44th AIAA Aerospace Sciences Meeting and Exhibit
9 - 12 January 2006, Reno, Nevada

AIAA 2006-421

Unsteady Turbomachinery Computations Using Massively Parallel Platforms

Edwin van der Weide [*], Georgi Kalitzin [†], Jörg Schlüter [‡] and Juan J. Alonso [§]
Stanford University, Stanford, CA 94305-4035

The CPU power offered by the latest generation of supercomputers is such that the unsteady simulation of the full wheel of a compressor or turbine is now within reach. This CPU power is mainly obtained by increasing the number of processors beyond several thousands, e.g. the BlueGene/L computer of Lawrence Livermore National Laboratory has approximately 130,000 processors. Consequently extreme care must be taken when the simulation codes are ported to these platforms.

This paper discusses the computer scientific aspects of simulating unsteady turbomachinery flows on massively parallel systems when multi-block structured grids are used. Load balance, parallel IO and search algorithms are addressed, especially for the case where the number of processors is larger than the number of blocks, i.e. when blocks must be split during runtime. Preliminary results for cases with more than 200 million nodes running on 1,800 processors are presented.

I. Introduction

The compressor and turbine of a modern turbofan engine typically have two counter-rotating concentric shafts to allow for different rotational speeds of their components as well as a reduction of net torque. The outer, low pressure parts rotate at a lower rate than the inner, high pressure components. Typical rotation rates are 5,000 to 7,000 RPM for the former and 15,000 to 20,000 RPM for the latter. The compressor and turbine themselves consist of a series of rotors and stators for which the blade counts are normally chosen such that no sector periodicity occurs. Combined with the inherently unsteady nature of turbomachinery flows, due to the motion of the rotors, this means that for the numerical simulation of the true problem the full wheel must be computed time accurately.

The computational requirements for such a simulation are severe. Only consider the high pressure components of a typical aircraft jet engine. The high pressure compressor, HPC, typically consists of 5 stages (rotor/stator combinations) and 100 to 200 blade passages per stage. Since approximately a million nodes are required per blade passage to obtain a grid-converged Reynold-Averaged Navier-Stokes (RANS) solution, the computational mesh for a full wheel HPC simulation should contain 500 million to 1 billion nodes. Due to the favorable pressure gradient in the turbine, the blades can be loaded more heavily thus reducing the number of stages. However a full wheel high pressure turbine, HPT, simulation still requires 150 to 300 million nodes.

The spatial mesh is to be integrated in time for 2,000 to 10,000 time steps, based on the estimate that 50 to 100 time steps are needed to resolve a blade passing, to remove the transient effects. As the corresponding physical time step leads to a CFL number of O(10,000) for the smallest cells in the grid, an implicit time integration scheme is most efficient. In this work the second order backward difference formula,

$$\frac{3U^n - 4U^{n-1} + U^{n-2}}{2\Delta t} = \text{Res}(U^n), \tag{1}$$

[*] Research Associate, Department of Aeronautics and Astronautics, AIAA Member
[†] Research Associate, Center for Turbulence Research, AIAA Member
[‡] Research Associate, Center for Turbulence Research, AIAA Member, current address: Nanyang Technical University, School of Mechanical and Aerospace Engineering, Aerospace Department, 50 Nanyang Ave, Singapore 639798
[§] Associate Professor, Department of Aeronautics and Astronautics, AIAA Member

where $\mathrm{Res}(U^n)$ is the residual of the spatial discretization, is used. The resulting nonlinear system for the state U^n is solved using the dual time-stepping approach.[1] Using this technique it is expected that for a full wheel unsteady HPC simulation 1 to 10 million CPU hours are needed on today's fastest computers. For an HPT simulation this number scales according to the ratio mentioned above. If the low pressure components are added these numbers roughly quadruple, because both the computational mesh and the number of time steps double. These requirements are far beyond what is currently affordable for practical applications and therefore approximations are used to reduce the computational costs.

The most widely-used industrial practice for solving turbomachinery problems is the mixing plane assumption.[2] A circumferential averaging of the flow variables is applied at the interface between rotor and stator. These average quantities are then imposed as upstream and downstream values for the following and preceding blade rows respectively and a steady-state computation can be performed for both the rotor and the stator. Due to this averaging and the periodicity assumption only one blade passage needs to be simulated per blade row, independently of the blade counts. For the above HPC example the mixing plane assumption reduces the problem to a steady-state computation on a 10 million node mesh.

Although this assumption models the mean effect of the rotor/stator interaction, all the unsteady information is lost due to the averaging. Especially for the high pressure components the unsteady effects can be important. Hence it is desirable to obtain an estimate of these effects. The current approach to accomplish this is to perform an unsteady simulation of one stage, usually combined with rescaling. The upstream and downstream boundary conditions are obtained from the mixing plane solution. Rescaling means that the blade counts are changed such that a periodic sector is obtained, which then can be simulated instead of the full wheel. To maintain solidity also the shape of the airfoils must be altered, which means an additional geometry modification. Because of these changes it is clear that only approximate information can be obtained from unsteady sector simulations. Alternative approaches include the use of phase-lagged boundary conditions,[3,4] which account for first order unsteady effects, and the deterministic stress approach.[5] Both techniques use simplifying assumptions and again only an approximate solution of the true problem is obtained.

Therefore it is realized that, although too long for practical design at present, a wealth of information can be extracted from full wheel unsteady analyses. They will lead to better understanding of the detailed interactions between blade rows and therefore to improved stand-alone component analyses. Figure 1 shows

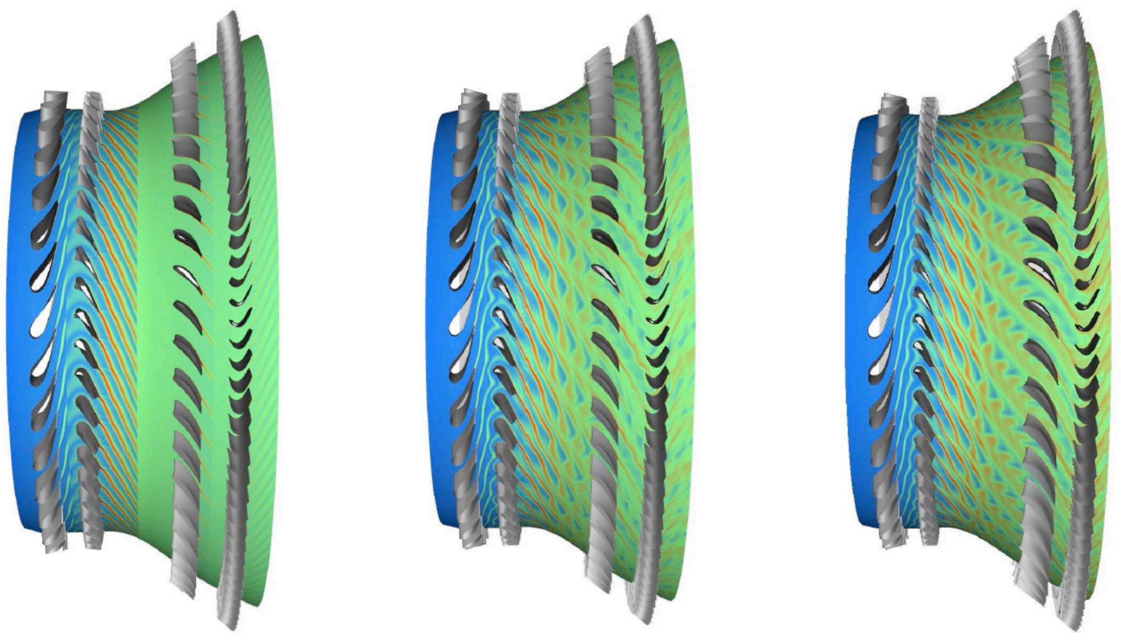

Figure 1. Entropy distributions for simulations of the first two stages of a modern turbine. Steady solution with the mixing plane approximation (left), unsteady solution for a 20^o sector with scaled geometry (middle) and unsteady solution for the full wheel (right). For the unsteady solutions an instantaneous distribution is shown.

a comparison between the mixing plane assumption, the unsteady simulation of a scaled geometry (the scaling is such that periodicity is obtained for a 20^o sector) and the unsteady simulation of the full wheel of the first two stages of a turbine. The quantity displayed is entropy on a surface located half-way the hub and the case. For the unsteady simulations an instantaneous distribution is shown. The difference between the steady computation using the mixing plane assumption and the unsteady simulations is evident. The unsteady nature of the flowfield combined with the different blade counts leads to a strong interaction (both downstream and upstream) between the rotor and stator. This information is lost due to the circumferential averaging and results in a completely different picture of the flow field. The difference between both unsteady simulations is not as clear, but subtle differences can be distinguished. Due to the rescaling of the blade counts in the 20^o sector simulation the wake patterns of the preceding blade rows differ from the true geometry and hence the interaction between the blade rows shows different frequencies.

