MULTIDISCIPLINARY AIRCRAFT DESIGN OPTIMISATION USING PARALLEL COMPUTER ARCHITECTURES

C. Peebles, C. Bil
School of Aerospace, Mechanical and Manufacturing Engineering
RMIT University, Melbourne Australia.
Keywords: *Aerospace Engineering, Design Optimisation, Parallel Computing, Message-Passing*

Abstract

ADAS (Aircraft Design and Analysis Software) is a FORTRAN package that allows designers to optimise design systems consisting of numerous often-conflicting design variables, subject to given constraints and a merit function. While created with the principal aim of aircraft design applications, the software can be utilised for any scenario of multi-disciplinary optimisation. ADAS also executes in a number of modes, supplying the user with ranges of outputs that can be beneficial during many stages in the design process.

There are many software tools available to engineers and designers, which aid in the decision-making process by allowing the user to analyse the impact of certain design choices. These tools include CFD/FEM Analysis packages and CAD drawing applications, among others. With recent improvements in computer technology and parallel computer architectures, it is becoming easier and more efficient to fully integrate these tools to accelerate the design process.

ADAS has been modified for execution in a parallel environment, utilising the MPI (Message-Passing Interface) standard to handle inter-processor communications and load sharing. This allows ADAS to execute on any number of processors, and allocate workloads in the most efficient manner at runtime, dependent on the characteristics of the design problem being solved, and the mode of execution utilised.

Excellent results have been obtained using ADAS on the VPAC Facilities in Melbourne Australia, showing that systems incorporating many design variables can be optimised without an increase in response time. Tests were performed using several simulations of increasing design complexity, with speedup results being in agreement, showing that significant benefits can be gained by using this software for very complex designs.

1 Introduction

The design process is an iterative process and consists of many stages in which the design is further refined. In aircraft design, the first stage is generally referred to as conceptual design, where the basic aircraft configuration and layout are determined. Once the aircraft configuration is selected, more advanced calculations are made in the preliminary design stage, where individual components of the aircraft are designed and synthesised into a complete system. These two phases together are generally referred to as configuration development, and for the majority of aircraft design projects, this is as far as they will go. It has been said [8] that the principal aim of configuration development is to obtain the information required in order to decide whether the concept will be technically feasible and have satisfactory economic possibilities.

Conceptual design is the pivotal stage in aircraft design, as it determines the technological feasibility of the design, its applicability to the problem at hand, and what can sometimes be of more significance, the overall production and operational cost.

During configuration development, a team of design engineers and specialists will work together to optimise the overall design. However, there are a large number of design requirements that can influence the end product, such as transport capacity, aerial refuelling capability, take-off and climb performance, maximum range, etc. As such, compromises need to be made in order to achieve the best possible overall design. A graphical method to illustrate the need for compromise is called a matching chart.

1.1 Matching Chart

A matching chart is a graphical representation of design constraints as functions of two design variables, for example, wing loading (W/S) and thrust-to-weight ratio (T/W). Design constraints may include range performance, take-off distance, and lift coefficients for various phases of flight. An example of a typical matching chart is given in Figure 1. Each constraint line displays hatch-marks denoting the non-feasible area for that constraint. When all constraints are plotted, it produces a feasible design region from which a design point can be chosen.

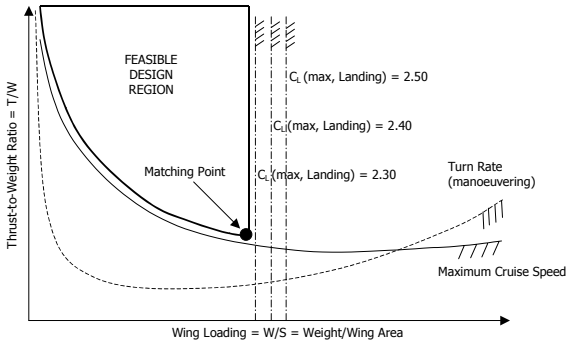

Figure 1: Example Matching Chart for Aircraft Design

The design point selection will depend on the design specification, and in particular the importance of each of the variables under consideration. For example, given the variables in Figure 1, the optimal choice is to minimise T/W, while maximising W/S. In more complicated scenarios, however, there may be a more complex objective Function such as fuel weight, maximum takeoff weight or direct operating cost.

When selecting the optimum design point, numerical algorithms can be utilised to progress from an initial set of values to an optimum design. Some of these methods are explained in the following sections.

1.2 ADS – Fortran Software for Automated Design Synthesis

ADS [2] is an open source Fortran program for general-purpose numerical optimisation. The basic functionality of the program serves to minimise an objective function subject to given design constraints, using gradient-based hill climbing methods to achieve the optimum.

