several times in order to meet the workload threshold set. In cases where a low f_t is set or the number of processors far exceeds the number of initial blocks, a high mesh-size factor c results due to the increased number of block splittings. For example, in the case of 250 processors, c goes as high as 1.15, as seen in Figure 4.6b when f_t=1.10. This indicates that the number of nodes in the grid increases by 15% after the final load balancing iteration. Considering our parallel solution process, a high c results in longer preconditioning time and a higher number of Krylov iterations, as observed in Figures 4.7a and 4.7b.

During the Schur preconditioning stages, approximate solution values are obtained for all the internal interface nodes, which is accelerated using GMRES. Consider a single block of size $37 \times 37 \times 37$ which is split into two in all three index directions resulting in 8 blocks; we record a c value of approximately 1.08. This $\sim$8% increase in the size of the problem results in a $\sim$100% increase in the number of interface nodes translating into a larger Schur complement problem, therefore increasing preconditioning time significantly. Figure 4.7a shows the CPU time spent at the preconditioning stages for the cases in Figure



(a) Execution times obtained for 16, 30, 48, 72, 128, 250, 500, 720 and 1024 processors when $n_b^{(o)} > n_p^{(o)}$ with varying f_t.



(b) Mesh-size factor, c obtained for Fig 4.6a

Figure 4.6: Execution times and mesh-size factor c obtained for $n_b^{(o)} > n_p^{(o)}$

4.6a.

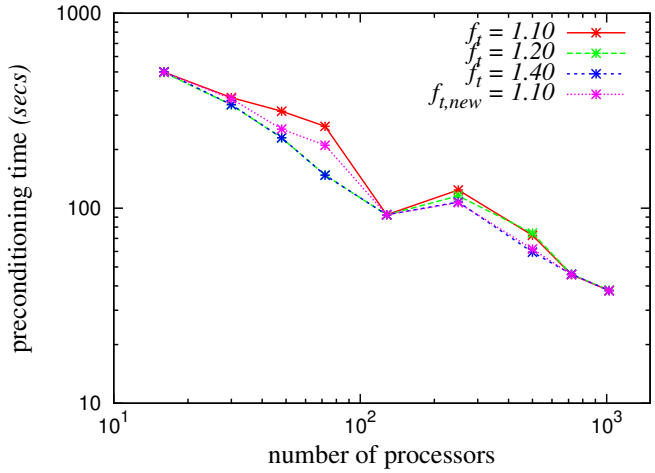
In [23], Hicken and Zingg showed that the number of Krylov iterations required to solve the linear system at each inexact-Newton iteration of the Newton-Krylov algorithm presented using the approximate Schur preconditioning technique is independent of the number of processors, but dependent on the size of the problem. Therefore for any two or more problems with the same measure of c but different f_t and number of processors,

the total Krylov iterations required to solve the problem remains the same. For instance, in Figure 4.6b, for processor numbers 72 and 250 and f_t set to 1.10 and 1.40 respectively, the problem size is the same with a c of 1.08 and we record approximately the same number of Krylov iterations. The value of c begins to grow steadily as we increase the number of processors due to the small number of blocks started with, which in turn leads to increased Krylov iterations and increased computational time.

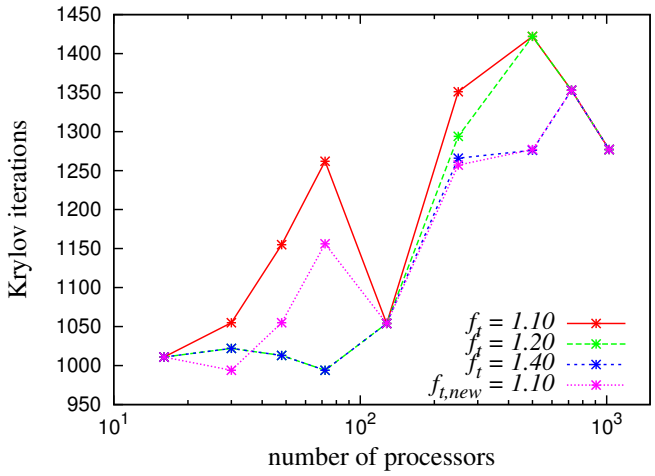
To improve the parallel performance for homogeneous multi-block grids that use the load-balancing tool, a new exit criterion is introduced for the main load-balancing loop. Since for homogeneous grids a small f_t ensures a good static load balance but does not necessarily improve parallel performance in terms of overall CPU time, the strict enforcement of f_t is relaxed by allowing it to operate only when a blocks-to-processors ratio (b_r) constraint has not been met.

This new constraint is set by the user as well. Once the blocks-to-processors ratio for an iteration is equal or greater than b_r, the main load-balancing iteration loop is exited even if f_t has not been met. The lines in Figures 4.6a, 4.7a and 4.7b indicated with "*new*" show the effect of introducing this new constraint. We show results only for f_t = 1.10, b_r = 2.0 and processors 30, 48, 72, 250, 500. The processors selected had b_r greater than 2.0 (two times more blocks than processors) after the final iterations for the initial runs without the introduction of b_r. Enforcing the load-balance threshold constraint while exercising control over the number of blocks ensures a fairly good load-balance for smaller thresholds while reducing the number of block edge-cuts as well.

Table 4.2 shows the percentage reductions obtained for overall CPU time, preconditioning time and Krylov iterations for the large number of processors which were of interest in this scaling study in comparison to f_t=1.10 in Figures 4.6 and 4.7. Although increasing f_t can be used to achieve the results in Figures 4.6, 4.7 and Table 4.2 compared to introducing b_r, it is preferable to use b_r since it gives the user a direct control on the number of blocks.



(a) Preconditioning time obtained for Fig 4.8



(b) Krylov iterations obtained for Fig 4.6a

Figure 4.7: Preconditioning time and Krylov iterations obtained for $n_b^{(o)} > n_p^{(o)}$

Processors	CPU time	Preconditioner time	Krylov iterations
72	12%	20%	8%
250	5%	13%	7%
500	6%	15%	10%

Table 4.2: Percentage reductions obtained after the introduction of b_r=2.0 relative to f_t=1.10 in Figures 4.6 and 4.7

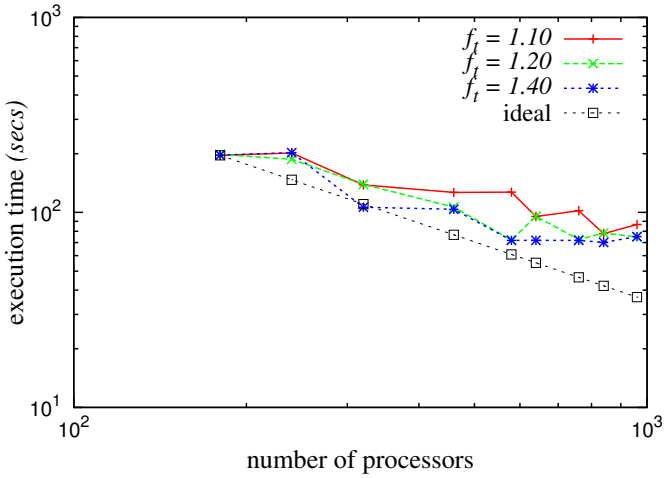


Figure 4.8: Execution times obtained for 180, 240, 320, 460, 580, 640, 760, 840 and 960 processors when $n_b^{(o)} < n_p^{(o)}$ with varying f_t.