The objective of the Stanford ASC project[6] is to develop a framework able to perform multi-disciplinary simulations on massively parallel platforms. This framework can be used to compute the flowpath through an entire jet engine (see figure 2). A major milestone for such an integrated computation is the unsteady simulation of the individual components, the compressor, the combustor and the turbine. This paper focuses on the turbomachinery components of the engine and presents the the computer scientific aspects of performing unsteady simulations of entire compressors and turbines on massively parallel platforms (MPP's) as well as some preliminary results. Details of the computational framework needed for an integrated simulation of the full engine can be found in.[7,8]

II. Parallel algorithms

Despite the progress made in unstructured grid technology during the last 10 years, the quality of solutions obtained on structured grids is still superior compared to their unstructured counterparts. This is especially true for high Reynolds number RANS simulations where high aspect ratio cells must be used to capture the anisotropic flow phenomena. In combination with the fact that it is relatively straightforward to create multi-block structured grids for the geometries used in the turbomachinery components of a jet engine, all the compressor and turbine simulations carried out within the Stanford ASC program use multi-block structured grids.

The flow solver performing these simulations is SUmb, which has been developed under the sponsorship of the Department of Energy Advanced Strategic Computing (ASC) Initiative. SUmb solves the compressible Euler, laminar Navier-Stokes and RANS equations on multi-block structured meshes. Although the primary objective of this code within the Stanford ASC program is to compute the flows in the rotating components

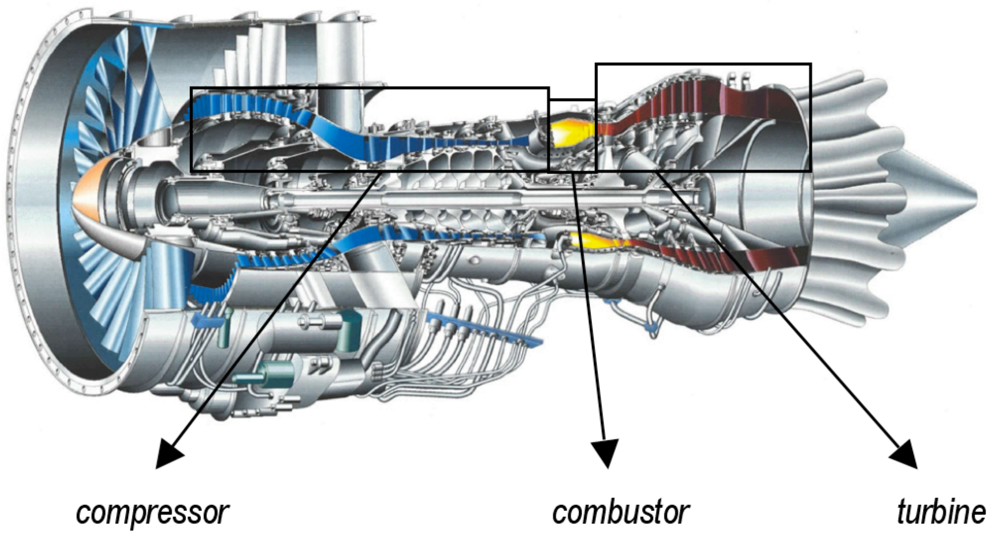

Figure 2. Schematics of an aircraft jet engine.

of jet engines, SUmb has been developed as a generic solver and it is therefore applicable to a variety of other types of problems, including external aerodynamic flows. A Python interface is available when used in a multi-disciplinary environment.[9]

SUmb can be used to solve steady-state problems, unsteady problems (with moving geometries) and time periodic problems, for which a Fourier representation is used for the time derivative leading to a coupled space-time problem.[10,11] A number of turbulence models is available, including k-ω,[12] Spalart-Allmaras[13] and v^2-f.[14] To reduce the grid resolution requirements in the near wall region it is possible to use adaptive wall functions.[15,16]

For the discretization of the inviscid fluxes either a central difference scheme augmented with artificial dissipation[17] or an upwind scheme in combination with Roe's approximate Riemann solver[18] is used. The viscous fluxes are computed using a central discretization. All the results presented in this paper are obtained with a second order cell-centered discretization. The second order implicit time integration scheme, equation (1), is used for all unsteady computations. The convergence is accelerated via a standard geometrical multi-grid algorithm in combination with an explicit multi-stage Runge-Kutta scheme.

SUmb is a parallel code, suited for running on massively parallel platforms. In the following sections the load balancing, parallel IO and parallel search algorithms will be discussed in more detail.

A. Load balance

A critical issue to obtain good scalability on machines with thousands of processors is the load balance. The grid must be distributed such that every processor performs the same amount of work while the cost of communication is minimized. This is a typical graph partitioning problem. The complication that arises for multi-block structured meshes is that the number of blocks in the grid can be similar or even smaller than the number of processors used. Consequently it is necessary to allow the automatic splitting of blocks during runtime such that an equal load can be obtained. At the moment a segregated approach is used, which is shown in figure 3. The first step is to apply the graph partitioner on the original mesh for which a certain load imbalance is obtained. If this imbalance exceeds the user specified tolerance, normally 5 to 10 percent, the largest blocks in the mesh are split and the graph partitioner is applied again. This process is repeated until either an acceptable load imbalance is obtained or the prescribed number of iterations is exceeded. Due to the segregated nature of this algorithm and the rather heuristic way the blocks are split, only suboptimal results can be expected. However, the algorithm is fairly robust if the modifications to the graph partitioner mentioned below are considered.

The graph partitioner used in this work is Metis.[19] Metis is intended for the distribution of unstructured grids for which the number of entities to be partitioned is orders of magnitude larger than the number of processors. Consequently it performs well if the number of blocks is significantly larger than the number of

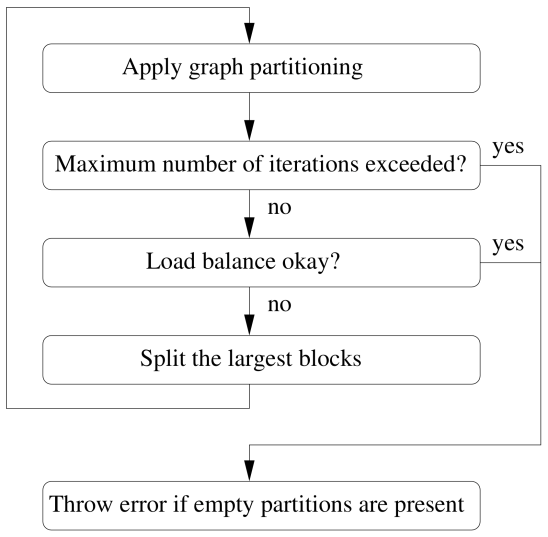
Apply graph partitioning
Maximum number of iterations exceeded?
yes
no
Load balance okay?
yes
no
Split the largest blocks
Throw error if empty partitions are present

Figure 3. Iterative load balance algorithm for multi-block grids.

processors.

However when the number of blocks is just marginally larger than the number of processors, a case that usually occurs when running a multi-block solver on a massively parallel machine, the assumptions used in the algorithms in Metis are violated and problems can be expected. Indeed, experience has shown that for more than 50 processors empty partitions can occur and for $O(1{,}000)$ processors approximately 10 percent of the partitions are empty, which is clearly undesirable. However, it was found that Metis can still be used for these cases if the cost of communication is neglected, i.e. the number of neighbors is set to zero for every vertex of the graph. This is not the best solution possible, but for the cases shown in section III it works reasonably well and better than other solutions attempted, such as Greedy algorithms.

Figure 4 shows the speed up relative to 200 processors for a full wheel turbine simulation for both single-grid and multi-grid (using a three level W-cycle) computations. The grid consists of 496 blocks and 88 million cells and is considered coarse as the average number of cells per blade passage is only 250,000. This case was run on 200, 400, 600, 800, 1,200 and 1,800 processors on the ALC Linux cluster at the Lawrence Livermore National Laboratory. The speed up is perfect as long as an acceptable load balance could be obtained. The deviations from the linear curve in figure 4 are solely caused by load imbalance. It was found that on systems like ALC a linear speed up was obtained as long as at least 20,000 cells are stored per processor. For BlueGene/L, which has relatively slow processors compared to its communication network, this number could be reduced to 5,000.

B. Parallel IO

The SUmb flow solver has IO filters for both CGNS[20,21] and PLOT3D.[22] Due to the large amount of data that must be read and written for unsteady simulations of entire compressors and turbines, parallel IO is an absolute necessity. Despite the recent efforts to create a parallel IO option for CGNS,[23,24] this is not applicable yet to the problems considered in this work and therefore the discussion in this section will be limited to the PLOT3D format.

