Enhanced Efficiency Steam Turbine Blading – For Cleaner Coal Plant

Report No.
COAL R283
DTI/Pub
URN 05/658

March 2005

by

A Fowler, D Bell and C Cao (ALSTOM Power)
R Fowler, P Oliver and C Greenough (CCLRC)
P Timmis (Cranfield University)

ALSTOM Power
Newbold Road
Rugby
CV21 2NH

Tel: 01788 531 777
Email: andrew.fowler@power.alstom.com

The work described in this report was carried out under contract as part of the DTI Cleaner Coal Research and Development Programme. The programme is managed by Mott MacDonald Ltd. The views and judgements expressed in this report are those of the contractor and do not necessarily reflect those of the DTI or Mott MacDonald Ltd

First published 2005

Enhanced Efficiency Blading for Cleaner Coal Plant
Project Number 303

ALSTOM Power
CCLRC, Rutherford Appleton Laboratory
Cranfield University

Objectives of the Project

The aim of this project was to increase the efficiency of the short height stages typically found in high pressure steam turbine cylinders. For coal fired power plant, this will directly lead to a reduction in the amount of fuel required to produce electrical power, resulting in lower power station emissions. The continual drive towards higher cycle efficiencies demands increased inlet steam temperatures and pressures, which necessarily leads to shorter blade heights. Further advances in blading for short height stages are required in order to maximise the benefit. To achieve this, an optimisation of existing 3 dimensional designs was carried out and a new 3 dimensional fixed blade for use in the early stages of the high pressure turbine was developed.

The milestones for the project were defined around the following specific objectives:-

- Increase the accuracy and execution speed of in-house C.F.D. codes.
- Develop new CNC techniques to efficiently produce model turbine blade rows with highly curved 3-D blades.
- Optimise existing 3-D fixed blade designs to improve performance and gain better understanding of how to deal with very short height blading.
- Produce a novel design for very short height fixed blades.
- Perform a model air turbine test on the new design of fixed blade to assess the performance benefit gained from it.

Results of the Project

The work that CCLRC undertook on the ALSTOM C.F.D. code was very successful. The 3-D flow solver code supplied by ALSTOM was analysed and two methods of parallisation implemented. The OMP method of parallisation is only suitable for use on “shared memory” multi-processor computers. The MPI method of parallisation is suitable for use on “distributed memory” computers, sometimes know as “Beowulf Clusters”, which tend to be significantly cheaper to buy than large shared memory computers of similar processing power. As a result of this work, ALSTOM Power have purchased a Beowulf Cluster, and it has become the main workhorse of the Aerodynamics Group.

The development of CNC techniques at Cranfield University has lead to one of their existing CNC machines being converted into a 6-axis machining centre capable of manufacturing a wide range of complex 3-D blade profiles for testing in model turbines. Complete rings or disks can be manufactured on this relatively small CNC machine by making use of an index turntable.

ALSTOM Power developed an improved grid generation package for CFD calculations. The use of “templates” means that different design ideas can be calculated with as similar grids as possible in order to minimise the effects of grid dependency. The use of templates also greatly increases the speed of grid generation.
Optimisation of ALSTOM’s existing patented “controlled flow” design philosophy lead to the development of an “evaluation matrix” for comparing different design concepts. The use of the evaluation matrix has since been further developed in other ALSTOM Aerodynamics design projects, and it is intended to publish this method for ranking competing designs in the future. The evaluation matrix was used to analyse a number of different concepts for a novel very short height fixed blade for use in the early stages of a high pressure turbine. The design that evaluated the best was manufactured and tested in the model air turbine at ALSTOM Rugby.

The model turbine test of the very short height fixed blade design produced a disappointing result, with the performance of the new blade being worse than a prismatically stacked blade. Whilst the reasons for this are still not understood, the result has led to changes in the way that such blades will be designed in the future. It also provides a challenging test case for future CFD code validation.

ALSTOM

FINAL REPORT

Power
Steam Turbines

September 2004

Enhanced Efficiency Steam Turbine Blading for Cleaner Coal Plant

1 Introduction

1.1 Objectives of the Project

The aim of this project was to increase the efficiency of the short height stages typically found in high pressure steam turbine cylinders. For coal fired power plant, this will directly lead to a reduction in the amount of fuel required to produce electrical power, resulting in lower power station emissions. The continual drive towards higher cycle efficiencies demands increased inlet steam temperatures and pressures, which necessarily leads to shorter blade heights. Further advances in blading for short height stages are required in order to maximise the benefit. To achieve this, an optimisation of existing 3 dimensional designs was carried out and a new 3 dimensional fixed blade for use in the early stages of the high pressure turbine was developed.

The milestones for the project were defined around the following specific objectives:-

- Increase the accuracy and execution speed of in-house C.F.D. codes.
- Develop new CNC techniques to efficiently produce model turbine blade rows with highly curved 3-D blades.
- Optimise existing 3-D fixed blade designs to improve performance and gain better understanding of how to deal with very short height blading.
- Produce a novel design for very short height fixed blades.
- Perform a model air turbine test on the new design of fixed blade to assess the performance benefit gained from it.

1.2 Participants in the Project

The participants in the project were ALSTOM Power Ltd (lead partner), Council for the Central Laboratory of the Research Councils (CCLRC) and Cranfield University.
CCLRC created a parallel version of an existing ALSTOM viscous flow solver and assessed the optimum hardware to run the code on a cost per calculation basis.
ALSTOM undertook an optimisation exercise on their existing, patented “controlled flow” fixed blades that had already been shown to give a significant performance increase at taller blade heights. The short height blades in the

high pressure cylinder of a steam turbine have a lower efficiency than longer blades due to the increased effect of “secondary flow” or endwall losses. A novel 3-D design for use in early high pressure stage fixed blades was developed.
The final measure of the success of the project was a model turbine test of the novel design of the very short height fixed blade. In order to facilitate the manufacture of components for model turbine tests, Cranfield University developed a novel method of machining complete bladed rings or “blings”.

1.3 Activities of the Project

1.3.1 Analysis of Existing “GENESIS” C.F.D. Code (CCLRC)

The 3-D Viscous Flow solver “GENESIS” has been written and developed at ALSTOM over a number of years. The present state of the code is such that it is the primary tool used by ALSTOM for analysing new steam turbine blading. Benchmarking has shown that the code is more accurate than any other commercially available turbomachinery code currently on the market. Execution time is a major factor of any 3-D C.F.D. calculation. CCLRC used their experience gained from analysing other large computer codes to assess what improvement in performance could be achieved simply by restructuring the code. CCLRC then implement their findings.