4.2.2 Case II : $n_b^{(o)} < n_p^{(o)}$

For this case the number of processors exceeds the number of blocks at the starting point. As shown in Section 4.2.1 for $n_b^{(o)} > n_p^{(o)}$ the mesh-size factor does not increase dramatically until $n_b^{(o)} < n_p^{(o)}$. For a case that starts with $n_b^{(o)} < n_p^{(o)}$ there is an increase in the mesh-size factor immediately at the start of the scaling study. Figure 4.8 shows the scaling performance for this case. In the previous case, a perfect static load balance was obtained for the smallest number (16) of processors without any block splitting. In the present case, some block splitting was required for the smallest number (180) of processors. The insight gained from this case is that, as we keep adding processors to a simulation, the case with the largest number of blocks to start with will tend to scale better as the processor number is increased. This observation is evident in Figures 4.2 and 4.3, where the extrapolation based on 180 partitions from 332 blocks shows better relative efficiencies for large numbers of processors than observed for efficiencies based on 16 partitions from 64 blocks.

Properties	ONERA M6	CRM
Blocks	128	569
Nodes	6,470,640	10,132,263
$\text{block}_{\text{min}}$	$13 \times 26 \times 26$	$11 \times 11 \times 15$
$\text{block}_{\text{max}}$	$45 \times 55 \times 60$	$44 \times 41 \times 43$
Equations	Euler	RANS
α	3.06°	2.356°
Re	-	5×10^6
M	0.699	0.85

Table 4.3: Mesh properties and flow conditions for the heterogeneous cases

4.3 Heterogeneous Multi-block Grids

We refer to grids with different block sizes as heterogeneous multi-block grids. The results for these cases were obtained with two different grids. The first is a grid around the ONERA M6 wing, the second is a grid around the CRM wing-body geometry from DPW-IV [55]. For the ONERA M6 wing an inviscid solver is used, while for the CRM a RANS solver is used. These two cases speak to the primary thrust of this project, which is to develop a tool to help reduce turnaround times for multi-block CFD grids with different block sizes. Table 4.3 shows the grid properties and flow conditions used for both cases. Figure 4.9 shows the C_p distribution on the CRM wing-body geometry.

4.3.1 ONERA M6 Wing

In this case a medium-size multi-block grid for the ONERA M6 wing is used. The ratio of the largest block to the smallest block is ~ 17. The case is first run without load balancing (i.e. on 128 processors). Figures 4.10a, 4.10b, and 4.10c show the parallel performance properties obtained with the approximate-Schur preconditioner. In Figure

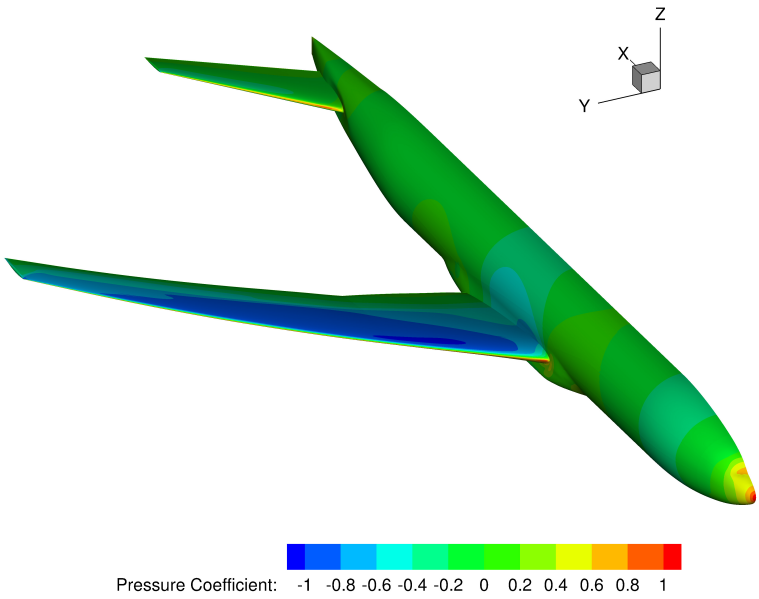
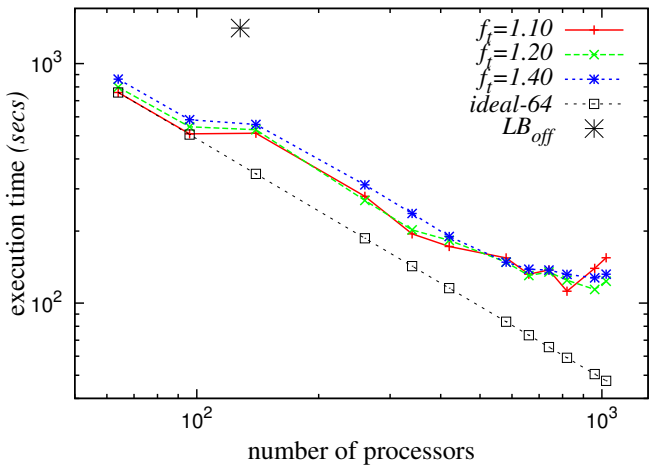
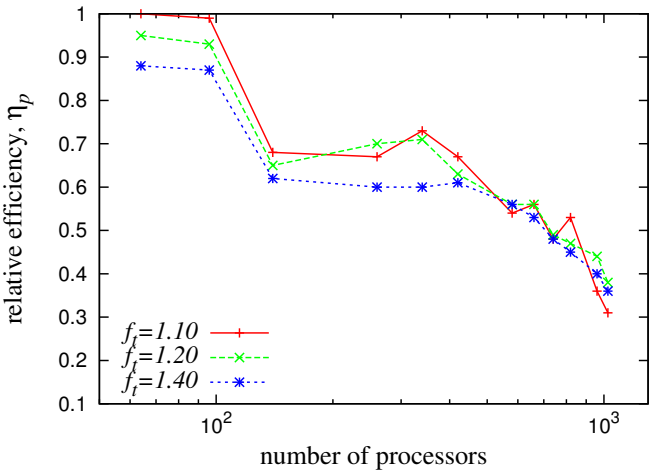


Figure 4.9: Coefficient of pressure distribution on the Common Research Model wing-geometry using 560 processors with $f_t = 1.40$