One possibility for parallel IO is that every processors reads/writes the information it is responsible for. The advantage of this approach is that the method is local, i.e. no explicit communication is required, and therefore easy to implement. If blocks are not split during runtime this indeed is the most efficient solution, because the PLOT3D format stores the information contiguously on a block by block basis[22] and the size of a block is usually big enough to guarantee good performance of the MPI IO routines.

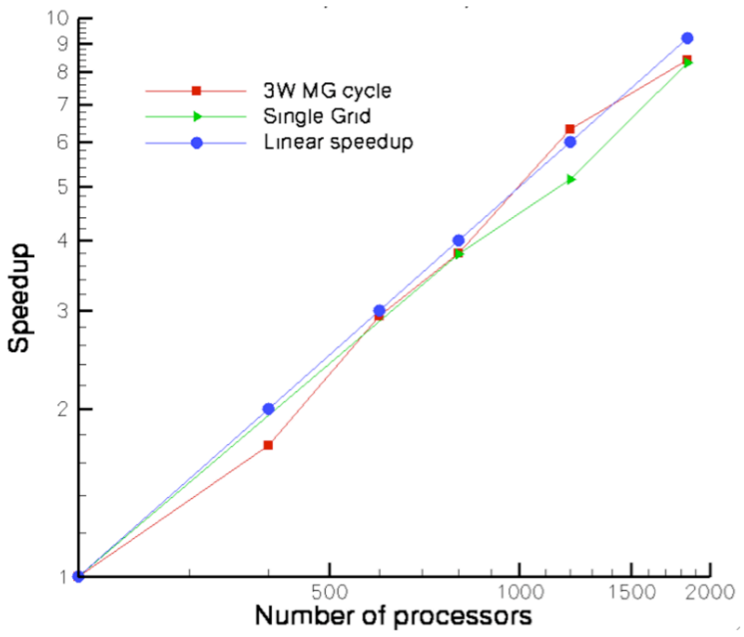

Figure 4. Speed up relative to 200 processors for a full wheel turbine simulation for both a single-grid and a 3W multi-grid cycle. The grid consists of 496 blocks and 88 million cells. The computation was performed on the LLNL ALC Linux cluster.

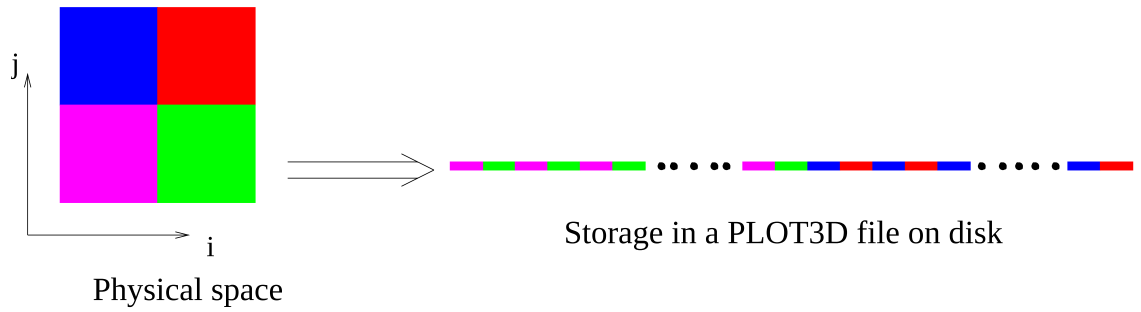

Figure 5. Splitting of a computational block into 4 subblocks. Left the physical situation, right the storage in a PLOT3D file on disk.

However the splitting of blocks needed to obtain a good load balance (see section A) complicates the situation significantly. Consider a block that is split into four subblocks (see figure 5). The data to be read/written for a subblock is not stored contiguously anymore and consequently the algorithm described above leads to very many, small IO-requests. This is known to result in very poor performance.[25] In three space dimensions the situation is even more pronounced because of the additional dimension.

One possible solution to this problem is to use MPI derived data types, which can be considered as a non-contiguous data layout for the many requests, in combination with collective IO. Using this approach the MPI IO implementation is able to optimize the reading and writing significantly.[26] This derived data type IO paradigm was implemented in SUmb and tested on the LLNL ALC Linux cluster. This machine is equipped with the Lustre parallel file system with a theoretical performance of 6.1 to 8.0 GBytes/s. For a relatively small number of processors, 100 or less, good performance was obtained. However for a large number of processors the situation changes drastically. Figure 6 shows the IO performance of the same test case used for the scalability test, see figure 4. For this case the file size for reading was 2.1 GBytes and the file size for writing was 10 Gbytes. Similar results as shown in figure 6 were obtained for different file sizes. It is clear that beyond 400 processors no significant speed up in IO is obtained anymore. Also the actual numbers, 300 MBytes/s for reading and 150 MBytes/s for writing, are somewhat disappointing.

Two additional complications occur when using derived data types for IO. Firstly, when a block is split a processor can store at most one subblock, because the MPI IO implementation assumes that derived data types, i.e. the subblocks, stored on the same processor cannot physically overlap in the file. This puts an

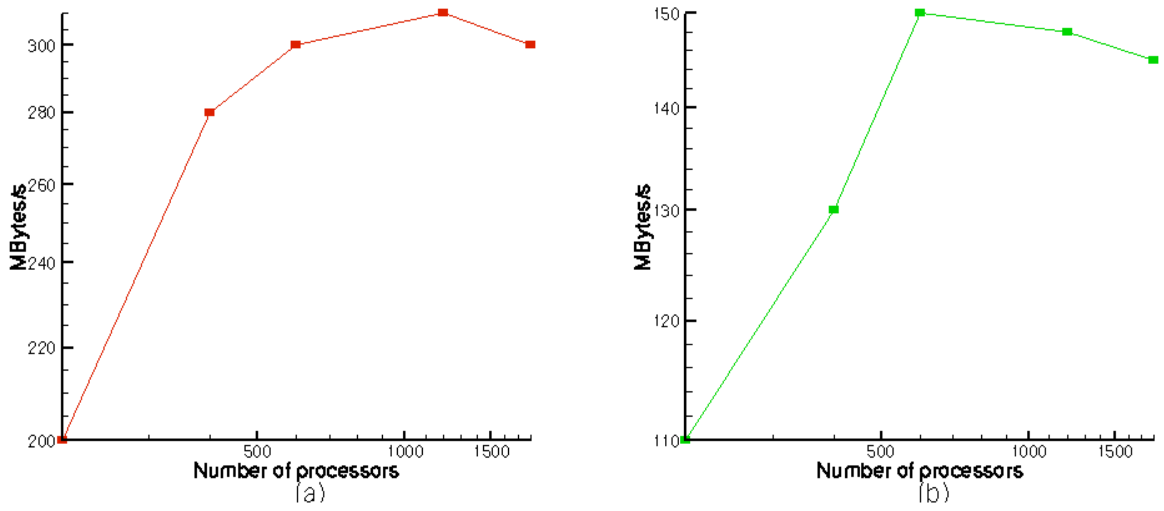

Figure 6. IO performance on the LLNL ALC Linux cluster using derived data types for a full wheel high pressure turbine simulation. (a) Reading performance, (b) Writing performance.

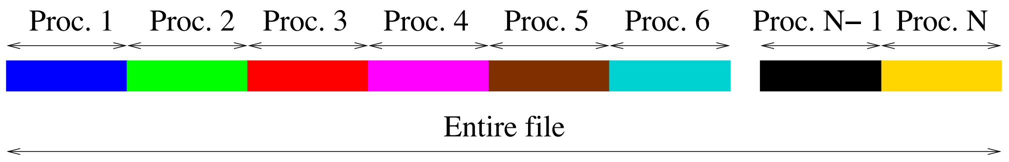

Figure 7. Splitting of the file in equally sized chunks for the currently used parallel IO implementation.

additional constraint on the load balancing algorithm (see section A) which usually cannot be met. The second complication arises for file sizes larger than 2 GBytes. During the construction of the derived data type for collective IO the MPI function *MPI_Type_struct* must be used. The offset argument of this function is a 4 byte integer and consequently files larger than 2 Gbytes cannot be handled without modification.

This modification consists of splitting the IO in chunks of at most 2 GBytes, which also solves the problem of multiple subblocks per processor. However this leads to a very complicated piece of code, which does not perform very well for a large number of processors, see figure 6. Therefore it was decided to abandon the approach using derived data types, also because of robustness issues.