The ADS software allows for use of a wide range of algorithms, incorporating different optimisation strategies, numerical methods, and search options used to determine the path to an optimum value. It is written such that it is called from a user-written program, and parameter settings determine both the design case being studied (variable values and constraint definitions), and the methods used to determine the solution.

The generality of the software means that its scope of use is not limited to design in any one field. As such, it can be used for any numerical optimisation process, however in the current project it was applied to aerospace design.

1.2.1 Gradient Calculation

The ADS software allows the option for a user-supplied program to supply gradients at certain points during execution. This is particularly useful if gradients can be determined analytically. Alternatively, ADS can approximate gradients internally using finite difference calculations. However these calculations are performed in serial and it was the objective of this project to parallelise the gradient calculation, allowing such calculations to be performed simultaneously and thereby

increasing the speedup efficiency of the parallel code.

Newton's Divided Difference is the numerical method used for gradient calculations. Due to the non-linear nature of the constraints and objective functions, a quadratic gradient approximation is applied using a central finite difference technique. This requires three data points to be determined for each function for which gradients are required. Since the central point is already known due to previous ADS calculations, this requires a further two data points for each function (per design variable) as shown in Figure 2.

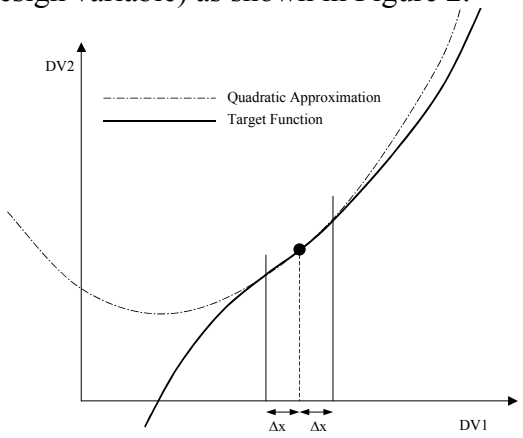

Figure 2: Quadratic Approximation of Constraint Equation.

The step size from the central point for that gradient is denoted Δx, which is set at 5% of the current value of that DV (Design Variable). Note however that this process must be performed not only for each of the constraints plus the objective, but each gradient must be separately calculated with respect to each of the design variables in turn, keeping the others constant. Thus, if the number of design variables increases, the time saving due to parallelisation also increases.

1.3 ADAS – Aircraft Design & Analysis System

ADAS [1] is a computer-based tool for conceptual aircraft design and configuration optimisation applications. ADAS is integrated with other design tools, such as CAD, and the aforementioned ADS numerical optimisation software.

ADAS is designed such that engineers and designers can modify the code to suit a particular design problem and then compile and execute it. The user-supplied code is contained in a separate subroutine called DSPROG (DeSign PROGram). DSPROG contains the necessary design calculations, including the constraint and objective functions for optimisation. To make the development of the DSPROG code more user-friendly, a library of pre-programmed routines with standard design calculations can be accessed.

For example, to generate the matching chart in Figure 1, DSPROG would contain the equations defining the constraint curves, where input files would contain names of variables and constraints, initial variable values to begin optimisation, and definitions of feasible design regions (hatch-marks) for each constraint.

1.3.1 ADAS Modes of Execution

ADAS has a number of execution modes, which can be selected to provide different levels of output:

- Design Analysis Mode, where the program simply returns constraint tolerances (how close the input design point is to the constraints). This mode is executed in a single pass, and is typically used for design-point calculation for a chosen aircraft configuration;
- Parametric Survey, where a separate input file lists numerous parameters, each with a range of values. This is similar to "Design Analysis"; however analysis is performed for each of the sets of values in the new input file. Parametric survey mode is generally used to determine the influence or sensitivity of design characteristics (dependent variables) with respect to design parameters (independent variables);
- Optimisation Mode, where the program makes numerous calls to the ADS software, in order to optimise a non-linear objective function, subject to non-linear constraints. This mode results in a single optimum design; and

- A combination of parametric survey mode and optimisation modes can also be selected.

The user defines the design problem using three files:

- *Inputfile.txt:* Formatted text file containing variable and constraint names and limits/tolerances; also includes parameters that govern the optimiser;
- *Para_input.txt:* Formatted text file containing names and values for each set of parametric data. This file is only used in Parametric Survey Mode; and
- *DSPROG.FOR:* FORTRAN routine containing design calculations. This defines the relationships of constraints to design variables, which procedures are required for calculations, external routines or programs to call upon, etc.