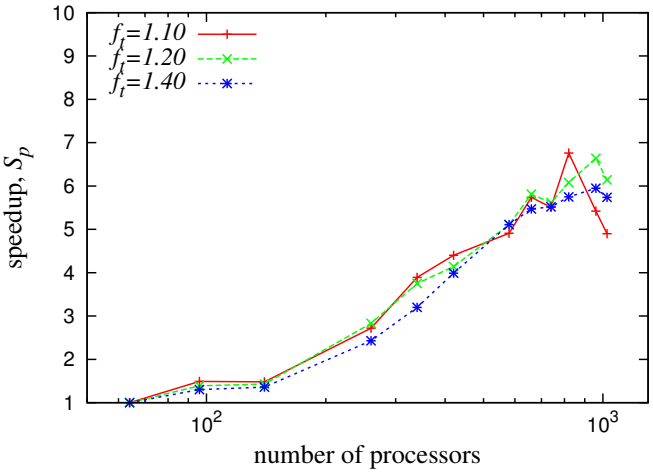
4.10a, the point (LB_{off}) shown with the * symbol is the execution time without load balancing. Up to about 700 processors we are able to recover 50% efficiency based on the ideal time extrapolation. Beyond 700 processors we are unable to see much speedup. Our ability to improve the parallel performance for the algorithm diminishes as the number of processors is increased well beyond the initial number of blocks. Since the ratio of the initial number of blocks to the number of processors becomes small in such cases, the number of block edge-cuts increases, thereby introducing additional nodes leading to a larger Schur complement system which impacts the performance of the algorithm. As we increase the number of processors, the average workload per processor also becomes small and hence communication overhead becomes significant ($\sim 5\% - 10\%$ of computational time). We also expect better performance from processor numbers that are closer to the number (128) of blocks started with. For example from 140 to 500 processors the minimum efficiency was $\sim 60\%$ for the $f_t = 1.40$ line (i.e. the worst performing line), while



(a) Computational time



(b) Parallel efficiency



(c) Parallel Speedup

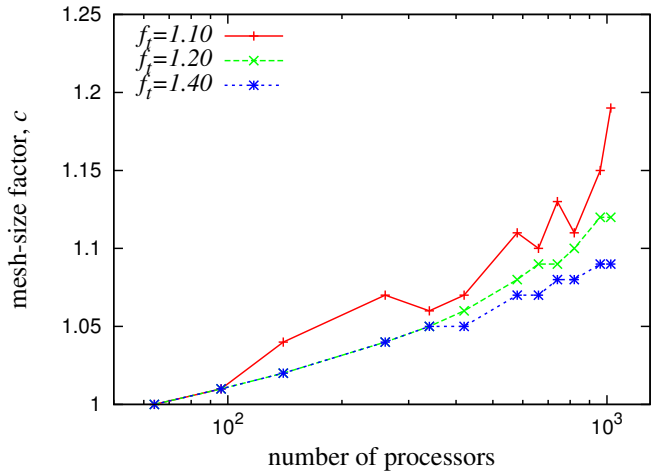
Figure 4.10: Parallel performance properties for the heterogeneous multi-block grid for the ONERA M6 wing

sharp increases in speedup are also observed between 140 and 500 processors. For this problem the smaller load-balance thresholds (i.e. $f_t = 1.10$ and $f_t = 1.20$) record better turnaround times than $f_t = 1.40$ up to about 700 processors. Overall, a load-balance factor of 1.20 improves turnaround time compared to 1.40 and 1.10.

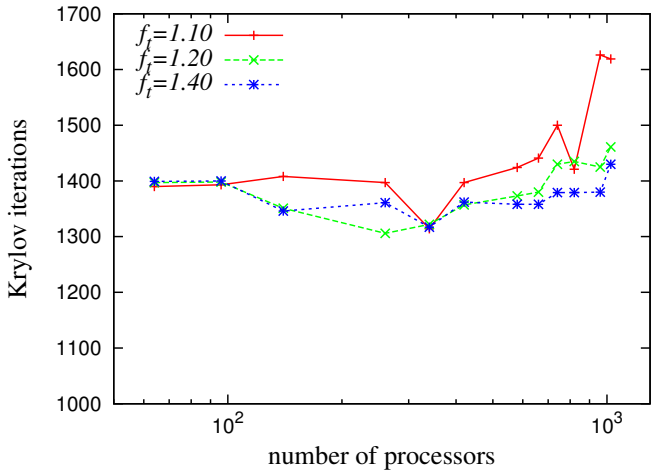
Figures 4.11a, 4.11b and 4.11c show the mesh size factor, Krylov iterations and preconditioned times observed. The conclusions reached on the scaling studies for the homogeneous blocks using the Euler equations remain fairly valid for this case as well. Using the approximate Schur preconditioning, for problems that have the same mesh size factors the same number of Krylov iterations are required to fully converge the flow solution. As we increase the number of processors, most of the computational time is spent on preconditioning since a considerable number of interior nodes are converted to interface nodes during block splitting.

Figures 4.12a, 4.12b and 4.12c compare the Schwarz and Schur preconditioning techniques for the heterogeneous multi-block grid for the ONERA M6 wing. Similar to the homogeneous case, the turnaround times recorded for both preconditioning techniques follow the same trend. This case started with 128 blocks and considering the heterogeneous nature of the mesh, the number of block splittings required to enforce the load-balance constraints considered was fewer. Hence, the Schur technique gives a slightly lower turnaround time compared to the Schwarz technique from 100 to 800 processors. From 60 to 100 processors, the Schur cases record a slightly higher execution time because the ratio of the number of blocks to the number of processors is high, which leads to a bigger local Schur complement system. Beyond 800 processors the number of edge-cuts increases, which has an adverse effect on the Schur technique compared to the Schwarz technique.

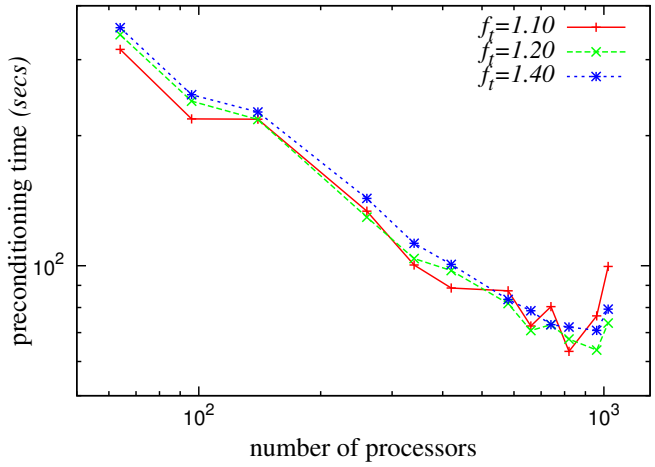
In Figures 4.13a and 4.13b we show the values computed for C_L and C_D for the different cases considered. For every case, the multi-block mesh is slightly modified. This modification is a result of block splitting, which introduces new blocks. Since we