Instead a much simpler algorithm is currently used, see figure 7. The entire file is treated as one big chunk of data, which is equally chopped up in a number of chunks. This number of chunks equals the number of processors used in the simulation. Every processor is responsible for the reading/writing of one chunk and data is communicated to/from other processors, if needed. This approach avoids all the problems with derived data types at the expense of increased communication costs. However it can be expected that these costs are minor compared to the costs of the actual IO.

The implementation of this paradigm was only finished recently and detailed performance tests have not been carried out yet. However initial results indeed show better performance than results obtained with the derived data type approach.

C. Parallel ADT search algorithm

Geometric searches are an important part of the unsteady simulation of turbomachinery flows. Wall distances may be needed for the RANS turbulence model and interpolation coefficients must be determined for the sliding mesh interfaces between rotor and stator. Furthermore geometric searches are also needed for the domain interfaces used for an integrated simulation.[7,8] Moreover, due to the relative motion of the different parts, these interpolations must be repeated every time step, at least for the sliding mesh interfaces. Hence an efficient search algorithm, whose local memory usage is limited, is a prerequisite. The reason for the memory requirement is that the total amount of memory available on modern MPP's is very large, but that is caused by the large number of processors ($O(10,000)$); per processor only 256 MBytes may be available (as in BlueGene/L).

Therefore a parallel version of the Alternating Digital Tree, ADT, algorithm[27] is used. First the bounding boxes of the elements of the local grid are created. These bounding boxes are characterized by the lower left and the upper right coordinate and are therefore equivalent to a point in 6 space dimensions. Every processor creates a local 6-dimensional ADT of the bounding boxes (see figure 8). Logical rather than geometric splits

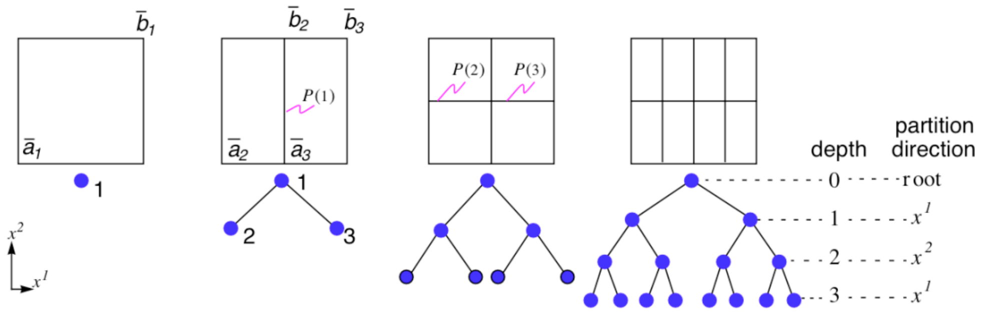

Figure 8. Building of the local ADT. Logical rather than geometrical splits are used to avoid degenerate trees. Figure taken from.[27]

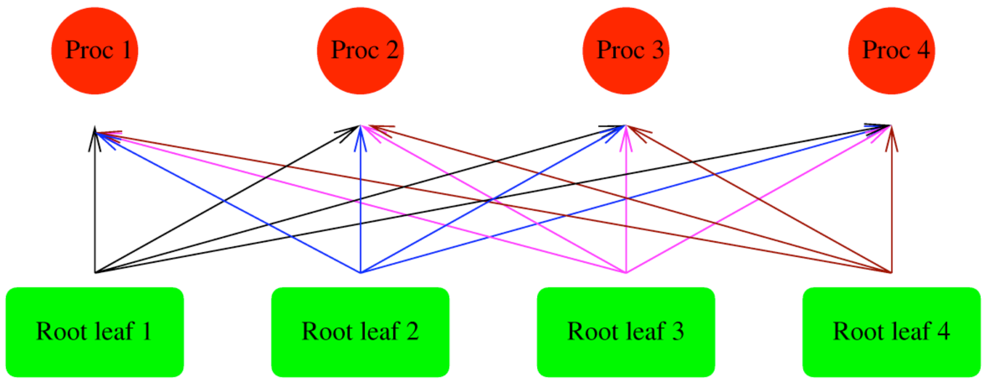

Figure 9. Gathering of the root leaves on all processors.

are used to avoid degenerate trees. The root leaves of the local trees are made available to all processors, see figure 9, such that the geometric range covered by each local tree is known on every processor. Hence possible target trees can be determined for every point to be searched.

The global algorithm proceeds as follows

- Determine for every point to be searched the target trees and determine the number of local trees to be searched for the current set of points.
- Make this number available to all processors $\Longrightarrow$ Every processor can determine the total number of points it has to search in the local tree.
- Determine the number of search rounds to avoid a possible memory bottleneck.
- Determine the sending and receiving processors for every search round.
- Loop over the search rounds
  - MPI_Alltoall to request data from processors.
  - Send coordinates to be searched to the target processors.
  - Perform the interpolation in the local tree to overlap between communication and computation.
  - Loop over the number of processors for which I need to interpolate data in my local tree
    - Receive the coordinates.
    - Interpolate the data in my local tree.
    - Send interpolation data back to the requesting processor.
  - Loop over the number of processors to which I sent coordinates.
    - Receive the interpolation data.

This algorithm works well when the global tree is evenly distributed over the processors, e.g. volume or sliding mesh interpolation. When this is not the case, e.g. wall distance computation, the load imbalance can be severe and a dramatic decrease in performance is observed. To improve this situation migration of data combined with duplication of trees should be considered.

III. Results

This section presents some preliminary results for both the turbine and compressor of a typical aircraft jet-engine. The tip gap regions of the rotors have been neglected and therefore shrouded casings are assumed. Both the compressor and the turbine grids have the same topology consisting of an O-grid around the blade and an H-type grid in the passage, see figure 10. The initial spacing for the turbine grids corresponds to an average y^+ value of 60, thereby requiring the use of wall functions.[15,16] The total number of cells in a passage, i.e. the O and H-block in figure 10, is 356,352. For the compressor grid the number of cells

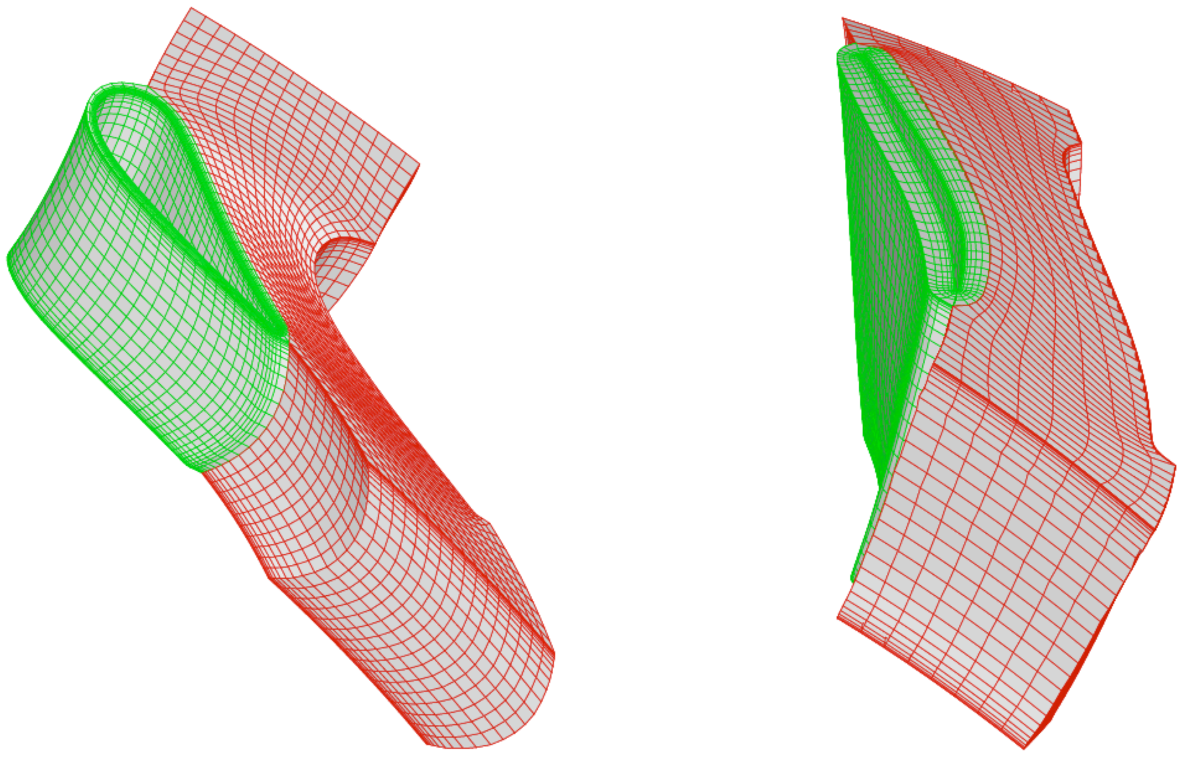

Figure 10. Grid topologies for a turbine passage (left) and compressor passage (right). Coarser versions of actual grids are shown.