No other files in ADAS require modification. After these three files are defined, the program can be compiled and executed.

1.4 MPI – The Message-Passing Interface

MPI is a portable package of routines developed for message passing within a parallel computer architecture. When a program is run in parallel, typically each processor deals with variable values that are local to that processor; however, MPI can be used if values need to be shared across different processors.

There are many memory architectures in modern parallel computers. The Message-Passing Model defines each process as having local memory, but also having the ability to communicate through messages with other processes. This data transfer from the local memory of one process to that of another requires actions from both processes.

MPI has become one of the more widely used paradigms for expressing parallel algorithms. Although it has its limitations, it has several advantages, such as high performance, ease of debugging and universality. Although MPI contains a large set of commands, about 125, the majority of those are for special-purpose algorithms and for simpler parallel programs usually six standard MPI calls are sufficient.

A basic framework for a simple MPI program could look something like this:

- Start Program;
- Initialise MPI (MPI_INIT);
- Determine how may processors are running the program, and find out which process I am (MPI_COMM_SIZE, MPI_COMM_RANK);
- Send any messages (MPI_SEND);
- Receive any messages (MPI_RECV);
- Finalise MPI (MPI_FINALIZE); and
- End Program.

However, more advanced procedure calls were made during the implementation of MPI into ADAS, to perform such tasks as:

- Collective communication (broadcasting data to numerous processors at once);
- Non-blocking communication (where the processor can continue to perform work while it waits for communication to complete); and
- Rearrangement of the communicator into a 2D Cartesian Topology (used for more than one degree of parallelism).

As per the MPI standard, when an MPI parallel program is written and run, each processor executes identical code, and it is up to the programmer to include the relevant MPI calls for inter-processor communication and program control.

1.5 Design Case Study

In order to evaluate the software, a relatively simple design program was used to demonstrate an aircraft design application. The design case was to generate a matching chart, comprising two design variables (W/S and T/W), three constraints (take-off distance, landing distance and cruise speed), and an objective function (T/W). The aircraft design specification was based on the work of

Bittugieg et al [6], in the design of a Cruise Missile Carrier with multi-role capabilities. Given a specific mission profile and payload requirements, initial estimates can be made for the various operational aircraft weights, through largely statistical/empirical methods. Relationships are also included for various performance parameters, as functions of the given design variables (W/S, T/W). This resulted in the following matching chart:

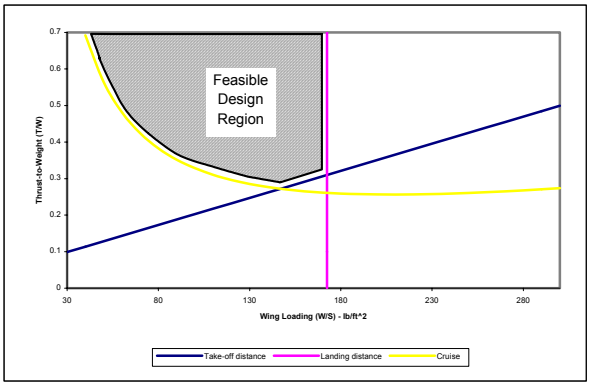

Figure 3: Matching Chart for Cruise Missile Carrier.

Further modifications were made to increase the complexity of the design case, however they will not be mentioned here in detail. The design case was expanded to include five design variables and six constraints. This allowed ADAS to operate on a larger number of processors, allowing a more thorough performance analysis to be performed.

2 ADAS – Incorporating Parallel Algorithms

A parallel program runs simultaneously on a number of processors, as opposed to a conventional single processor architecture. Parallel programs have many advantages over serial (single processor) programs, due to the possibility of performing several operations simultaneously. Parallel programs can share information and allocate tasks, so that all processors are kept busy. Different parallel software layouts (or paradigms) are available that define the relationships between processors, how and when tasks are performed, and how information is shared and collected between processors. Serial programs, however, are limited by the requirement to perform operations once at a time, and in a set order.

When incorporating parallel modifications into an originally serial algorithm (a process known as parallelisation), problems may occur during implementation, and the end product may not be as efficient as one that was written specifically for a parallel environment. This is due to portions of the code still requiring execution in serial; therefore not all the code is strictly parallel. This is the case with the ADAS software when running in optimisation mode.

When ADAS runs in Design analysis or parameter survey mode (without optimisation), calls are made to the user-supplied DSPROG for design-point calculation (only one processor is required for design analysis), and parameter survey calculations are done in parallel. When the optimiser is included, however, ADS is executed. As ADS was originally written for serial execution, this limits the extent of speedup achievable through parallelisation. As the complexity of the design case contained in DSPROG increases, proportionally less time is spent in ADS, and hence the speedup increases; however, for optimal speedup of the ADAS software, the modification of the ADS software, or finding a suitable parallel version, may be required.