1.3.2 Parallisation of “GENESIS” C.F.D. Code (CCLRC)

The next step in reducing the execution time of “GENESIS” was to run the programme simultaneously on multiple processors. CCLRC again undertook this work. As part of this phase of the program, CCLRC executed the code on the range of computer facilities they have available to them to assess the optimum hardware on a cost per calculation basis. The investigation included the use of a large array of relatively small, cheap individual processors (a so called “Beowulf Cluster”).

1.3.3 Improved Computational Grid Generation (ALSTOM)

The steep exit angle of the flow from an impulse steam turbine fixed blade is a particular problem when trying to calculate the flow field. The skewness of traditional “H-type” grids in the trailing edge region leads to increased numerical errors in the solution and an inaccurate calculation of the efficiency of the blade profile. In order to minimise these errors a different gridding procedure is necessary that reduces the skewness of the grid in the trailing edge region. The use of “O-type” grids was investigated, which are better able to keep the computational mesh orthogonal to the blade surface.

1.3.4 Optimisation of Current Controlled Flow Designs (ALSTOM)

ALSTOM’s patented “controlled flow” fixed blade has been proved to work both in the model air turbine in Rugby and in real applications at site. The designs used at site are based on the model turbine results. However the range of conditions found in site applications is much larger than can be feasibly tested on the model turbine. Therefore many applications of controlled flow are not fully optimised. By using the enhancements to the calculation methods described above, it was possible to develop an “evaluation matrix” for comparing and optimising different design concepts.

1.3.5 Development of Very Short Height Fixed Blades (ALSTOM)

The above mentioned model turbine tests have highlighted one shortcoming of the current “controlled flow” design used by ALSTOM. The design is not

applicable to the very short height blades typically found at the front of high pressure steam turbine cylinders. The short height blades in the high pressure cylinder have a lower efficiency than longer blades due to the increased effect of “secondary flow” or endwall losses. The optimisation study outlined above provided valuable insight onto how these very short height stages should be designed. This led to a novel 3-D design for early high pressure stages that was tested in the ALSTOM model turbine in Rugby.

1.3.6 Improved Methods for Producing Model Turbine Components (Cranfield)

Cranfield University undertook to develop a technique for machining large bladed rings and bladed disks (so called “blings” and “blisks”) for use in model turbine test facilities. This involved using a six axis machine: three rotary axes, three linear in the interests of accuracy and moderating the extent of slide movements of the machine.

The practicality of producing large “blings” and “blisks” using high speed milling was investigated. High speed milling of these components on a five / six axis machine required different cutting speeds and depth of cut than is given in standard data. It was also different from that which is optimum for cutting individual blades, where the access to the blade surface allows the use of short, stiff cutters. The investigation included high speed machining trials to determine this data.

1.3.7 Model Turbine Test of Very Short Height Fixed Blade (ALSTOM)

A model turbine test of the novel, very short height fixed blade was carried out to give a direct measurement of the benefit gained from using the new design.

1.3.8 Project Management and Reporting (ALSTOM)

The project was managed by a project manager appointed by ALSTOM. Quarterly progress reports were submitted as well as this detailed final report.

2 Analysis and Parallisation of “GENESIS” C.F.D. Code

This section describes the current message passing (MPI) implementation of the parallel version of the ALSTOM code “Genesis”. The overall structure of the parallel version is given along with some more detailed notes on the changes to particular subroutines and new functions. Performance figures are given for the message passing code on a range of different machines. The combination of OpenMP and message passing implementations is briefly discussed along with possible future developments on the software.

2.1 Introduction

The message passing version of Genesis has been developed to allow the code to run on Beowulf style systems which use distributed memory. In this implementation the aim was to ensure that the results produced by the parallel version are identical to those of the serial case. This should give confidence in the parallel results, though may not give the best possible performance. Further work could be done to give better overall performance by changing the algorithms used, and some suggestions about this are given in the conclusions.
Genesis is written in Fortran 90 and hence the parallel aspects of the code also use this language. The message passing is built on top of the MPI library, which includes a Fortran binding and is the most widely supported standard at the present time. To allow for adoption of other message passing systems in the future, a simple interface library between Genesis and the underlining MPI functions has been used. This is described in a separate document.
A brief over view of the current structure of the code is given in section 2.2, while the changes that have been made for the parallel version are listed in section 2.4. Some discussion of mesh partitioning and load balancing requirements are then given followed by some notes on the individual routines that have been changed and added. Performance figures on a range of machines are then given, while possible future optimisations are discussed in the final sections.

2.2 Structure of “Genesis”

Genesis is a multiblock code with each mesh block a regular IJK mesh that will have been fitted to the local geometry. At each block-block interface the neighbouring meshes are currently conforming and overlap each other. The overlap region consists of four points. However, each block is solved for only the points “two in” from the edge. The first two points are only used as boundary values. Hence each point is only evaluated in one block. Frequent exchanges of boundary information are made between blocks, using routines such as *set_phi,* which are called after each set of iterations of the linear solver.
The first task in developing a parallel version of Genesis was an analysis of the existing structure of the code and the amount of computational time spent within each part. The table below shows the CPU analysis of a 3D test case for all routines taking more than 2% of the total time.