(a) Mesh-size factor

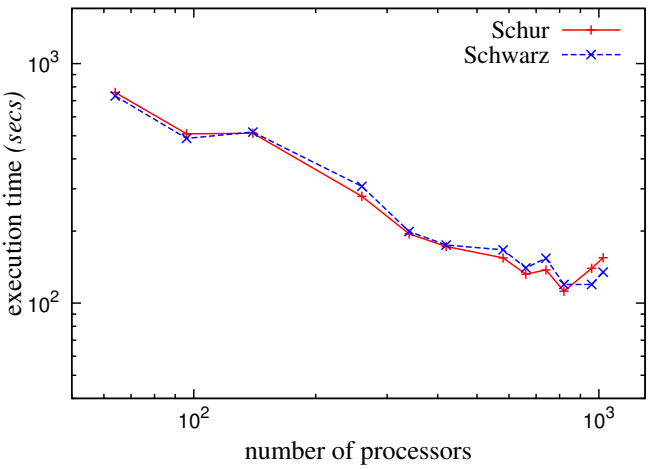


(b) Krylov iterations

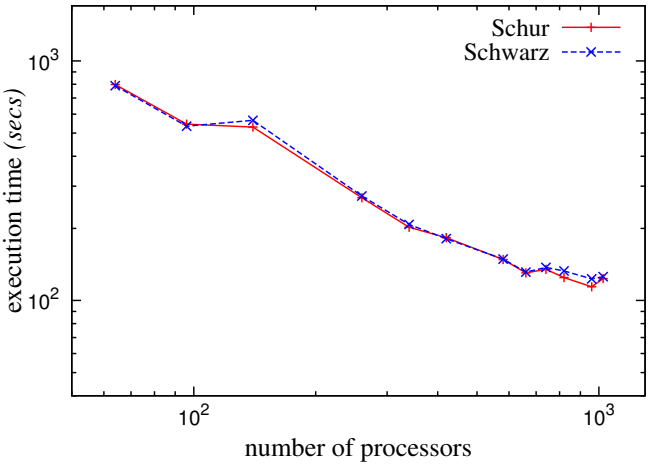


(c) Preconditioning time

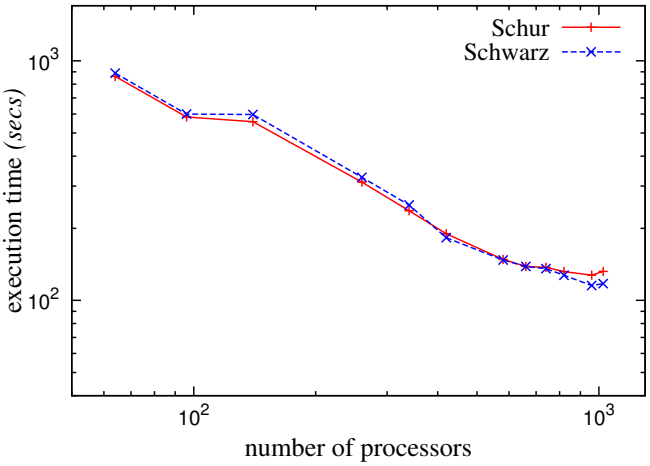
Figure 4.11: Newton-Krylov algorithm performance for the heterogeneous multi-block grid for the ONERA M6 wing



(a) Schur vs. Schwarz preconditioning for $f_t = 1.10$

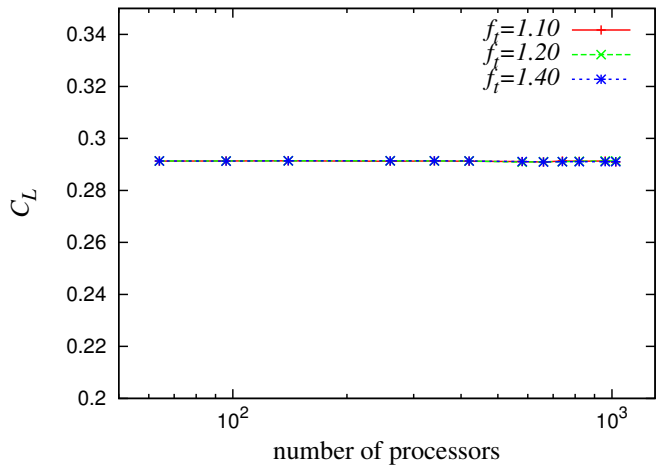


(b) Schur vs. Schwarz preconditioning for $f_t = 1.20$

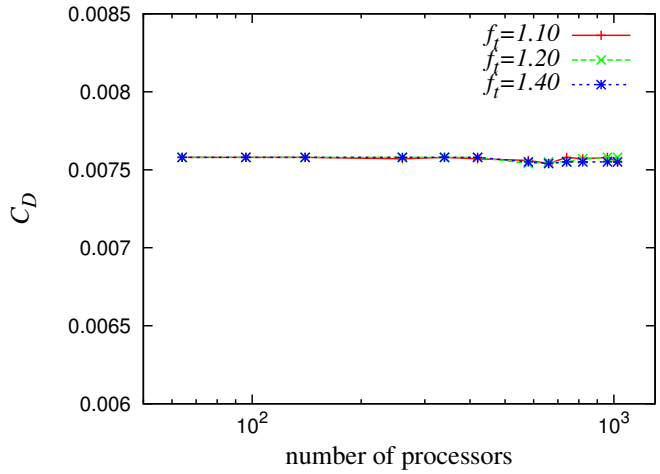


(c) Schur vs. Schwarz preconditioning for $f_t = 1.40$

Figure 4.12: A comparison of the Schur and Schwarz preconditioning techniques for the heterogeneous multi-block grid for the ONERA M6 wing



(a) C_L values with varying number of processors



(b) C_D values with varying number of processors

Figure 4.13: C_L and C_D values for the heterogeneous multi-block grid for the ONERA M6 wing

apply different discretization schemes at the interface and interior nodes it is important to know if the functionals calculated remain similar. For the case of C_L, we observe that the functional remains the same for all the cases considered. In the case of C_D, we observe a $\sim 0.4\%$ variation between functional values on processor numbers before and after 600.

Table 4.4 shows the speedup observed using the case without load balancing as a basis. Using a smaller number of processors (i.e. 64, 96) with the load-balancing tool we

record considerable speedups compared to using 128 processors without load-balancing, which confirms that for heterogeneous blocks considerable time savings can be gained making use of the load balancing tool. The improvements in turnaround times due to the load balancing tool are promising for the optimization framework for which the Newton-Krylov solver was developed. For aerodynamic shape optimization problems that use heterogeneous grids we can use a smaller number of processors and still obtain lower turnaround times during the computation of objective functions. It is also impractical to use homogeneous multi-block grids for all flow simulations especially for simulations involving complex geometries, and as such a load-balancing tool will play an important role in ensuring that maximum benefit is derived from a parallel computing platform. In addition, simulations involving third-party meshes can be run more efficiently on massively-parallel computers with the aid of the load-balancing tool.