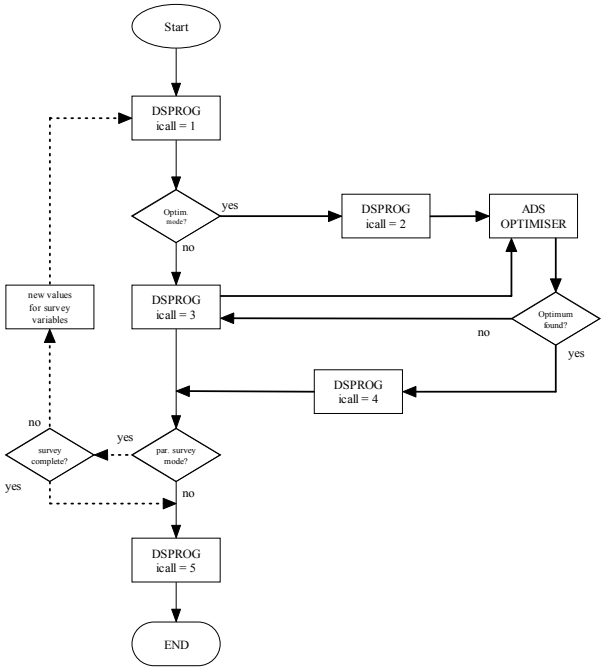

Figure 4: Original ADAS program flowchart.

Figure 4 shows the original ADAS flow chart. Note that the tasks performed by DSPROG vary with the 'icall' parameter. The value of this parameter can be set to allow DSPROG to perform input/output calls, calculate the required constraint and parameter values, etc. There are additional values (icall = 2,4) that are currently blank, allowing the user to insert code at those points if required.

2.1 Optimisation Mode

The first stage taken in the implementation of MPI into the ADAS software, included adding simple MPI commands to perform "Optimisation" mode execution in parallel (Parametric survey mode was not included). The "Task Farming" method was utilised, where a single process acts as Master, with other processes acting as Slaves, sending their results to the master after completion. The general execution flowchart is shown below in Figure 5.

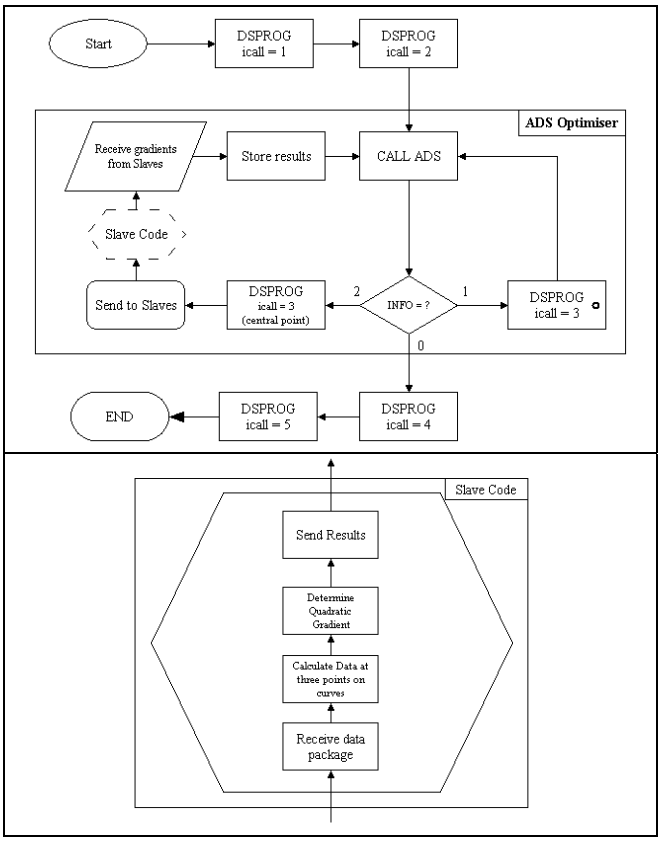

Figure 5: Initial Parallel ADAS Execution Flowchart.
(a) Overall program execution
(b) Specific Slave Code

As can be seen from Figure 5, unless the content of DSPROG is complex, only a small amount of time is spent in the parallel portion of ADAS. The rest is spent in either serial ADS calculations or communications delays, which occur during sending/receiving of messages, etc. These are an inevitable part of using MPI, and even though these delays are relatively small, care must be taken in the organisation of the program to ensure that these delays do not become significant. In the case of ADAS, because such a large amount of time may be spent in serial execution, with slaves idling, special attention was given to maximising program runtime efficiency, both in MPI calls and in improving general software execution.