FUNCTION	CALLS	TIME	%
Lisolv	289380	22035	32
Quicks	30618	7140	10
Calcp2	15000	6213	9
Mflux	15006	5851	9
Coeff	44706	3803	6
Set_phi	430134	3679	5
Calcte_c	14700	3125	5
Calcp_c	15000	2194	3
vist	14700	2130	3
Calcvz_c	15000	2065	3
Calcvy_c	15000	2046	3
Calcvx_c	15000	2006	3
Total		68275	91

The results show that, as would be expected, the most time consuming routine is the one called LINSOL which performs an iterative solution of a sparse system of linear equations using the line-SOR method. The other routines are mainly associated with the assembly of the linear equations which are to be solved. The main exception to this is routines such as SET_PHI which exchange boundary data between mesh blocks. In fact Genesis steps through mesh blocks one at a time, forming a local solution on each. It depends on the exchange of block boundary data at the end of each such pass for overall convergence to the global solution. This structure of the code allows a straightforward approach to parallelisation at the block level which can be exploited in both shared and distributed memory cases.
The solution process in Genesis can be described in general terms by the following set of steps:

1. **Data input:** The routines *read_genesis_input* and *read_cgns_input* are used to read the run data and the mesh data from the *.prm* and *.cgns* files. The filenames are determined from command line arguments. Arrays to store the input mesh and related data are dynamically allocated. If an initial solution is to be calculated on a coarse mesh, the code attempts to delete half the cells in each dimension at this point.
2. **Initialization:** The routine *startup* is called to set many initial values, allocate further data space and determine the distances of grid points from the nearest wall. This last calculation is quite computationally expensive of itself, though in relation to the overall run time it is not significant.
3. **Read existing solution:** An existing solution from a previous run can be read in. This section of the code is not exercised by the test cases provided. No attempt has been made to implement this case in the parallel code, though it should not be difficult to do so.
4. **Perform other initial calculations:** This includes calls to *calc_m_iso* and *calc_time_step* which perform block based calculations, but also determine global maximum and minimum values across the whole mesh. *Set_init_fl_f* performs a number of iterations over the mesh blocks to set initial flow values. These iterations are independent of each other, apart from boundary data exchange (*set_phi).* Also calls to *density* and *mflux* are

made which again are block based calculations, though the former is followed by calls to *set_phi*.

5. **Main iteration loop:** This is where the vast majority of the Genesis run is spent. Each iteration involves a call to the *solver* routine. Calls are also made to *mflux* and to *ypout.* Various log files are written during the solution process. If the two level mesh option is selected, then after a fixed number of iterations the original fine mesh is reloaded and used for subsequent calculations. This requires calls to the data reading subroutines again and to routines to interpolate the existing solution onto the finer mesh.
6. **Data output:** After the required number of iterations have been made calls are made to *cgns_out* and *wall_press.* The former saves the current solution data to a CGNS file while the latter writes the pressures on selected walls to a text file.

The *solver* routine, which performs most of the computational work, calls a large number of routines itself. The basic flow is as follows:

1 **Calculate flux:** Routines *tot_flux* and *bl_flux* are called to get flux related values. These are blockwise calculations but require global sums across all blocks. The latter routine writes flux data to a file.

2 **Loop over variables:** For each state variable (velocity components, density, etc.) do:

- 2.1 Calculate associated terms on a block by block basis.
- 2.2 Solve the resultant linear system on each block using a fixed number of passes of a line solver routine.
- 2.3 Swap block boundary values of related variables.
- 2.4 Repeat until the residual is small enough or until a fixed number of iterations have been made.

3 **Output monitor values:** This includes printing values such as the mass error and the location of the minimum pressure cell.

There are many other routine calls made from the *solver* and its descendants, but these are mainly block based calculations. The details of mixing plane calculations have not been examined and are not exercised by either of the test cases available.

2.3 Parallelisation Methods

Given the existing structure and programming language used by Genesis there are three approaches which could be used to develop a parallel version of the code. These are:

1. OpenMP based parallelisation. Many Fortran90 vendors support the OpenMP standard for parallelisation on shared memory machines. This method is based on inserting directives into the code which instruct the compiler to execute certain loops or code sections in parallel. As these directives appear as comments to any compiler that does not support OpenMP they are simply ignored in such cases. This means that, in general, a single set of source files can be used for both the serial code and the parallel code. The fact that all variables are shared in this model makes the parallelisation task a lot easier as each processor has access to all variables. The main problem is in identifying which loops to parallelise and defining the status of variables within the selected loops. In some cases an important loop is intrinsically serial and it is then necessary

to see if algorithmic changes can be made to the code to enable parallelisation. Shared memory machines are however expensive and have limited scalability.

2. Message passing parallelisation. This approach is more flexible than OpenMP, since it will work for both shared and distributed memory machines. It is however much more difficult to implement since the programmer is responsible for making sure that each processor has access to all required data. The Message Passing Interface (MPI) is emerging as the dominant standard here and is supported on virtually all major parallel platforms.
3. High Performance Fortran (HPF). This is another parallelisation method that again uses comment style directives to instruct the compiler as to the best way to distribute the computation. Unlike OpenMP, HPF is designed to work on both shared and distributed memory architectures. To do this it includes directives not just to specify which loops should be parallelised but also how the data should be laid out across machine memories. It is particularly suited to computations on regular grids such as that used by Genesis. It terms of complexity this method lies somewhere between the OpenMP and MPI approaches. However at present there are few compilers that fully support HPF. In addition those that do are known to have somewhat variable performance and it is difficult to write code that is both portable and efficient in this language.

Because of the portability and optimisation problems of HPF, this method was not looked into further. A message passing implementation was seen as the most useful approach to take as it would provide a version of Genesis that would run on both shared and distributed memory machines.

It is becoming more common for Beowulf clusters to include shared memory multi-processor nodes. On such systems it is possible to combine OpenMP and MPI methods in a single code. This can be advantageous in some circumstances. For example MPI distributed memory parallelism may be used at high level for the mesh blocks within Genesis while OpenMP can be used at lower level to parallelise individual loops on each shared memory node. This may be particularly useful when there are only a limited number of mesh blocks to distribute, as may be the case for Genesis.

2.4 Message passing implementation

The chosen method of parallelisation is one based on splitting the available mesh blocks between processors. Each processor will then perform the standard set of calculations on its set of blocks. The only modifications to the code that runs on each processor are related to:

1. The initial data distribution. One processor, the *master,* will be responsible for all file I/O. It will deal with reading the input data and sending it to the other processors, the *slaves.*
2. The calculation of certain terms in the pre-processing phase. In particular the determination of cell wall distances is most easily done on the master, since all wall data is available at this point.
3. Calculation of global values over the whole domain. At various points it is necessary to find quantities such as the minimum pressure over the whole grid and this requires global operations between all processors.

4. Exchange of boundary data for block interfaces where the two blocks are assigned to different processors. When this occurs it is necessary to pack interface data and send it to the remote processor, and in return receive back data that must be placed in the local mesh block.
5. Collection and output of log files and final results. As the master deals with all file I/O, the slaves have to send data to be written to file to the master. This includes data such as wall pressure and the final solution. Getting the data printed in the same order as the serial case makes the task a little more complex.