4.3.2 CRM Wing-body Configuration

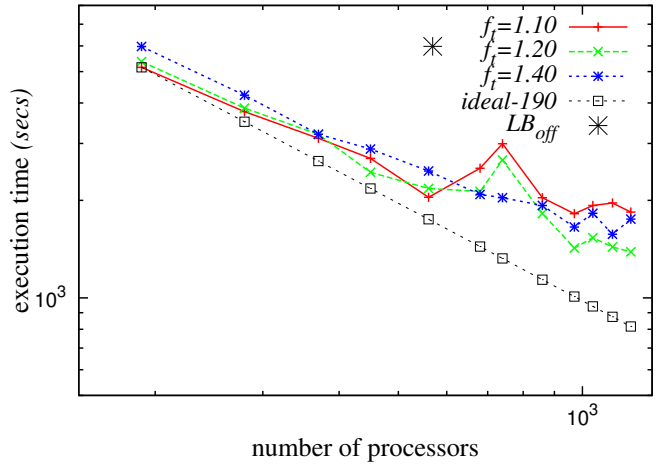
We use a medium-size heterogeneous multi-block grid with 569 blocks for the CRM wing-body geometry, solving the RANS equations. The ratio of the largest block to the smallest block is ~ 43. Without load balancing the case is run on 569 processors. The scaling study starts from 190 processors and goes up to 1200 processors. Figures 4.14a, 4.14b and 4.14c show the parallel performance properties observed. The * symbol in Figure 4.14a is the execution time without (LB_{off}) load balancing. As observed in previous cases, the algorithm performance drops as we increase the number of processors. Between 190 and 560 (which is approximately equal to the number of blocks started with) processors the efficiency of the algorithm for the worst performing case is $\sim 70\%$, which indicates a good scaling for the Newton-Krylov algorithm in that region. The jump in execution time between 560 and 860 for $f_t = 1.10$ and $f_t = 1.20$ is due to the sharp increase in the mesh-size factor in that region. Hence, the speedup and efficiency between 560 and 860 processors also drops. For this case we record better relative efficiencies compared

Processors	f_t=1.10	f_t=1.20	f_t=1.40
64	1.9	1.8	1.6
96	2.8	2.6	2.4
140	2.7	2.7	2.5
260	5.0	5.2	4.5
340	7.2	7.0	5.9
420	8.2	7.7	7.4
580	9.1	9.5	9.5
660	10.7	10.8	10.2
740	10.2	10.4	10.2
820	12.5	11.3	10.7
960	10.1	12.3	11.0
1024	9.1	11.4	10.6

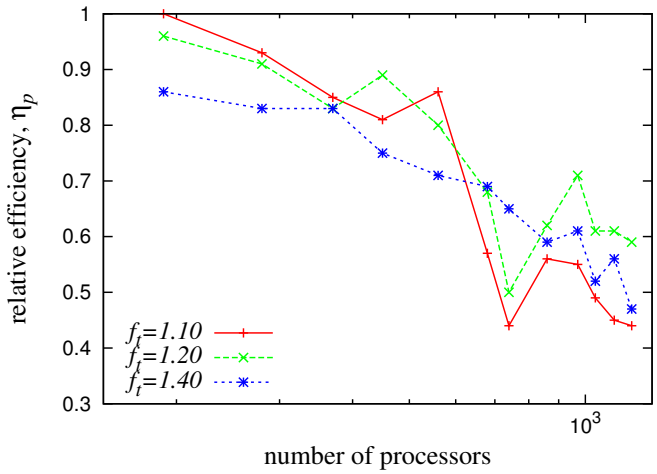
Table 4.4: Speedup relative to 128 processors without load balancing

to the Euler case. This is a result of the large number of blocks started with. The size of the mesh used makes is it possible to run on a large number of processors, but only processor numbers up to 1200 were considered. Similar to the Euler case, $f_t = 1.20$ gives a better results compared to the other load-balance thresholds especially as the number of processors is increased.

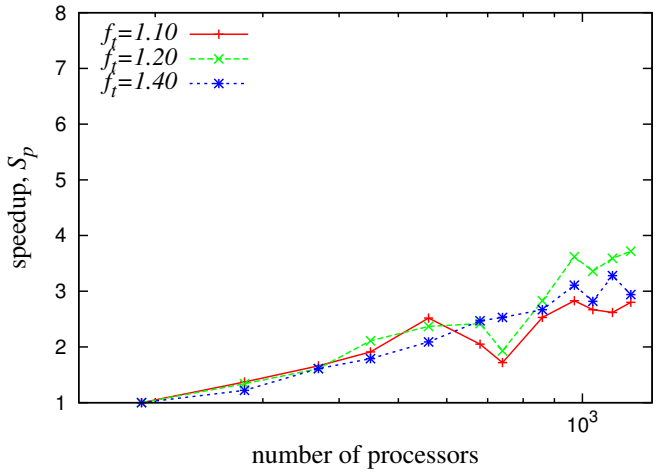
In Figures 4.15a, 4.15b and 4.15c we show the mesh size factor, Krylov iterations and preconditioning times observed. The mesh size factor plateau beyond 600 processors for $f_t = 1.10$ shows a problem with the same size as the processors increase, since the same mesh meets the load-balancing constraint set for the number of processors used. Unlike the Euler case, the same number of Krylov iterations are not recorded for problems with the same size when the Schur preconditioning technique is used. This is likely because



(a) Computational time



(b) Parallel efficiency



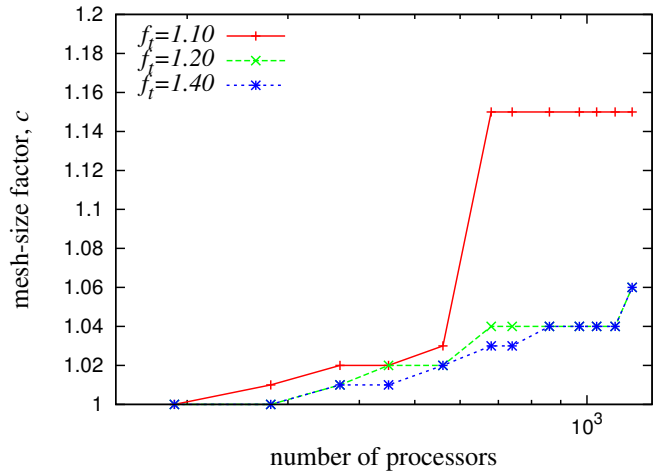
(c) Parallel Speedup

Figure 4.14: Parallel performance properties for the heterogeneous multi-block grid for the CRM wing-body geometry