At runtime, one processor is defined as being the "Master", governing the roles of all other processors running the program. This processor reads and writes input and output files at the start and end of the program respectively, and also manages workload allocation to the slaves, and general communication issues. The code was written such that it can run on any number of processors (of course, user discretion is required to limit processor idling).

2.2 Parametric Survey Mode

The next phase was that of integrating the Parametric Survey Mode into the parallel code. This required the use of more specialised MPI calls, and was in general more complicated than the first phase, as ADAS can be run both modes at once.

Although ADAS was able to organise its allocation of processors to calculate gradients to optimise a single set of values, incorporating parametric survey became more complicated, as multiple such groups need to handle more than one set of values to optimise. In order to achieve this, MPI contains a series of procedures that can be used to re-organise the communications network into a grid structure in any number of dimensions (in this case only 2 are required).

The primary grid length is dimensioned according to the number of parameter sets being studied. The primary grid length is defined to be

equal to the number of parameter sets being studied, or the next lowest equal fraction (e.g. 1/2, 1/3, 1/4, …). The secondary grid length is then determined to utilise all the allocated processors in a rectangular grid. An example of this is shown below, where there are eight sets of parametric values to optimise, and twenty processors free to run the software:

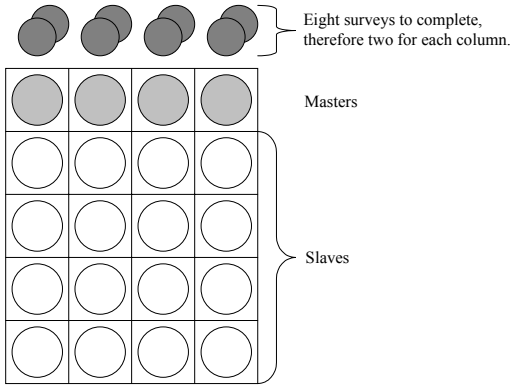

Figure 6: Definition of MPI Communicator Grid Topology.

As can be seen in Figure 6, the grid has been arranged such that no processor is constantly idle, and also the layout is optimised to give the best load balancing between parameter sets, as this is likely to give the best overall efficiency.

Now that the grid is defined, MPI calls are made to define each column in the communicator (as communications networks are defined in MPI) as a separate communicator, such that communication calls can be specified either to communicate internally within each column, or globally, as would be the case when final results are sent from the "Local Master" of each column to the "Global Master", which controls the execution of the program as a whole. The definition of these masters is of no real importance; in effect any processor is suitable to act in these roles, however for simplicity and ease-of use, the 'top' processor in each column acts as a "local Master" to that column, and the local master of the first column acts as the "global master", and has roles similar to the master processor mentioned in the previous section.

This scenario has again been written such that it may be executed on any number of processors (including one), even if it only acts with a single column in the grid.

3 Results – Speedup Achieved

Excellent results have been achieved using the ADAS system. A number of runs were performed, and the complexity of the design case was increased to allow it to benefit from the use of more processors by adding more design variables. Parametric survey analysis was later incorporated into the design case, allowing another dimension of parallelisation to be utilised.

To measure ADAS' performance, three different versions of the program were compiled and run over a range of processors. Each had different lengths of time spent within the parallel code DSPROG, in order to depict designs of increasing complexity. To do this, a simple command was added to the DSPROG code to cause the process to pause execution for 1, 2, 3 or 4 seconds. As DSPROG is called many times during execution, this was sufficient to give an indication of the expected performance. These results were obtained using a design case consisting of 5 design variables, 4 constraints, 3 parametric equations and 20 parametric studies. The parametric studies and equations were only integrated for the results in section 3.2.

3.1 Optimisation Mode

This section shows the results of a single optimisation performed by a varying number of processors. As mentioned previously, several different versions of the software were compiled, to integrate differing levels of complexity. The results for this optimisation case are shown below:

Table 1: Summarised ADAS Optimiser Execution Times.