per passage differs per blade row, but on average approximately 280,000 cells per passage are used. As this grid is a wall integration grid, with an average y^+ value of 3, this is clearly too coarse to obtain a grid-converged solution and therefore the compressor simulations should be considered as a demonstration only. The subsonic inflow and outflow boundary conditions correspond to the regular cruise condition of the engine. For all results shown the turbulence is modeled using the k-ω model.[12]

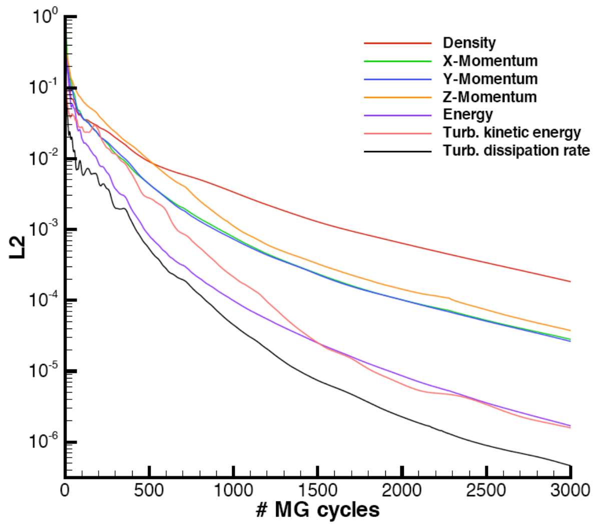

Figure 11. Convergence histories for the turbine, mixing plane assumption. 3W cycle in combination with a 5 stage Runge Kutta smoother.

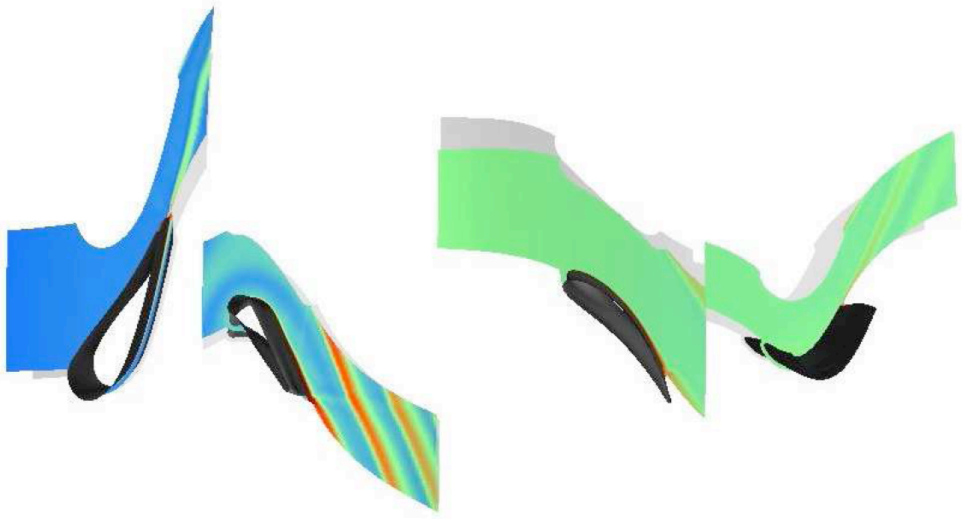

Figure 12. Entropy distribution for the mixing plane computation of the high pressure and first stage of the low pressure turbine.

A. Steady computations

Steady-state computations using the mixing plane assumption are carried out for the high pressure turbine, consisting of 1 stage, combined with the first stage of the low pressure turbine and for the high pressure compressor, which contains 5 stages.

The convergence history for the turbine case is presented in figure 11. Although 3,000 multi-grid cycles were used, the solution reached engineering accuracy after approximately 1,000 cycles. Note that the initial solution on the fine grid was obtained by applying grid sequencing on the coarser meshes. Consequently, the relative initial position for the convergence histories shown in figure 11 is already two orders of magnitude lower than when the solution is initialized from free-stream conditions. The entropy field of a steady-state mixing plane solution in a turbine passage is shown in figure 12. The abrupt disappearance of the wakes between the blade rows is a direct consequence of the one-dimensional interpolation/averaging on the sliding mesh interfaces.

The HPC simulation is far more challenging to compute due to the high compression ratio that results from the work transmitted from the rotating blades to the fluid. Since the exit pressure is an order of magnitude higher than the pressure at the inlet, the flow tends to reverse its direction as soon as large pockets of separation occur in the passage. As a consequence, the computations must be carried out in several steps, first by slowly raising the rotational speed of the wheels while keeping the pressure low at the exit, and then by increasing the pressure once the full rotational speed was achieved. The entropy field of a steady-state mixing plane solution in a compressor passage is shown in figure 13.

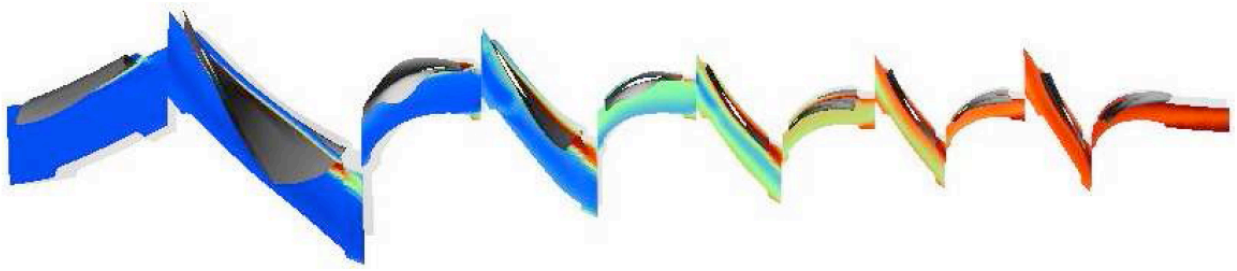

Figure 13. Entropy distribution for the mixing plane computation of the high pressure compressor.

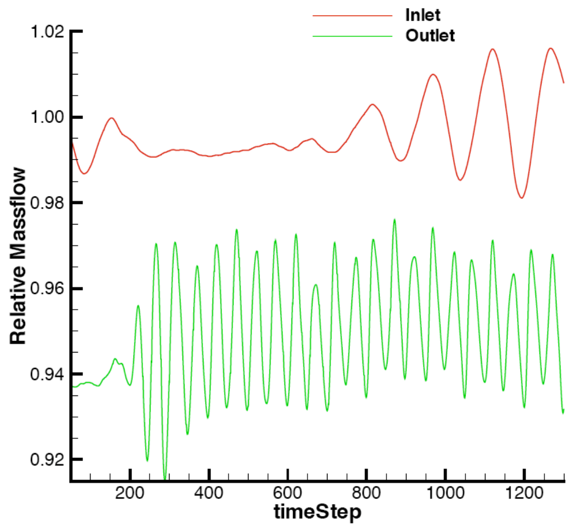

Figure 14. Evolution of the relative inlet and outlet mass flow for the scaled turbine geometry.

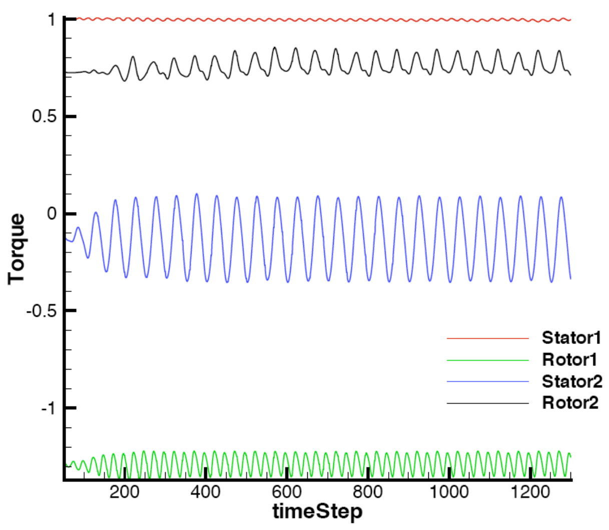

Figure 15. Evolution of the relative torque on the four blade rows for the scaled turbine geometry.

B. Unsteady computations of 20^o sectors

For the 20^o sector simulations the blade counts are changed such that the full wheel can be split into 18 sections and periodicity conditions can be used. The pitch and chord of the blades are adjusted to preserve the flow blockage. The steady solutions obtained with the mixing plane assumption are used as initial conditions for the unsteady computations. The physical time step is chosen such that a blade passage of the blade row with the highest blade count is resolved with 50 time steps. For the turbine this corresponds to 2,700 time steps per revolution, while for the compressor 6,300 time steps per revolution must be taken.