These changes have been implemented by modifiying about 19 routines in Genesis along with the addition of a number of routines explicitly related to the parallelisation. As far as possible the original code has been retained to avoid too great a divide between the parallel and serial versions.
All new routines have been written as Fortran 90 modules to allow explicit checking of interfaces. Existing routines have been left with their existing implicit interfaces.

2.5 Partitioning and load balancing

While the chosen message passing method is compatible with the existing structure of Genesis, it has some limitations. The most important of these is load balancing. Assuming a homogeneous cluster of processors, then it is desirable to allocate the same amount of work to each of them. If it is assumed that the amount of work is proportional to the number of cells in each block, then we want to assign as close as possible the same total number of cells to each processor. For the 3D test case supplied there are only 6 mesh blocks, which instantly puts a limit of using at most 6 processors in the parallel case. However the block sizes vary greatly and it is hard to go beyond 4 partitions and maintain reasonable load balance. It may be that in future the mesh generation process can be modified to take some account of load balancing requirements. Larger simulations will generally require the use more blocks and give more opportunity for using greater levels of parallelism.
The mapping of mesh blocks to processors is currently left up to the user who must supply a file with the required information. In keeping with the existing file usage within Genesis, this file must have the standard run prefix and a suffix of the form *partNN,* for example *45in_mb.part02* would be the partition information for run 45_mb using two processors. The format of the file is very simple, it just contains the processor number of each block. Some example partitioning files are included with the parallel Genesis software. One such is the file *668n2_3d_ch.part03* which contains:

0
0
1
1
2
2

This just means that of the six blocks in this mesh, 1 and 2 are mapped to processor 0 (the master), 3 and 4 to processor 1, while 5 and 6 go to processor 2. The number of cells in mesh blocks 1 to 6 is: 42900, 39000,

27300, 62400, 19500 and 46800. This gives a total of 237900 cells and hence the best load balance would be when each processor has one third of this, i.e. 79300 cells. The above mapping actually gives 81900 cells to processor 0, 89700 to processor 1 and 66300 to processor 2.
The load balance in this case is not perfect, with the greatest load, on processor 1, being about 13% greater than the ideal value of 79300. If we ignore the communication costs of the parallel method and just look at the lack of load balance, it can be seen that the best possible speed up on three processors is now not 3.0 but rather 2.65. Another possible partition mapping for three processors is:

0
0
1
2
2
1

This actually gives a rather better load balance, within 3.3%, and a maximum possible speed up of 2.9 on three processors. While this partitioning is rather better for load balance, it will lead to more communication. The reason for this is that in the 3D example the mesh blocks have a simple linear topology: block 1 connects only to block 2, block 2 connects to both 1 and 3, and so on. With the second partitioning of blocks given above there are now three interfaces between processors, where previously there were only two. This may not be a significant problem when communication is very fast (e.g. for shared memory machines) but may be expensive for a Beowulf cluster. A simple test on a four processor SGI machine (with shared memory) showed about 8% better performance with the second partitioning.
The current implementation of parallel Genesis has a restriction on the partitioning that blocks with a periodic interface in common must be assigned to the same processor. This is not a problem for the above 3D test case, where no such blocks occur. It is a consideration for the 2D example (*45in_mb)*, where blocks 1 and 5 must be on the same processor, as must blocks 4 and 8. This restriction may add to the difficulty in finding a good load balanced partition with low communication requirements.

2.6 Parallel Performance

The parallel genesis code has been compiled and run on the following machines:

Name	Specification
Wulfgar	850MHz Athlon Classic Myrinet
Hrothgar	1.2GHz AthlonMP Wulfkit
Proton	SGI Origin2000300MHz R12k
Columbus	Compaq ES40500MHz EV6
Wiglaf	1.6GHz AthlonMP
Herodium, Ashkelon	1.6GHz Pentium P4 networked machines.

These machines make use of several different MPI implementations, several are based on the public domain MPICH, while others are proprietary versions. This gives some confidence in the portability of the parallel implementation. Results have been obtained for the 3D case *668n2_3d_ch* which contains 6 mesh blocks, as mentioned in the discussion on partitioning. Due to the limited number of blocks, and the large variation in size of these, measurements have only been made using up to 4 processors. The partitions files used were:

part02: 0 0 0 1 1 1 error= 8.2%
part03: 0 0 1 1 2 2 error=13.1%
part04: 0 1 1 2 3 3 error=11.5%

The errors quoted above indicate the extent of the load imbalance due to the partitioning of the limited number of blocks and their sizes. This is ratio of the ideal number of cells per processor to the worst case, using the block cell counts mentioned previously. This limits the speed up that could be expected if communication costs were negligible.
The following table shows the run times observed for 1000 steps of the 3D case using between 1 and 4 CPUs:

Name	1	2	3	4
Wulfgar	8774	5028	3533	4120
Hrothgar	5594	3066	2180	2247
Proton	7132	3844	2650	2046
Columbus	2944	1660	1651	1253
Wiglaf	4358	NA	NA	NA

The best results in terms of speed up are seen on the shared memory SGI machine, Proton, with a speed up of almost 3.5 on 4 processors. All the machines except Columbus show good speed up of 2.5 to 2.7 on three processors, considering that the load balance is not perfect and there is some pre-processing work that has not been parallelised. It is disappointing that both the Beowulf clusters show a drop in performance when using 4 processors, though with further optimisation work it should be possible to extend the scalability. Larger test cases are also likely to show better scaling. The speed up values for each of the above machines is shown in the table below. Figure 1 shows a plot of these results as a function of number of processor.