	Execution Times [seconds]			
#procs	ADAS1	ADAS2	ADAS3	ADAS4
1	67.7833	134.8433	201.8600	268.9167
2	43.9000	86.9000	129.9233	173.0567
3	31.9367	63.0767	93.9800	124.9633
5	20.1833	39.1133	58.0933	77.1233

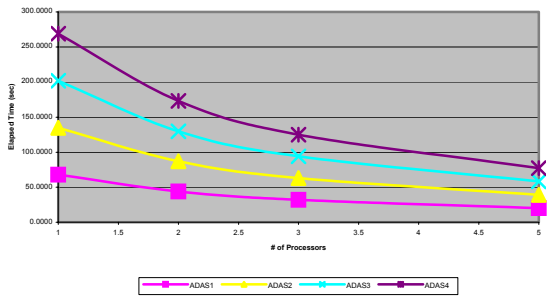

Figure 7: ADAS Optimiser Execution Times.

The method used by the ADAS software to distribute tasks to the slave processors greatly affects the time performance of the software. When distributing tasks for gradient calculations, the master processor allocates them much like a card dealer distributing playing cards, starting with itself, and progressing through all available processors, looping if required. As a result, there may be cases where the allocation of tasks is imbalanced and where processors will have more than one optimisation task to complete. For example, in a 5-Variable design case such as that used to obtain these results, using two processors will result in one being allocated three tasks, and the other only two; hence the second processor will be idle while it waits for the first to complete its extra task. User discretion is advised to choose the appropriate number of processors to use to minimise this imbalance. Especially when solving complex design systems, where execution times are long, even a small load distribution could mean a noticeable waste of precious computer resources.

A common measure of performance of parallel programs is called Speedup, and is a measure of the time taken to execute a program on n processors, compared to the time taken to execute the same program on a single processor. For example, if a program takes 1/3 of the time to execute on four processors compared to a single processor, then the speedup on four processors is 3. The speedup values obtained for the ADAS optimiser are shown below:

Table 2: Summarised ADAS Optimiser Speedup Results.

	Speedup Obtained [-]			
# procs	ADAS1	ADAS2	ADAS3	ADAS4
1	1.0000	1.0000	1.0000	1.0000
2	1.5440	1.5517	1.5537	1.5539
3	2.1224	2.1378	2.1479	2.1520
5	3.3584	3.4475	3.4748	3.4868

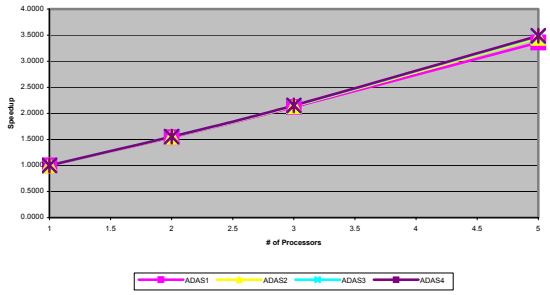

Figure 8: ADAS Speedup Results.

3.2 Parametric Survey Mode

In Parametric Survey mode, a number of optimisation calls, such as those shown in the previous section, were completed simultaneously. The communicator is organised into a grid structure, with each column being responsible for one or more optimisation tasks. A summary of the results obtained is contained in Table 3 below:

Table 3: Summarised ADAS Parametric Execution Times

	Execution Times [seconds]						
#procs	1	2	4	5	10	20	40
ADAS 1SEC	299.8 3	178.3 2	118.8 3	107.1 6	80.70	68.91	57.10
ADAS 2SEC	593.0 8	353.3 4	232.3 9	211.7 2	159.6 1	135.6 2	112.0 4
ADAS 3SEC	891.6 2	529.5 5	348.4 7	315.3 2	239.2 9	203.0 5	167.5 2

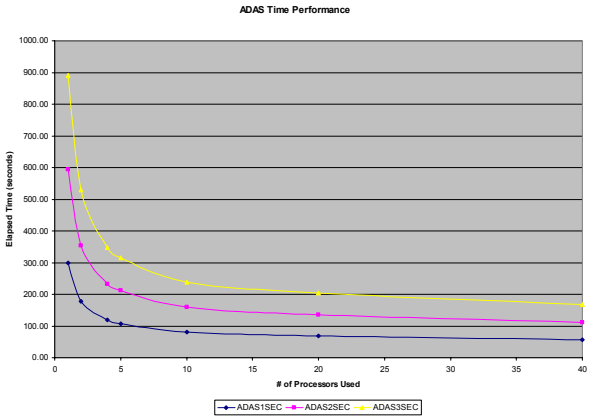

Figure 9: ADAS Parametric Execution Times.

The numbers of processors chosen were such that the MPI communicator in each run had a single row of processors, but increasing numbers of columns, as described previously. If other data points were chosen, there would be irregularities in the chart of Figure 9, due to irregular distribution of tasks.

The speedup of ADAS in Parametric Survey mode is shown below:

Table 4: Summarised ADAS Speedup Performance.