The 20^o turbine sector computation has been advanced for 1,300 time steps. Transient effects are still present in the flow, as illustrated by time evolutions of the relative inlet and outlet mass flow rate, see figure 14. However, the torque on the blades seems less sensitive and has reached a seemingly periodic state, figure 15. Figures 16 and 17 show the mass flow evolution during the last 400 time steps compared to mixing plane solutions computed using both the scaled and unscaled geometry. First, it can be seen that all

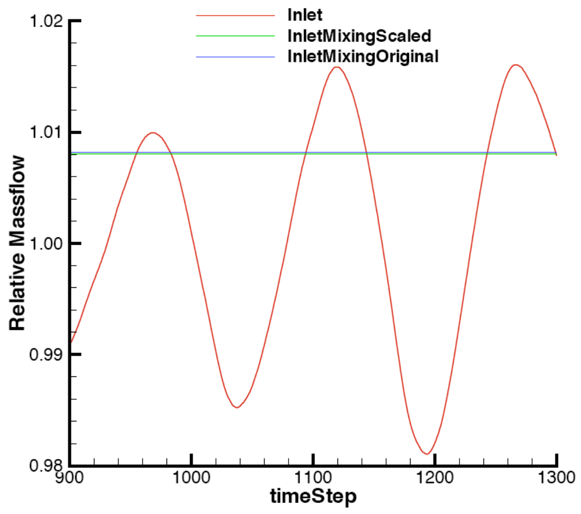

Figure 16. Comparison of the relative inlet mass flow of the unsteady solution and the two mixing plane solutions.

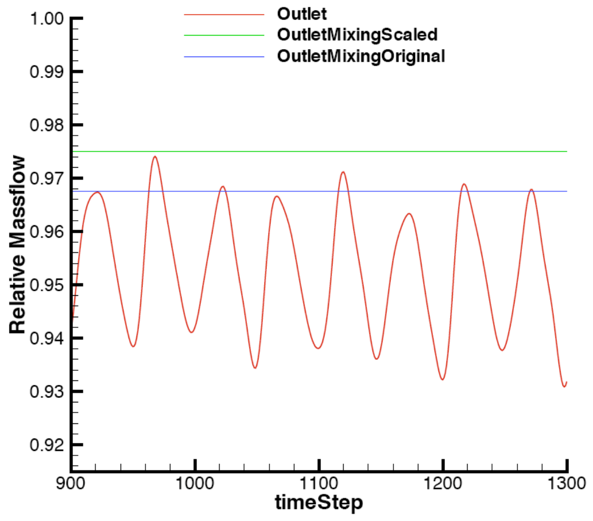

Figure 17. Comparison of the relative outlet mass flow of the unsteady solution and the two mixing plane solutions.

solutions have a lower mass flow rate at the outlet than at the inlet, despite the use of a conservative scheme. In a conservative scheme only the sum of the central and dissipative fluxes is conserved. The values shown in figures 14, 16 and 17 correspond to the central contribution, which is not conserved as such. The difference in mass flow is 3 to 4 percent, indicating that the solutions cannot be considered grid-converged. This result was anticipated, but the decision was made to proceed with these grids in order to learn some initial lessons about the unsteady nature of these large-scale computations at somewhat lower computational cost. Secondly, the average mass flow for the unsteady computation is lower than for the mixing plane solutions. Although this could still be a transient effect, figure 14 indicates that this is not the case. Finally, the unsteady mass flow rate at the outlet, figure 17, shows a much higher frequency than the mass flow rate at the inlet, figure 16. The flow at the outlet is influenced by all blade rows, while the flow at the inlet seems to be influenced only by the first blade row.

Figures 18 to 21 show the relative torque on the blade rows for the unsteady solution of the last 400 time steps, its time-averaged values and for the mixing plane solutions.

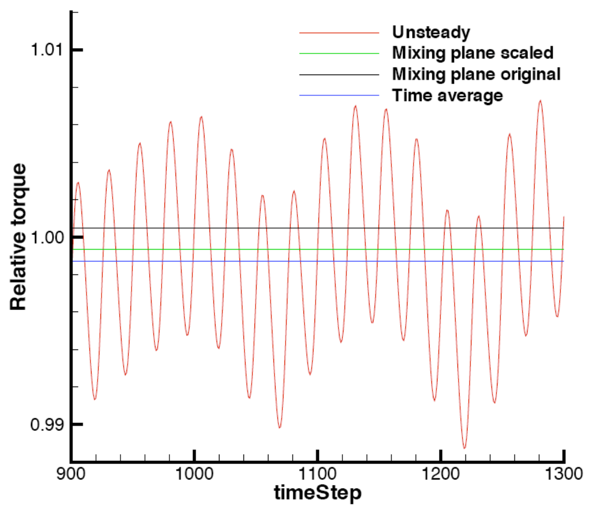

Figure 18. Comparison of the relative torque on the first stator of the unsteady solution and the two mixing plane solutions.

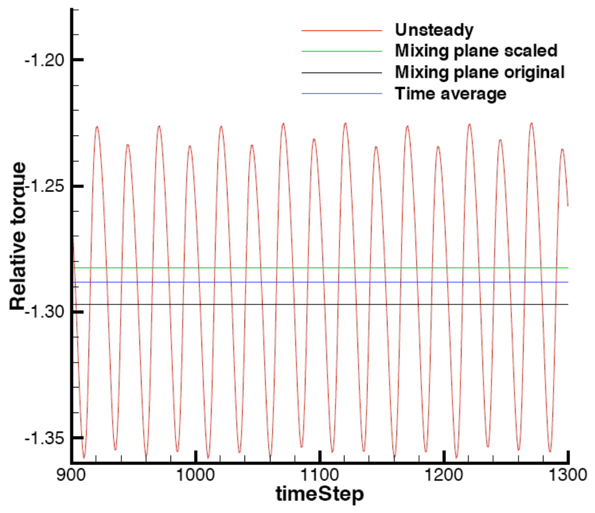

Figure 19. Comparison of the relative torque on the first rotor of the unsteady solution and the two mixing plane solutions.

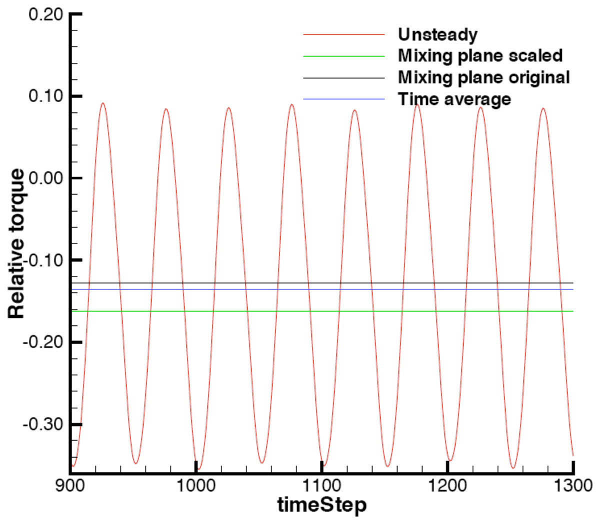

Figure 20. Comparison of the relative torque on the second stator of the unsteady solution and the two mixing plane solutions.

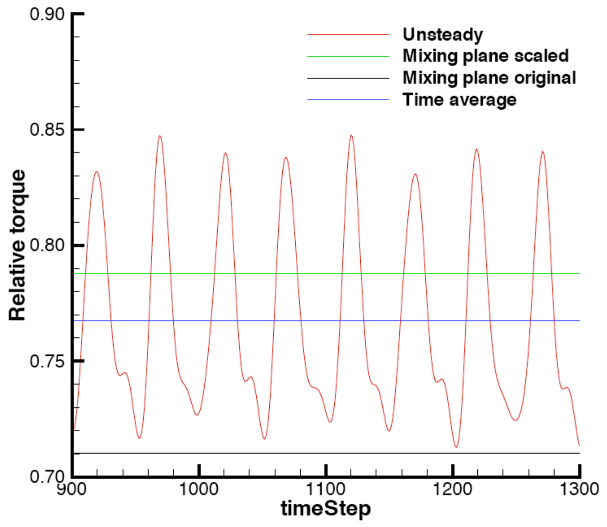

Figure 21. Comparison of the relative torque on the second rotor of the unsteady solution and the two mixing plane solutions.