Name	1	2	3	4
Wulfgar	1.0	1.745	2.483	2.130
Hrothgar	1.0	1.825	2.566	2.490
Proton	1.0	1.855	2.691	3.486
Columbus	1.0	1.733	1.783	2.350

2.7 Further Optimisation Work

The current message passing implementation has been designed to give, as close as possible, the same convergence as in the serial case. This means that it exchanges block boundary data very frequently. While this is a fairly cheap operation for the serial code, it can be quite expensive on a distributed

memory machine. Obviously there is a need to exchange boundary data for global convergence, but it may be more cost effective in the parallel case to perhaps do more passes of the line solver before exchanging data. For variables which converge quickly it may be more efficient to skip some of the boundary exchanges across processors associated with these quantities.
There is considerable scope for merging global operations, such as the global sums to find the residual in the solver iterations. While this is not too costly with small numbers of processors, it can become important at higher levels of parallelism. For 2D problems it may be useful to try and merge boundary data exchanges since the message size can be very small in these cases. Larger messages are usually more efficient.
These algorithmic changes need to be evaluated across a number of realistic problems on computer hardware of interest. There is also the need to improve load balancing and communication costs. One way to do this would be to actually split the existing mesh blocks into better shapes. In the current cases, where the mesh blocks have one to one connectivity, it should be possible to transfer parts of blocks from one to another. While it would be necessary to retain the rectangular topology, it should be possible to develop routines to try and find both good load balance and minimal size of the interface between processors.
The calculation of cell to nearest wall distance is currently performed only on the master processor. By sending copies of all the necessary wall data to every processor this could also be done quite efficiently in parallel. The only drawback with this is the extra memory that will be required on all processors for the wall data. With careful allocation it should be possible to reuse this space for arrays allocated later.

2.8 Combining Message Passing and OpenMP Parallelism

OpenMP parallel directives have been included in some 16 of the most computationally intensive routines for the serial version of Genesis. Loop based parallelism was chosen, rather than applying directives at a high level over the mesh blocks. This makes the OpenMP and message passing methods more complementary in the case where there are only a few mesh blocks to distribute yet the architecture provides a cluster of shared memory nodes.
Most of the OpenMP changes are just straightforward additions of directives outside of major loops. The only difficulty is to specify the state of loop variables as private, shared, reduction, etc. It was assumed that problems of importance will be 3D cases in selecting which are the most important outer loops.
The only case where such simple changes will not work is in the line solver *lisolv.* In this case the existing loops are strictly serial. Since this routine accounts for a least a third of the serial CPU time, it was necessary to introduce parallelism in some way. To this end the loops are reordered in a simple red-black fashion: we first step through even *k* values in the outer loop, then go through the odd values afterwards. This means that separate *k* iterations are independent of each other. This will have some effect on convergence, though initial tests suggest it is not very great. The explicit colouring of the *k* loops means that this is only effective for 3D problems.

When using both message passing and OpenMP together it may be necessary to explicitly control the number of threads assigned to the master and slaves. This is possible under OpenMP using the subroutine *omp_set_num_threads.* In the current version of the code, *gp_main* in the *./omp_dir* directory was changed to read two additional command line arguments, the number of threads to use when just the master is active and the number to use when the slaves are active.
To see how these could be used, consider running the MPI/OpenMP code on a four processor SGI shared memory machine. If 4 processors are used in the message passing side then it makes sense to only allow each slave to use one thread. However, when the master is performing the initial calculation the slaves are not active and 4 threads could then profitably be used. This can be achieved by adding “4 1” at the end of the command line to run the combined code, e.g.:

```
mpirun –np 4 P_GENESIS.exe /tmp/3d/ 668_3d 4 1
```

Similarly, if one wanted to run with just 2 message passing domains, then each could make use of two threads, while the pre-processing step would again take 4 threads.
On the SGI machine, Proton, it is found that the best performance is obtained using the pure message passing code. The run time for the 4 processor case was 2046s, as reported in the table in section 6. The combined MPI-OpenMP version, with 4 partitions and using 4 threads for the master and 1 for each slave, takes 2089s. In this case the pre-processing work is small, and the times for the two versions are very close. Running the same problem with just one domain, but 4 threads at all times, the elapsed time is 2426s, which is significantly slower. On an SGI machine with more processors it would be advantageous to use both OpenMP and MPI since there are not enough blocks in the mesh.
Tests have also been made on the Beowulf cluster Hrothgar, described above. Using three dual processor nodes, each running two threads for both master and slave, gives an elapsed time of 1217s, which represents a speed up of about 4.6 using six processors. Again the result gets slightly worse when moving to four dual nodes. This may be partly due to the load balance, though the communication load is likely to be the main problem. The speed-up results for the OpenMP-MPI code are shown in Figure 1, along with the pure MPI data.

2.9 Conclusion

Message passing and OpenMP versions of the Genesis software have been developed. Reasonable speed-ups have been observed for the two separate implementations on both shared memory machines and, for the former, on distributed memory machines. Tests of the combined OpenMP and message passing version of the code show this to be practical and to offer better performance than either method on its own for some computer architectures. The performance improvement from the combined version is anticipated to be greatest for large calculations run on distributed memory computers. This will be confirmed when the code is used for more realistic calculations later in this “Cleaner Coal” project.

There is still some scope for further improvements in performance of the code with algorithmic changes and optimisation of the block partitioning. These fall outside the scope of this “Cleaner Coal” project work.
The parallelised version of Genesis has been delivered to ALSTOM and successfully installed on their computer system. It is already regularly used by the Aerodynamics R&D group.