	Speedup Obtained [-]						
#procs	**1**	**2**	**4**	**5**	**10**	**20**	**40**
ADAS 1SEC	1.000	1.681	2.523	2.798	3.715	4.351	5.251
ADAS 2SEC	1.000	1.679	2.552	2.801	3.716	4.373	5.293
ADAS 3SEC	1.000	1.684	2.559	2.828	3.726	4.391	5.322

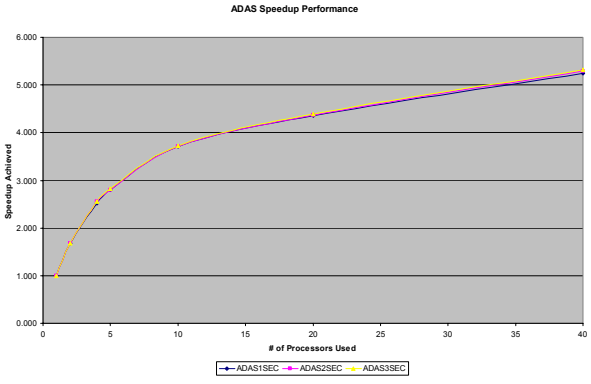

Figure 10: ADAS Parametric Speedup Performance.

It should be noted that the speedups obtained even using different versions of the software (depicting differing design complexities) achieve essentially identical levels of speedup performance (actually increasing slightly with design complexity), which is an excellent measure of efficiency; that even with a higher workload, the speedup performance achieved is still the same. This is important, as design cases become more and more complex, that the speedup performance does not decrease.

Ideally, the speedup achieved by using n processors would be n; however, this is rarely possible due to portions of the code running inherently sequential. The fraction of sequential code is termed *f*, and according to Amdahl's Law [8], the speedup obtainable can never exceed 1/*f*, no matter how many processors are used. Thus, as an example, a program with $f = 0.1$ would never achieve a speedup greater than 10. Factors affecting *f* include the time spent in parallel execution as compared to serial execution, which in this case would theoretically decrease with increasing design complexity; further study should be performed to analyse the performance of ADAS on large numbers of processors to investigate this. Methods of optimising the ADAS code will be incorporated in order to remove sequential overhead and maximise speedup.

4 Results – AutoCAD Output

A final outcome of the modification of the ADAS software was to integrate the existing AutoCAD Integration into the Parallel version. This integration allows for results to be shown graphically in 3 dimensions within the AutoCAD software. Routines were included in ADAS to create AutoCAD 'DXF' graphics files after analysis had been completed. This allowed for creation of 2D or 3D carpet plots of the data, allowing for independent studies of each design variable in the system. Some examples of the plots created for the above case study are shown below.

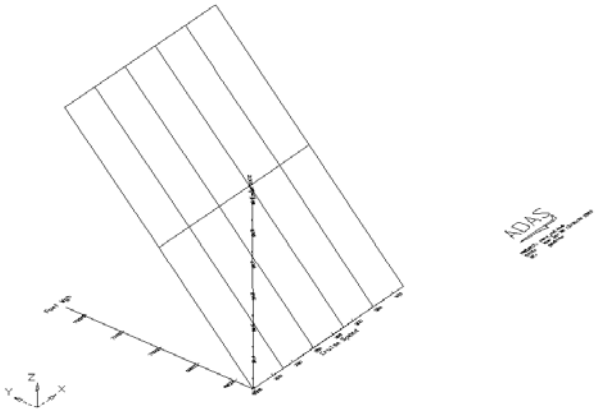

Figure 11: AutoCAD Carpet Plot of Aircraft Endurance as a function of Fuel Weight and Cruise Speed

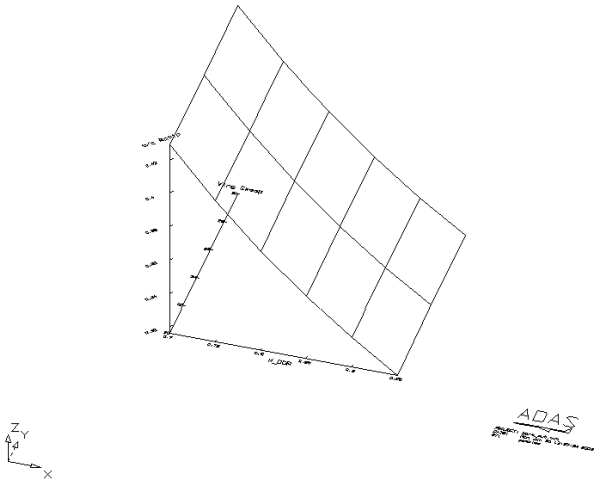

Figure 12: AutoCAD Carpet Plot of Optimum Thickness/Chord Ratio versus Wing Sweep and Mach Number