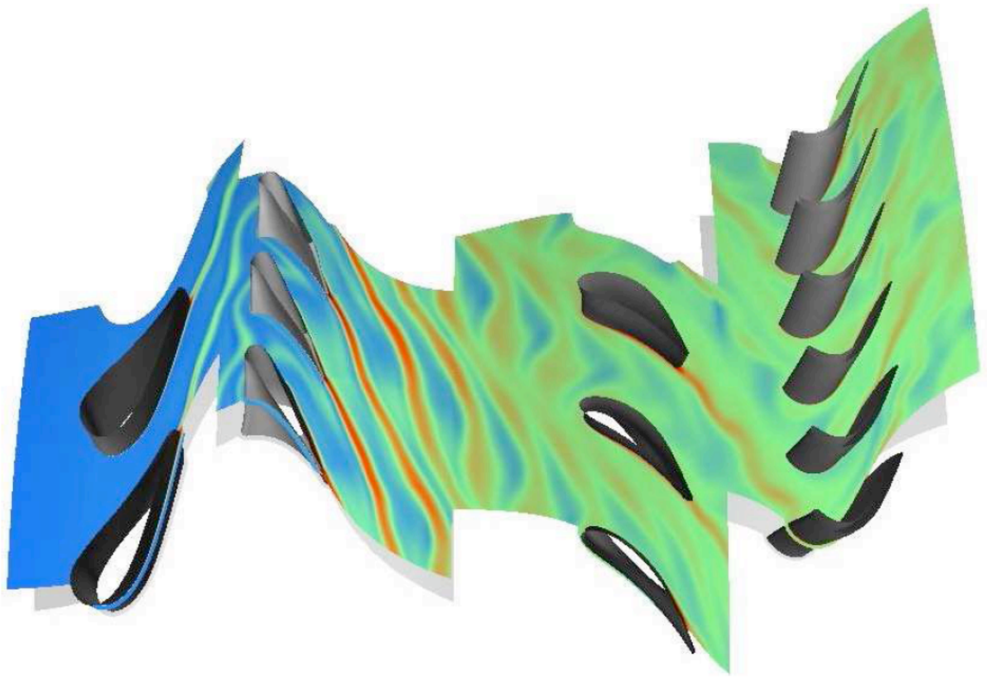

Figure 22. Turbine, 20^o sector; Instantaneous entropy distribution for the unsteady solution for the scaled geometry.

The difference between the value obtained with the mixing plane assumption and the time-averaged value is between 1 and 3 percent. However, due to the steady nature of the mixing plane assumption, it does not give any information about the amplitude and frequencies of the oscillations, which are important design properties.

Figure 22 shows the instantaneous entropy distribution in the same plane as figure 12. It is clear that the solution is now continuous over the sliding mesh interface, and the effect of the wakes on the downstream blades is evident.

For the 20^o compressor sector computation only 1,500 time steps have been computed, which corresponds to 85^o of rotation, starting from the mixing plane solution. This is not enough to reach the periodic state and therefore no detailed comparisons are shown. Figure 23 presents the instantaneous entropy distribution after 1,500 time steps. Again the interaction between the blade rows is clearly visible compared to the mixing plane solution in figure 13.

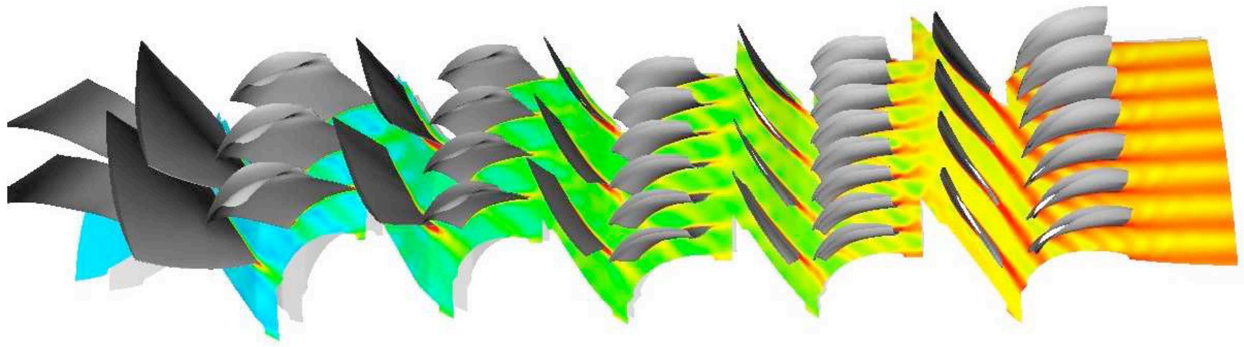

Figure 23. Compressor, 20^o sector; Instantaneous entropy distribution for the unsteady solution for the scaled geometry.

C. Unsteady computations of full-wheel geometries

Due to the smaller number of blade rows in the HPT, the computational costs for the unsteady simulation of the full wheel turbine are smaller than for the compressor. Consequently most of the attention for the full wheel simulations has been paid to the high pressure turbine augmented with one stage of the low pressure part. The computational grid consist of 496 blocks and 88 million cells. The disk space needed to store this grid in double precision is 2.1 GBytes. For the solution file at least 6 Gbytes of disk space is needed to store the set of independent variables. However, if more information is stored in the solution file this number increases. As for the 20^o sector simulation one revolution of the HPT is resolved with 2,700 time steps.

Consequently disk storage space becomes an issue for these full wheel computations. If a solution is written every time step at least $8 \times 2,700 = 21,600$ Gbytes = 21 TBytes are needed for one revolution of the turbine simulation. The question is whether to store the whole solution every time step or only to dump *a priori* chosen planes. The advantage of the latter approach is that the disk space requirements are reduced by at least one order of magnitude. The disadvantage is that some *a priori* knowledge of the solution is needed.

These numbers are based on a grid containing 88 million cells, i.e. approximately 300,000 per blade passage. In section B it has been shown that this grid resolution is not sufficient to obtain a grid-converged solution. It is estimated that at least a million cells are needed per passage for a grid converged solution if the turbulence model is integrated up to the wall (no wall functions). The corresponding disk space requirements increase by a factor 3 as well, under the assumption that 2,700 time steps for a full revolution are still enough to obtain a time converged solution.

This case has been run on the LLNL ALC machine on either 300, 600, 1,200 or 1,800 processors, depending on the available resources; SUmb writes the solution in a single file and a restart can be made on a different number of processors due to the fully integrated parallel preprocessor. The solution has been advanced 600 time steps starting from the mixing plane solution. A comparison with the mixing plane and unsteady 20^o sector simulation is shown in figure 1.

The full wheel HPC simulation has been initiated as well. The computational grid consists of approximately 220 million cells and requires 5.7 GBytes of disk space. The size of a corresponding solution file is at least 17 Gbytes, which leads to 23 Gbytes for grid and solution. Due to the higher blade counts in the compressor, compared to the turbine, 7,000 time steps are needed per revolution, which still corresponds to

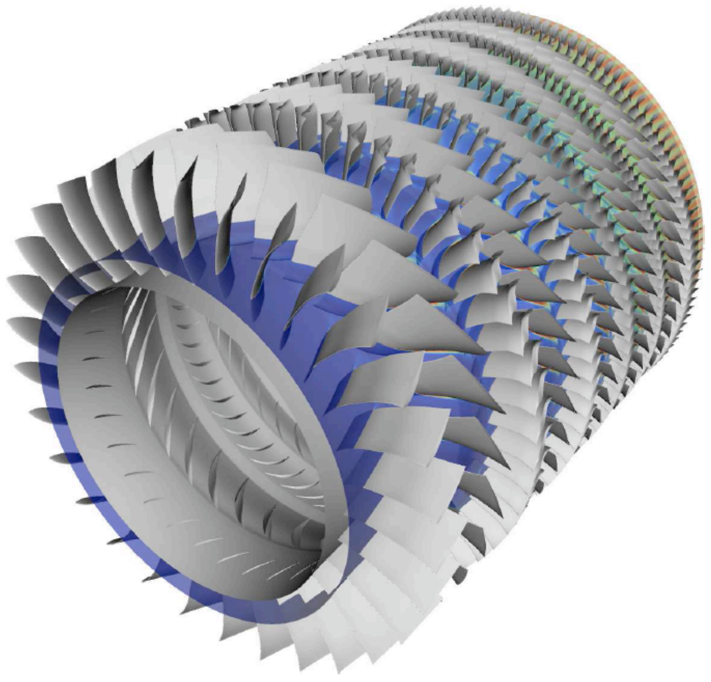

Figure 24. Instantaneous entropy distribution for the full wheel PW 6000 high pressure compressor simulation after 40 time steps, i.e. 2^o rotation of the HPC rotor.

50 time steps per blade passage of blade row with the most blades. Consequently 155 TBytes of disk space is required to store one revolution of the full wheel HPC for this grid.

Only 40 time steps, i.e. 2^o of rotation, have been computed, see figure 24, which took 20 hours of wall clock time on 600 processors on the LLNL ALC machine. Clearly more computational resources are needed for a case this large.