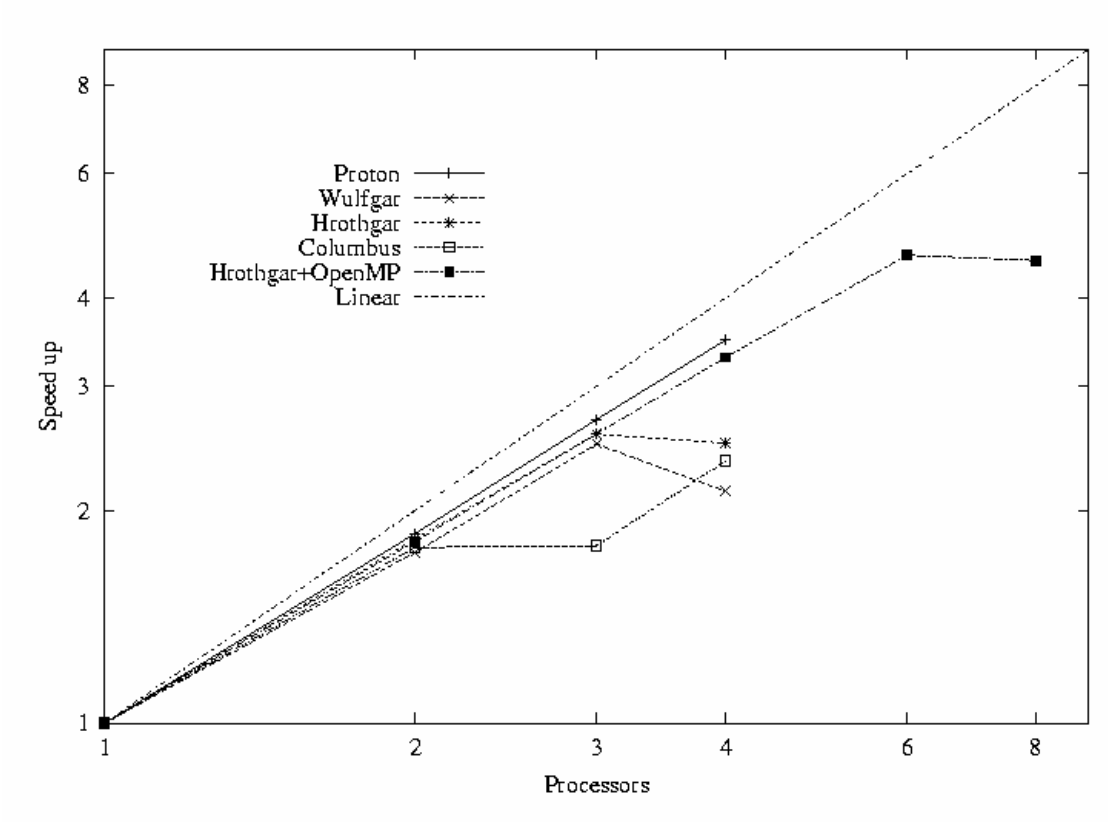

Figure 1: Speed up of 1000 steps of the 3D test case on a range of machines. The second Hrothgar data refer to the combine MPI-OpenMP version of the code.

3 Improved Computational Grid Generation

3.1 Introduction

The generation of high quality structured grids for CFD analysis of blade flows in impulse, high pressure (HP), steam turbine cylinders presents a significant challenge, due to the characteristics of the fixed (stator) blade sections and the typical layout of a stage. As illustrated by example in Figure 2, HP impulse turbine stages have very short inter-stage gaps (measured in the axial direction between the trailing edge of the fixed blade and the leading edge of the adjacent moving (rotor) blade). This, combined with the acute exit angle of fixed blades in impulse designs and the presence of matching periodic block boundaries (shown in Figure 3), effectively constrains the grid generation process in a manner that directly conflicts with the production of high quality grid.

Traditionally, blade flow CFD analysis within the axial-flow turbomachinery community has been performed on single block grids, and most commonly on so called simple-H grids (straight gridlines in R-Theta direction). Whilst simple-H grids may be acceptable for CFD analysis of axial flow compressors, because of their low turning angle, the high level of skewness associated with these types of grid when applied to impulse HP turbine blade domains gives rise to well documented numerical errors in the subsequent CFD analysis.

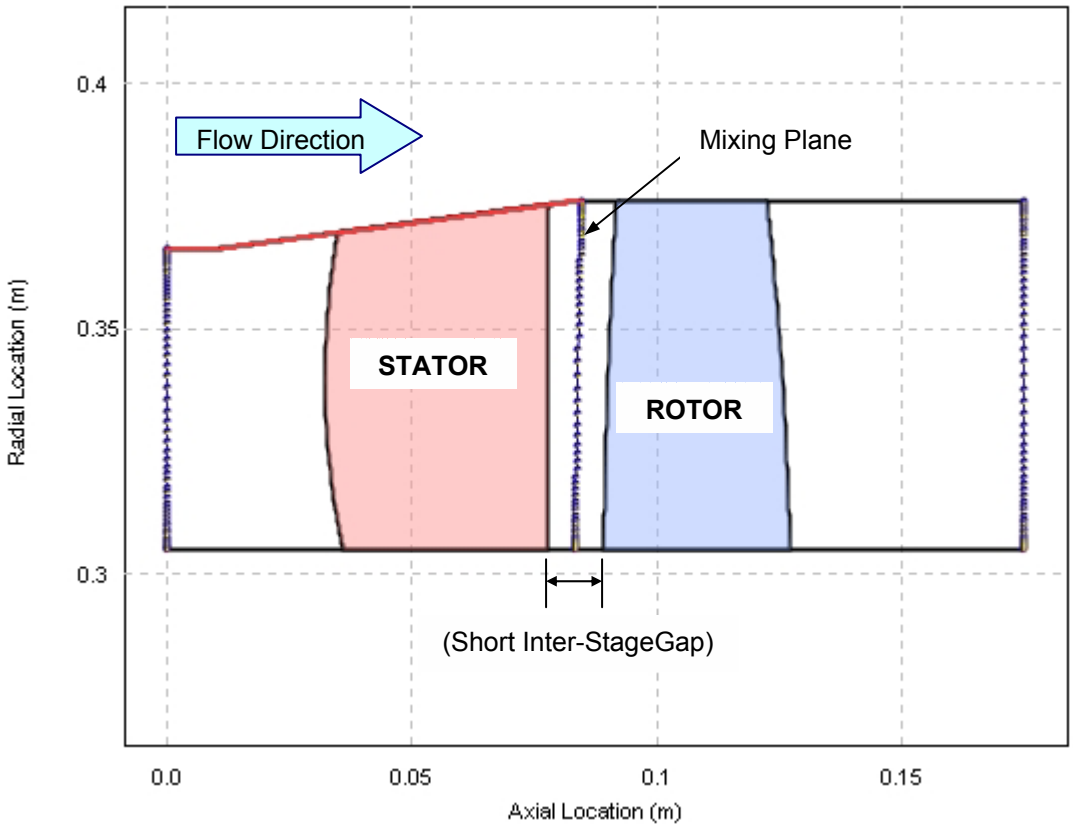

Figure 2: Typical Layout for HP Stage CFD Computation (Air turbine build 28)

To increase the flexibility of the structured grid generation process and improve grid orthogonality, thereby improving CFD solution quality, a multi-block grid approach has been adopted by ALSTOM Steam Turbines for application in the analysis of blade flows.