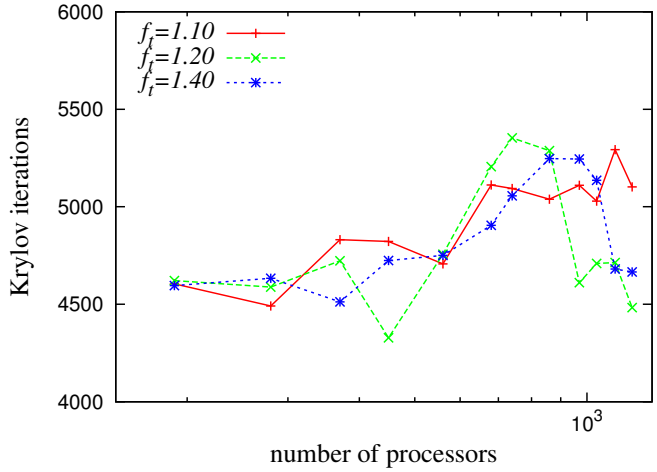
the difference in magnitudes of the turbulent and mean flow quantities leads to different scaling values being applied to the two quantities, thereby impacting the convergence of our Krylov solver. The trend observed in the time spent on preconditioning is similar to that observed in the previous cases. As we increase the number of processors, most ($\sim 50 - 70\%$) of our computational time is spent on preconditioning using the Schur technique.

In the case of C_L and C_D values (see Figure 4.16), we find that both functionals remain almost the same for all the cases considered. The computed C_L values for the processors considered are in the range of $0.50(\pm 0.001)$ while the C_D values are in the range of $0.0273(\pm 0.0005)$.

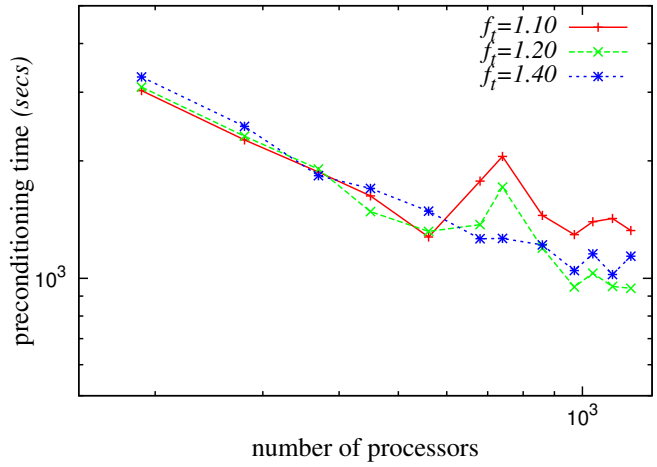
Table 4.5 shows the speedup observed based on the case (569 processors) without load balancing. As in the case of the Euler equations solution on a heterogeneous mesh, we record considerable time savings using the load-balancing tool and fewer than 569 processors. We are able to obtain $\sim 40\%$ improvement in turnaround time with a 50% reduction in the number of processors using the load-balancing tool.



(a) Mesh-size factor

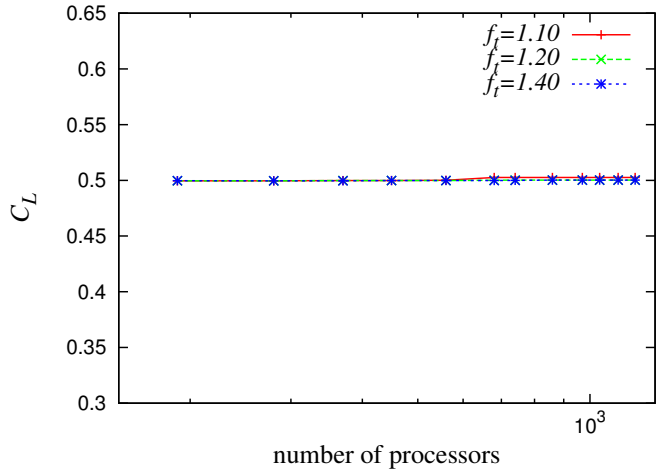


(b) Krylov iterations

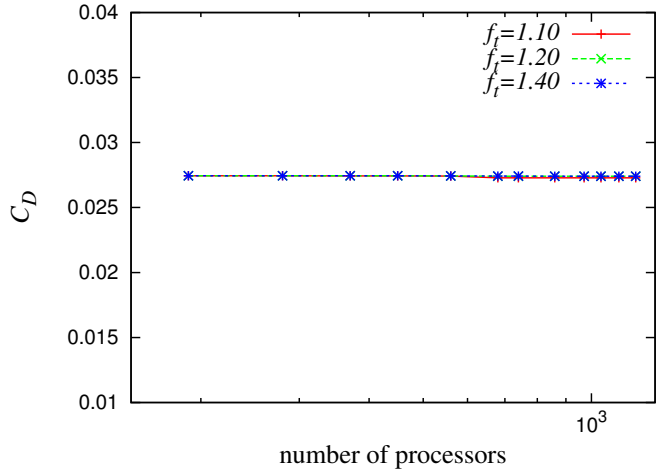


(c) Preconditioning time

Figure 4.15: Newton-Krylov algorithm performance for the heterogeneous multi-block grid for the CRM wing-body geometry



(a) C_L values with varying number of processors



(b) C_D values with varying number of processors

Figure 4.16: C_L and C_D values for the heterogeneous multi-block grid for the CRM wing-body geometry

Processors	f_t=1.10	f_t=1.20	f_t=1.40
190	1.2	1.1	1.0
280	1.6	1.6	1.4
370	1.9	1.9	1.9
450	2.2	2.4	2.1
560	2.9	2.7	2.4
680	2.4	2.8	2.9
740	2.0	2.2	2.9
860	2.9	3.3	3.1
970	3.3	4.2	3.6
1040	3.1	3.9	3.3
1120	3.0	4.2	3.8
1200	3.2	4.3	3.4

Table 4.5: Speedup relative to 569 processors without load balancing

Chapter 5

Conclusions and Recommendations

This chapter summarizes the conclusions drawn from the results presented in Chapters 3 and 4. I also review the contributions and insights gained from using the load-balancing tool within the Newton-Krylov framework. Recommendations are also made as regards improvements that can be made to the tool and the Newton-Krylov algorithm.

5.1 Conclusions

In this work, a load-balancing tool based on a coarse-grain parallelism approach has been presented.

The load-balance threshold factor constraint was introduced in order to achieve our objective of minimizing the ratio of the maximum to the average workload per processor. Depending on the type of multi-block grid, two additional constraints were introduced. For homogeneous blocks, a blocks to processor ratio constraint can be used to restrict block splitting. This constraint improves the preconditioning time, Krylov iterations and overall CPU time. For heterogeneous blocks, a block size constraint is introduced to prevent smaller blocks from being split until the largest blocks are about the same size as the smaller ones.

For the Euler solutions on both homogeneous and heterogeneous multi-block grids, the

number of Krylov iterations required to solve a problem using the Schur preconditioning was only dependent on problem size, as concluded in [23]. Independent of the workload threshold and the number of processors, as long as two or more problems have the same mesh-size factors, the number of Krylov iterations remain the same.