IV. Conclusions

This paper describes the parallel algorithms required for the efficient unsteady simulation of large scale turbomachinery problems on massively parallel platforms using multi-block structured grids. Load balancing, parallel IO and the parallel ADT search algorithm have been discussed. It is shown that load balance is absolutely essential to obtain good speed up and also that this is not a trivial task for multi-block structured meshes. For the cases presented in this paper blocks had to be split for this purpose.

A consequence of this splitting during runtime is that the parallel IO becomes complicated. To avoid very many, small requests the IO cannot be done locally anymore and an implementation using communication is needed. It was attempted to use MPI derived data types for this purpose, which worked reasonably well up until 200 processors. However, the performance for larger numbers of processors is rather disappointing (on the LLNL ALC machine) and problems arise when the file size exceeds 2 GBytes. Therefore the derived data type paradigm was abandoned. Instead in the currently used algorithm every processor is responsible for a contiguous chunk of data, which is possibly communicated to/from other processors. No performance tests have been conducted for this implementation.

The parallel ADT algorithm is a generalization of the sequential algorithm and is done such that no memory bottleneck occurs. The reason for this is that some of the modern massively parallel machines only have a very limited amount of local memory. Consequently, performance is sacrificed in the current implementation to satisfy the local memory restriction. For containment searches the performance is still very good, but for minimum distance searches, especially wall distances, this is not the case anymore.

Preliminary results of full-wheel unsteady simulations of the high pressure components of an aircraft jet-engine have been presented. For the HPT a grid consisting of 88 million cells is used, for the HPC this number is 220 million cells. Despite these large numbers of cells the grids are too coarse to obtain grid-converged results.

The difference with the mixing plane assumption, which allows for a steady approximation of these inherently unsteady problems, is clearly shown. Also differences with the unsteady solution using a scaled geometry can be observed, although the full wheel simulations are still in too early a stage to quantify these differences.

V. Future Work

It has been shown that the grids used in the full-wheel simulation are too coarse for a grid-converged solution and finer grids are needed. In order to deal with this increased problem size as well as to reduce the wall clock time more processors are needed than the 1,800 used so far. From tests done with the current load balancing algorithm it is clear that it does not give satisfactory results for processor counts of 5,000 and higher. Consequently a different algorithm is needed that should incorporate the block splitting in the partitioning algorithm rather than the segregated approach used in this work.

The parallel IO algorithm for which every processor is responsible for one contiguous chunk of data should be optimal for reading and writing itself. It may lead to increased communication, but it is expected that this is of minor importance. Further tests are needed to verify this expectation.

The efficiency of the parallel ADT search algorithm should be improved, in particular for minimum distance searches. A more flexible approach is needed where either duplication of trees or migration of data should be allowed to improve the efficiency. Also the very strict memory policy currently used could be relaxed, especially for surface data. Tests have indicated that this leads to a factor 3 or 4 increase in performance for wall distance searches.

For the unsteady simulations, both sector and full-wheel, a lot of work remains to be done, especially on verification and validation.

VI. Acknowledgments

This work has benefited from the generous support of the Department of Energy under contract number LLNL B341491 as part of the Advanced Simulation and Computing Program (ASC) program at Stanford University. Furthermore the authors would like to acknowledge support from Saadat Syed, Dilip Prasad and William Sprout of Pratt & Whitney, who provided grids and boundary conditions.

References

[1]Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," *AIAA paper* 91-1596, June 1998.

[2]Denton, J. and Singh, U., "Time Marching Methods for Turbomachinery Flows," *VKI LS 1979-07*, 1979.

[3]Chen, J. and Barter, J., "Comparison of Time-Accurate Calculations for the Unsteady Interaction in Turbomachinery Stage," *AIAA paper 98-3292*, July 1998.

[4]Wang, X. and Chen, J., "A Post-Processor to Render Turbomachinery Flows Using Phase-Lag Simulations," *AIAA paper 04-615*, AIAA 42nd Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2004.

[5]Adamczyk, J., "Aerodynamic analysis of multistage turbomachinery flows in support of aerodynamic design." *ASME-99-GT-80*, 1999.

[6]"Annual ASC Report," http://cits.stanford.edu.

[7]Schlüter, J., Wu, X., van der Weide, E., Hahn, S., Alonso, J., and Pitsch, H., "Multi-Code Simulations: A Generalized Coupling Approach," *AIAA paper 05-4997*, 35th AIAA Fluid Dynamics Conference and Exhibit, Toronto, Ontario, June 2005.

[8]Schlüter, J., Wu, X., van der Weide, E., Hahn, S., and Alonso, J., "Integrated LES- RANS of an Entire High- Spool of a Gas Turbine," *AIAA paper 06-0897*, 36th AIAA Fluid Dynamics Conference and Exhibit, Reno, NV, January 2005.

[9]Alonso, J., LeGresley, P., van der Weide, E., and Martins, J., "pyMDO: A Framework for High-Fidelity Multi-Disciplinary Optimization," *AIAA paper 04-4480*, 10th AIAA/ISSMO Multidisciplany Analysis and Optimization Conference, Albany, NY, August 2004.

[10]Gopinath, A. and Jameson, A., "Time Spectral Method for Periodic Unsteady Computations over Two- and Three-Dimensional Bodies," *AIAA paper 05-1220*, AIAA 43rd Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2005.

[11]van der Weide, E., Gopinath, A., and Jameson, A., "Turbomachinery Applications with the Time Spectral Method," *AIAA paper 05-4905*, 35th AIAA Fluid Dynamics Conference and Exhibit, Toronto, Ontario, June 2005.

[12]Wilcox, D., "Reassesment of the scale-determining equation for advanced turbulence models." *AIAA Journal*, Vol. 26, 1988, pp. 1299–1310.

[13]Spalart, P. and Allmaras, S., "A one-equation turbulence model for aerodynamic flows." *La Recherche Aerospatiale*, Vol. 1, pp. 1–23.

[14]Durbin, P., "Separated flow computations with the k-ε-$\overline{v^2}$ model." *AIAA Journal*, Vol. 33, 1995, pp. 659–664.

[15]Kalitzin, G., Medic, G., Iaccarino, G., and Durbin, P., "Near-wall behavior of RANS turbulence models and implications for wall function," *Journal of Computational Physics*, Vol. 204, 2005, pp. 265–291.

[16]Medic, G., Kalitzin, G., Iaccarino, G., and van der Weide, E., "Adaptive wall functions with applications." Tech. rep., 36th AIAA Fluid Dynamics Conference and Exhibit, 2006.

[17]Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes," *AIAA paper 81-1259*, 1981.

[18]Roe, P., "Fluctuations and Signals - A Framework for Numerical Evolution Problems," *Numerical Methods for Fluid Dynamics*, Academic Press, 1982.

[19]Karypis, G. and Kumar, V., "Multilevel Algorithms for Multi-Constraint Graph Partitioning," http://www-users.cs.umn.edu/ karypis/publications/partitioning.html, 1998.

[20]Poirier, D., Allmaras, S., McCarthy, D., Smith, M., and Enomoto, F., "The CGNS System," *AIAA paper 98-3007*, 1998.

[21]Legensky, S., Edwards, D., Bush, R., Poirier, D., C.L.Rumsey, Cosner, R., and Towne, C. E., "CFD General Notation System (CGNS): Status and Future Directions," *AIAA paper 02-0752*, 2002.

[22]Walatka, P., Buning, P., Pierce, L., and Elson, P., "PLOT3D User's Guide," Tech. rep., NASA, 1990, TM 101067.

[23]Hauser, T., "Parallel I/O for the CGNS system," *AIAA paper 04-1088*, 2004.

[24]Pakalapati, P. and Hauser, T., "Benchmarking Parallel I/O Performance for Computational Fluid Dynamics Applications," *AIAA paper 05-1381*, 2005.

[25]Thakur, R., Gropp, W., and Lusk, E., "An Experimental Evaluation of the Parallel I/O Systems of the IBM SP and Intel Paragon Using a Production Application," *In Proceedings of the 3rd International Conference of the Austrian Center for Parallel Computation (ACPC) with Special Emphasis on Parallel Databases and Parallel I/O, pages 24-35*, September 1996, Lecture Notes in Computer Science 1127. Springer-Verlag.

[26]Thakur, R., Gropp, W., and Lusk, E., "A Case for Using MPI's Derived Datatypes to Improve I/O Performance," *Proceedings of SC98: High Performance Networking and Computing*, 1998.

[27]Aftosmis, M., "Solution Adaptive Cartesian Grid Methids for Aerodynamic Flows with Complex Geometries," *VKI LS 1997-02, Computational Fluid Dynamics*, 1997.