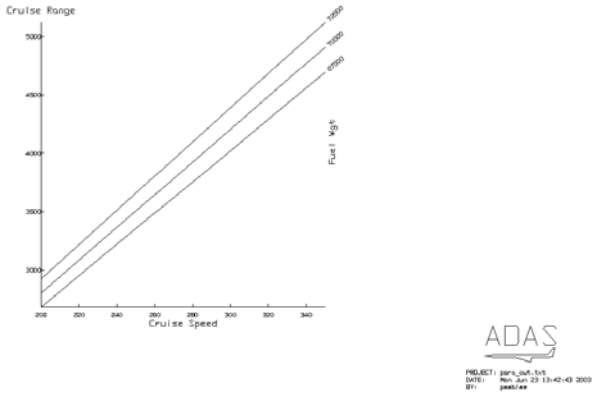

Figure 13: AutoCAD Plot of Required Fuel Weight versus Aircraft Cruise Range and Speed.

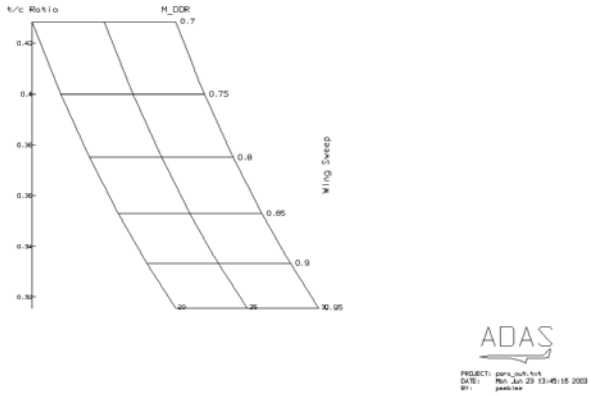

Figure 14: AutoCAD Plot of Optimal Wing Sweep versus Thickness-to-Chord Ratio and Mach Number.

5 Conclusions

This paper documents the method and results of modifying an existing Fortran software package for use on Parallel architectures, using the MPI standard. These modifications allow complex designs incorporating numerous design variables to be optimised without an increase in execution time, as workload is distributed between processors. Results show excellent decreases in execution time as workload is distributed amongst numerous processors, however user discretion is advised to maximise load balancing and minimise processor idling.

The ADAS software provides great advantages to engineers and designers in allowing complex design systems to be solved quickly and easily, and the adaptability of the software allows it to be used in any field of design optimisation. This coupled with the numerous tasks that the software can perform, shows that ADAS will be an invaluable tool for designers in the future.

6 Recommendations & Further Work

The ADAS software system includes many different tools and functions, including incorporation of AutoCAD Computer Drafting software, FEM meshing, etc. While the basic AutoCAD integration has been completed, there are other aspects of the ADAS system that have not yet been incorporated into the parallel

version. The interfacing of ADAS with AutoCAD is achieved through the use of 'DXF' geometry files; plans are currently underway aimed at achieving this same goal with the more modern and widespread CATIA software, in cooperation with Dassault Systems Inc.

Also, while the software may not yet be at a suitable stage in terms of general user-friendliness, an end goal is to make it available as a Teaching and Research tool in a university environment.

7 Acknowledgements

The authors wish to acknowledge the funding and computational support of the Victorian Partnership for Advanced Computing (VPAC).

8 References

[1] Bil C. *Aircraft Design and Analysis System (ADAS) User's Manual.* Delft University Handleiding LR-110, 1992.

[2] Vanderplaats G. *ADS – A Fortran Program for Automated Design Synthesis Ver 2.01 User's Manual.* Engineering Design Optimization Inc. California, 1987

[3] Gropp W, Lusk E, Skjellum A. *Using MPI – Portable Parallel Programming with the Message-Passing Interface.* MIT Press, London, 1995

[4] Bil C. *Development and Application of a Computer-based System for Conceptual Aircraft Design.* Delft University Press, 1988.

[5] Dowd K, Severance C. *High Performance Computing.* O'Reilly, Cambridge, 1998

[6] Buttigieg P et al. *AV465 Design Project – Cruise Missile Carrier Proposal (Group Charlie).* RMIT University Department of Aerospace Engineering, 2001.

[7] Torenbeek E. *Synthesis of Subsonic Airplane Design.* Kluwer Academic Publishers, Dordrecht, the Netherlands, 1982.

[8] Parhami B. *Introduction to Parallel Processing.* Plenum Press, New York, 1999.