For problems with the same size, the number of Krylov iterations observed for the RANS solutions on the heterogeneous multi-block varied considerably compared to the Euler case. This discrepancy is a result of the different scalings applied to the mean-flow quantities and the turbulent model.

In terms of parallel performance, as the number of processors is increased using the load-balancing tool, the Newton-Krylov algorithm's relative efficiency and speedup drops. In all the results presented for the parallel scaling studies, as the number of processors increases, a major part of the execution time is spent on preconditioning. This is as a result of the increase in the ratio of interface nodes to interior nodes. Since the Schur complement system consists only of the interface nodes, an increase in the number of these nodes leads to an increase in the time taken to obtain approximate solutions to the interface nodes.

It is also very important to know the effects domain decomposition coupled with the SBP-SAT treatment at the block interfaces have on functional values calculated after flow computations. As more processors are added, block splitting increases and an interior node is converted to two interface nodes for each splitting. The added interfaces lead to slight variations in functional values as the number of processors is increased. Since the variations observed are not significant, using the load balancing tool within an optimization framework will not introduce significant errors in the evaluation of functionals required to compute objective functions.

The most important contribution to our Newton-Krylov framework using a load-balancing tool is the improvement in turnaround times obtained for heterogeneous grids. For both Euler and RANS heterogeneous cases turnaround times were improved by $\sim$50%

with a ∼50% reduction in the number of processors without load balancing.

In addition, a load-balance threshold of 1.20 was found to be a good choice for all cases considered since it results in a smaller number of block edge-cuts compared to 1.10 and a smaller load imbalance compared to 1.40. A load balance threshold of 1.10 leads a good load balance but the number of block edge-cuts associated with it reduces the performance of the Newton-Krylov algorithm especially for a large number of processors. A load-balance threshold of 1.40 leads to a high load imbalance especially for a smaller number of processors.

Finally, the Newton-Krylov algorithm can now handle any arbitrary grid on any number of processors and still obtain a good static load balance based on the threshold set by the user. This addition to our Newton-Krylov algorithm should also ease processor and load balance requirements during mesh generation. Users of the flow solver can now focus on geometry and accuracy requirements during grid generation.

5.2 Recommendations

This work focused on the flow solver component of our optimization framework. For optimization problems involving heterogeneous multi-block grids, our control mesh which closely mimics the exact mesh will be heterogeneous blocks as well. Therefore, our adjoint solver for optimization will also benefit from a load balancing tool in the case of heterogeneous blocks.

Currently, our B-spline volume meshes only allow for one-to-one mesh interfacing. In the case of heterogeneous meshes and splitting of blocks using REB, blocks can have two-to-one interfaces. Therefore, in order to obtain the correct heterogeneous meshes after mesh movement, the B-spline volumes implementation needs to be modified to accommodate different block interface types.

Finally, further work into dynamic load balancing will be very useful for a solution

process that makes use of mesh adaptation. Since for such purposes the mesh at each solution stage is adapted to the changes in the flow solution, static load balancing tools will be insufficient in addressing the load balancing requirements associated with such problems. Therefore any work into adaptive mesh refinement for our Newton-Krylov solution process should make considerations for dynamic load balancing as well. In addition, investigation can be conducted into improving workload distribution for the static tool based on a weighting system considering nearest neighbours as well as overall workload in order to meet the load-balancing constraints set.

References

[1] The environmental effects of civil aircraft in flight. Special report, Royal Commission on Environmental Pollution, 2002.

[2] A. Àlund, P. Lötstedt, and M. Sillén. Parallel single and multigrid solution of industrial compressible flow problems. *Computers and Fluids*, 26:775–791, 1997.

[3] J. D. Anderson. *Fundamentals of Aerodynamics*. McGraw Hill, 1984.

[4] J. D. Anderson. *Introduction to Flight*. McGraw-Hill, 5th edition, 2005.

[5] K. P. Apponsah and D. W. Zingg. A load balancing tool for structured multi-block CFD applications. In *20th Annual Conference CFD Society of Canada*, Canmore, AB, May 2012.

[6] M. H. Carpenter, D. Gottlieb, and S. Abarbanel. Time-stable boundary conditions for finite difference schemes solving hyperbolic systems: Methodology and application to high-order compact schemes. *Journal of Computational Physics*, 111(2):220–236, 1994.

[7] M. H. Carpenter and J. Nordström. Boundary and interface conditions for high order finite difference methods applied to the Euler and Navier-Stokes equations. *Journal of Computational Physics*, 148, 1999.

[8] M. H. Carpenter, J. Nordström, and D. Gottlieb. A stable and conservative interface treatment of arbitrary spatial accuracy. *Journal of Computational Physics*, 148(2):341–356, 1999.

[9] T. F. Chan and K. R. Jackson. Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. *SIAM Journal of Scientific and Statistical Computing*, 5:535–542, 1984.

[10] M. R. J. Charest. *Numerical Modelling of Sooting Laminar Diffusion Flames at Elevated Pressures and Microgravity*. PhD thesis, University of Toronto, 2009.

[11] T. T. Chisholm and D. W. Zingg. A Jacobian-free Newton-Krylov algorithm for compressible turbulent flows. *Journal of Computational Physics*, 59:232–246, 1985.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[13] L. T. Diosady and D. L. Darmofal. Massively parallel solution techniques for high-order finite-element discetizations in CFD. *World Scientific Review Volume*, 2010.

[14] M. J. Djomehri and R. Biswas. Performance enhancement stratagies for multi-block overset grid CFD applications. Technical Report NAS-03-011, NASA, 2003.

[15] E. W. Evans, S. P. Johnson, P. F. Legget, and M. Cross. Automatic and effective multi-dimensional parallelisation of structured mesh based codes. *Parallel Computing*, 26:677–703, 2000.

[16] N. Gourdain, L. Gicquel, M. Montagnac, O. Vermorel, M. Gazaix, G. Staffelbach, M. Garcia, J-F Boussuge, and T. Poinsot. High performance parallel computing of flows in complex geometries: I. Methods. *Computational Science and Discovery*, 2, 2009.

[17] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, Cambridge, Mass, second edition, 1999.

[18] C. P. T. Groth and S. A. Northrup. Parallel implicit adaptive mesh refinement scheme for body-fitted multi-block mesh. AIAA Paper 2005-5333, Toronto, Ontario, 2005.

[19] B. Gustafsson, H. O. Kreiss, and A. Sundström. Stability theory of difference approximations for mixed initial boundary value problems. *Math and Computers*, 26(119), 1972.

[20] J. Häuser, P. Eiseman, P. Y. Xia, and Z. Cheng. Parallel multiblock structured grids. In *Handbook on Grid Generation*, chapter 12. CRC Press LLC, 1999.

[21] J. E. Hicken. *Efficient Algorithms for Future Aircraft Design: Contributions to Aerodynamic Shape Optimization*. PhD thesis, University of Toronto, 2009.