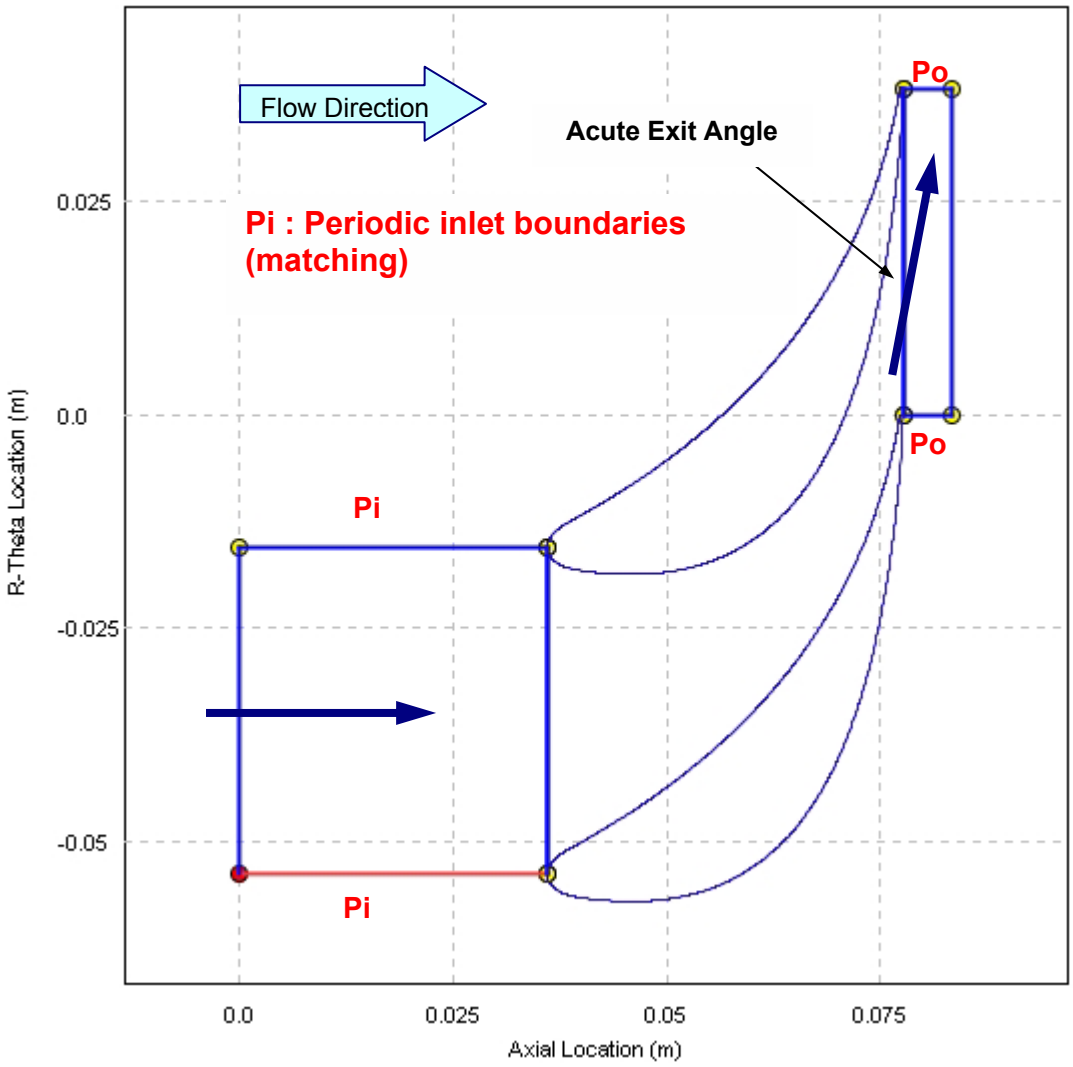

Figure 3: Typical Fixed Blade Section (Impulse Design) – Acute Exit Angle
(Section taken through the CFD domain shown in Figure 2**)**

3.2 Improvements in Grid Generation

As part of the Cleaner Coal project two developments have been made to improve the existing multi-block grid generation process. The first is the development and application of grid templates, which enables the generation of "similar" grids from case to case. This is important because practical CFD

computations invariably exhibit some degree of mesh dependence. The production of “similar” grids from case to case, should therefore minimise the risk of mesh dependence in CFD computations corrupting the comparison of competing designs. The second development is the implementation of grid smoothing, which is intended to directly improve the quality of the grid.

3.2.1 Development & Application of Grid Templates

Grid templates have been developed and implemented in ALSTOM Power Steam Turbine’s grid generator for blade flow applications. These templates are stored in files and contain grid data at specified radial stations in the grid (normally the root, mid-height and tip section), which is sufficient to generate a mesh for another geometry.

The grid templates store the following information:

- Grid block boundary type (simple-H, curvilinear-H, H-O-H)
- Blade inlet and outlet angles
- Normalised location of grid control nodes within the domain (shown in Figure 4)
- Specification of grid distribution along edges of the block boundaries
- Setting for smoothing

The normalised location of grid control nodes, stored within the templates for specific radial locations in the grid, is also used for interpolating the location of corresponding control nodes at other radial locations. This leads to a controlled development of grid block boundary structure away from the radial sections defined in the grid template, even if the blade is strongly three dimensional and exhibits a high degree of twist. In essence, this means that the user need only specify the appropriate location of grid nodes at a few radial locations within the grid in order to generate a high quality 3D mesh. It is normally found that specification of the grid control nodes at root, mid-height and tip is sufficient for this purpose.

Templates are automatically created for every case and can be loaded through the graphical user interface at any time. An example application of grid templates is shown below (Figure 4). In addition to the benefit of generating similar grids from case to case, the application of grid templates also reduces the overhead involved in generating a good quality multi-block grid.