[22] J. E. Hicken, M. Osusky, and D. W. Zingg. Comparison of parallel preconditioners for a Newton-Krylov flow solver. In *6th International Conference on Computational Fluid Dynamics*, St. Petersburg, Russia, 2010.

[23] J. E. Hicken and D. W. Zingg. Parallel Newton-Krylov solver for the Euler equations discretized using simultaneous-approximation terms. *AIAA Journal*, 46(11):2773–2786, 2008.

[24] C. Hirsch. *Numerical Computation of Internal and External Flows*, volume 2. Wiley and Sons, 1984.

[25] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM Journal for Scientific Computing*, 22(6):2194–2215, 2001.

[26] K. Sermeus K., E. Laurendeau, and F. Parpia. Parallelization and performance optimization of Bombardier multi-block structured Navier-Stokes solver on IBM eserver cluster 1600. AIAA Paper 2007-1109, American Institute of Aeronautics and Astronautics, Reno, Nevada, 2007.

[27] G. Karypis and V. Kumar. Parallel threshold-based ILU factorization. Technical Report No. 96-061, University of Minnesota, Department of Computer Science, 1998.

[28] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computinf fill-reducing orderings of sparse matrices, version 4.0. Technical report, University of Minnesota, Department of Computer Science and Engineering, 1999.

[29] G. Karypis, K. Schloegal, and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical report, University of Minnesota, Department of Computer Science and Engineering, 1997.

[30] D. E. Keyes. Aerodynamic applications of Newton-Krylov-Schwarz solvers. In M. Deshpande, S. Desai, and R. Narasimba, editors, *Proceedings of the 14th International Conference on Numerical Methods in Fluid Dynamics*, pages 1–20, New York, 1995. Springer.

[31] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: A survey of appraches and applications. *Journal of Computational Physics*, 193:357–397, 2004.

[32] E. M. Lee-Rausch, N. T. Frink, D. J. Mavriplis, R. D. Rausch, and W. E. Milholen. Drag prediction on a DLR-F6 transport configuration using unstructured grid solvers. AIAA Paper 2004-0554, American Institute of Aeronautics and Astronautics, Reno, Nevada, 2004.

[33] T. M. Leung. *A Newton-Krylov Approach to Aerodynamic Shape Optimization in Three Dimensions*. PhD thesis, University of Toronto, 2009.

[34] H. Lomax, T. H. Pulliam, and D. W. Zingg. *Fundamentals of Computational Fluid Dynamics*. Springer-Verlag, 2001.

[35] G. May, E. van der Weide, A. Jameson, and L. Martinelli. Drag prediction of the DLR-F6 configuration. AIAA Paper 2004-0396, American Institute of Aeronautics and Astronautics, Orlando, Florida, 2004.

[36] J. Mushtaq and A. J. Davies. A multi-block solution process for the equations of fluid dynamics on a network of transputers. *Advances in Engineering Software*, 26(2):133–149, 1996.

[37] M. Nemec. *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach*. PhD thesis, University of Toronto, 2003.

[38] M. Nemec and D. W. Zingg. A Newton-Krylov algorithm for aerodynamic design using the navier-stokes equations. *AIAA Journal*, 40(6):1146–1154, 2002.

[39] J. Nichols and D. W. Zingg. A three-dimensional multi-block Newton-Krylov flow solver for the Euler equations. AIAA Paper 2005-5230, American Institute of Aeronautics and Astronautics, 2005.

[40] G. Norris. Next-generation aircraft: Building blocks. Aviation Week and Space Technology, 2011. Pages: 36-42.

[41] M. Osusky, J. E. Hicken, and D. W. Zingg. A parallel Newton-Krylov-Schur flow solver for the Navier-Stokes equations using SBP-SAT approach. AIAA Paper 2010-116, American Institute of Aeronautics and Astronautics, Orlando, Florida, 2010.

[42] M. Osusky and D. W. Zingg. A parallel Newton-Krylov flow solver for the three dimensional Reynolds-averaged Navier-Stokes equations. In *20th Annual Conference CFD Society of Canada*, Canmore, AB, May 2012.

[43] J. E. Penner et al. *Aviation and the Global Atmosphere: A Special Report of IPCC Working Groups I and III in Collaboration with the Scientific Assessment Panel to the Montreal Protocol on Substances that Deplete the Ozone Layer*. Cambridge University Press, New York, 1999.

[44] A. Pueyo and D. W. Zingg. Efficient Newton-Krylov solver for aerodynamic computations. *AIAA Journal*, 36:1991–1997, 1998.

[45] T. H. Pulliam. Solution methods in computational fluid dynamics. Technical report, Lecture Notes for the von Kármán Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Rhode-St-Genese, Belgium, 1985.

[46] T. H. Pulliam. Efficient solution methods for the Navier-Stokes equations. Technical report, Lecture Notes for the von Kármán Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Brussels, Belgium, January 1986.

[47] T. H. Pulliam and J. Steger. Implicit finite-difference simulations of three-dimensional compressible flow. *AIAA Journal*, 18(2):159–167, 1980.

[48] T. H. Pulliam and D. W. Zingg. *Foundations of Computational Fluid Dynamics*. 2011.

[49] J. Rantakoko. Partioning strategies for structured multi-block grids. *Parallel Computing*, 26:1661–1680, 2000.

[50] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, second edition, 2003.

[51] Y. Saad and M. Sosonkina. Distributed Schur complement preconditioning. *SIAM Journal for Scientific Computing*, 21(4):1337–1356, 1999.

[52] M. L. Sawley and J. K. Tegnér. A comparison of parallel programming models for multi-block flow computations. *Journal of Computational Physics*, 122:280–290, 1995.

[53] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. AIAA Paper 92-0439, American Institute of Aeronautics and Astronautics, January 1992.

[54] J. C. Vassberg. A unified baseline grid about the common research model wing-body for the fifth AIAA CFD drag prediction workshop. AIAA Paper 2011-3508, American Institute of Aeronautics and Astronautics, Honolulu, Hawaii, 2011.

[55] J. C. Vassberg, E. N. Tinoco, M. Mani, B. Rider, T. Zickuhr, D. W. Levy, O. P. Brodersen, B. Eisfeld, S. Crippa, R. A. Wahls, J. H. Morrison, D. Mavriplis, and M. Murayama. Summary of the third AIAA CFD drag prediction workshop. AIAA Paper 2010-4547, American Institute of Aeronautics and Astronautics, Illinois, Chicago, 2010.

[56] D. C. Wilcox. *Turbulence Modelling for CFD*, volume 3. DCW Industries, 2010.

[57] A. Ytterström. A tool for partitioning structured multi-block meshes for parallel computational mechanics. *International Journal of Supercomputer Application*, 11(4):336–343, 1997.

[58] A. Ytterström. *Parallel Computing for Applications in Aeronautical CFD*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, 2001.