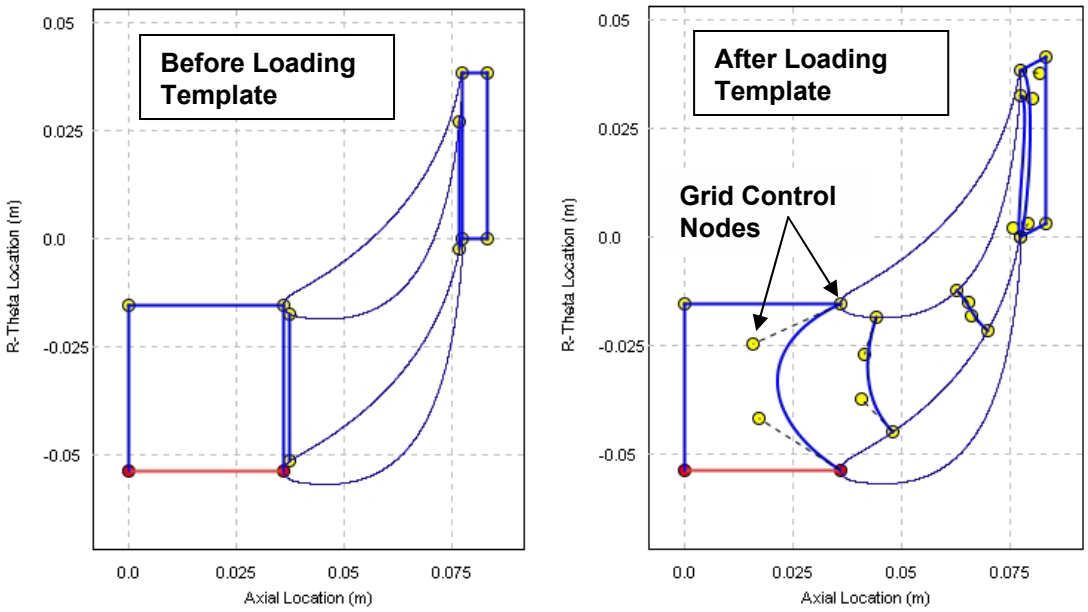

Figure 4: Application of a Grid Template
(shows positioning of grid control nodes for a an example curvilinear-H grid template)

3.2.2 Improved Grid Quality through Application of Grid Smoothing

In order to improve grid quality three additional modules have been introduced in the grid generator, which effectively smooth the grid to reduce the maximum grid skewness and the change in grid size in adjacent cells.

Four options are now presented to the user for grid smoothing:

1. *None*: This is the basic option – transfinite interpolation, without grid smoothing.
2. *Algebraic*: This applies algebraic grid smoothing to the complete grid that is originally generated through trans-finite interpolation.
3. *Elliptic-BB*: This solves a poisson system on a block-by-block basis with stretching functions included in an attempt to maintain the grid distribution defined at the block edges, throughout the grid block.
4. *Elliptic-GL*: This also solves a poisson system with stretching functions, but is applied to the complete grid, rather than on a block-by-block basis, in order to reduce the change in cell size of adjacent cells on either side of a block boundary.

Both the elliptic and algebraic grid smoothing yield a tangible improvement in grid quality in terms of the level of grid skewness and change in grid size in adjacent cells, with the elliptic options generating the highest quality grid. Unfortunately, with the present stretching functions, the distance to the wall of the first cell is not rigorously controlled in the elliptic grid option, and the resulting grid may not necessarily have the desired near wall grid distribution which is required for correct application of the turbulence model used in the CFD analysis. As a consequence, the preferred mode of smoothing at present is the algebraic form. Further development of the elliptic grid generation option

should, however, yield an additional improvement in grid quality, but this is beyond the scope of the planned Cleaner Coal project. An example application of algrebraic grid smoothing is shown in Figure 5.

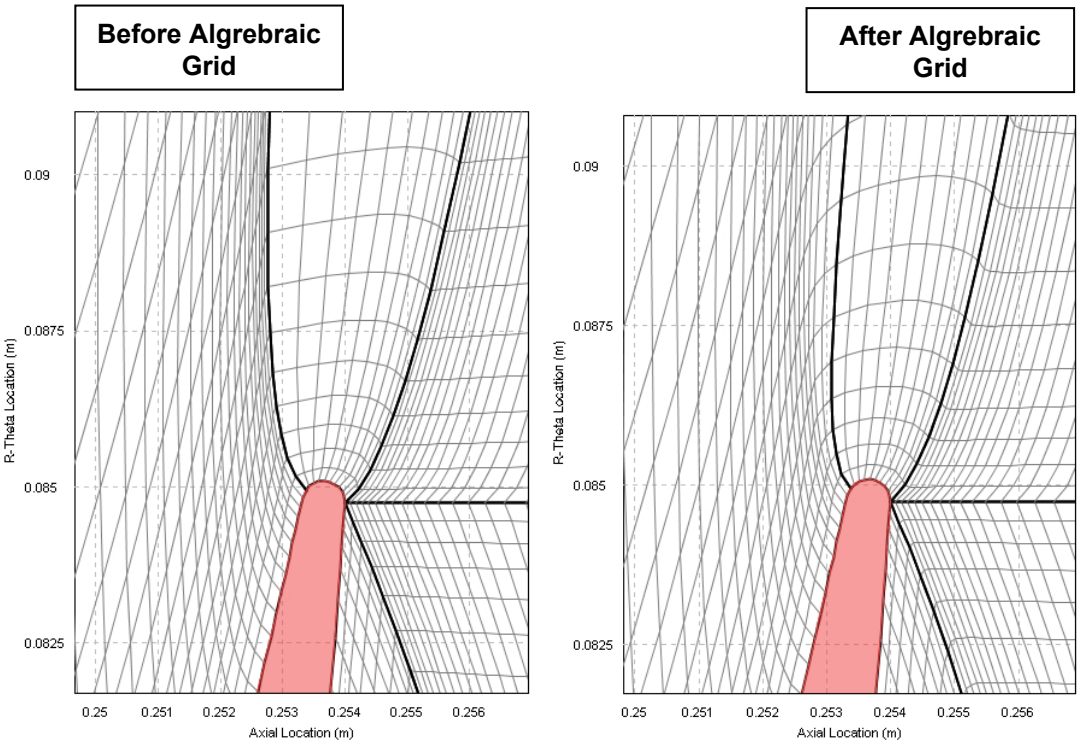

Figure 5: Application of Algebraic Grid Smoothing

3.3 Assessment of Grid Quality

In order to asses the quality of mesh, the evaluation and presentation of grid quality metrics has been introduced in the grid generator. These metrics categorise cells as “good”, “acceptable” or “poor”, in terms of grid skewness and grid aspect ratio. Contour plots are also provided in order to help the user locate the poorest regions of the mesh. The user interface to the grid quality metrics is shown in Figure 6.

The grid quality metrics are intended only to serve as a guide and help the user locate particularly poor regions of mesh. The user should, however, understand which regions of the mesh are most crucial to the CFD analysis, since it is often better to trade poor grid in these crucial regions for worse grid in regions of the flow where gradients are low.