Development and Achievement of NAL Numerical Wind Tunnel (NWT) for CFD Computations

H. Miyoshi

Foundation for Promotion of Material Science and Technology of Japan
1-18-6 Kitami, Setagaya-ku, Tokyo 157, JAPAN

M. Fukuda, T. Iwamiya,
T. Nakamura, M. Tuchiya,
M. Yoshida, K. Yamamoto,
Y. Yamamoto, S. Ogawa,
Y. Matsuo, and T. Yamane

National Aerospace Laboratory
7-44-1 Jindaiji-Higashimachi, Chofu, Tokyo 182, JAPAN

M. Takamura, M. Ikeda,
S. Okada, Y. Sakamoto,
T. Kitamura, and H. Hatama

Fujitu Limited
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, JAPAN

M. Kishimoto

Fujitu Laboratories Limited
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, JAPAN

Abstract

NAL Numerical Wind Tunnel (NWT) is a distributed memory parallel computer developed through joint research and development of NAL and Fujitsu. It is based on the analysis of CFD codes developed in NAL. The target performance is more than 100 times faster than VP400.

In this paper, the parallel computation model employed in the development of the NWT is described. The specification and feature of the NWT and the NWT Fortran are discussed. Finally, Performance evaluations and some applications are presented.

We find that the target performance is attained.

1 Necessity of Numerical Wind Tunnel development

Recent computational fluid dynamics (CFD) enables us to perform practical 3 dimensional viscous flow simulations around aircraft or through compressors and turbines. Simulation results around the design point of aircraft display good agreements quantitatively with experimental data obtained with conventional wind tunnels. 3 dimensional viscous flow simulations are also employed for the development of turbine blades. Physically complex flow simulations are conducted such as reacting flow in combustors, hypersonic flow with real gas effects.

Although CFD research is producing excellent results, there are some problems remained. At first, however we can say that simulation results provide good agreements with experimental data, it takes order of 10 hours to obtain on Fujitsu VP400 (1.14 GFLOPS peak) which is the former main supercomputer at National Aerospace Laboratory (NAL). Table 1 shows the computing times of some CFD simulations done at NAL. Each simulation needs more than 1 hour par 100 thousand grid points. It is too much to use the numerical simulation for the development of actual aircraft.

Table 1 Some CFD simulations

grid (Kpts)	time (hrs)	
2500	30	STOL experimental aircraft "ASKA", transonic flow
1300	15	civil transport, transonic flow
400	10	HOPE (H-II Orbiting Plane), hypersonic with real gas effect
180	30	OREX (Orbital Reentry Experiment), hypersonic with real gas effect
500	5	spaceplane, supersonic flow
500	10	turbine cascade with tip clearance, 1 pitch
200	50	SCRAM jet engine combustor, hydrogen fuel

Secondly it is said that 5 to 15 million grid points are required to simulate complete aircraft configurations with control surfaces. It means 50 to 150 hours to complete the simulation on the VP400 by simple calculation.

Present viscous flow simulations are exclusively based on the Reynolds averaged Navier-Stokes equations together with some turbulence model. Quantitative agreements are not enough in case of flow with large separation influenced by turbulence. Hence research and development of turbulence models with high accuracy is required. One of the promising approaches is the large eddy simulation (LES) that is said to require about 100 million grid points. It means more than 1000 hour computing time on the VP400. In case of reacting flow simulation, highly accurate physical models are required for turbulence phenomena involving mixing and chemical reaction. For the improvement of such physical models demands a large computing power.

This consideration leads to a simple conclusion that we need a high speed computer with large memory to overcome the barriers CFD researchers are confronting. NAL made such analysis and investigation by 1989 and set the target computer should have the actual performance more than 100 times faster than VP400 to promote the CFD technology. Such high performance could be realized only by a parallel computer.

2 Parallel computation model

In considering high performance computers for scientific and engineering computation the establishment of computation models is important. By consulting the computation models we can determine the hardware architecture and software specification.

Physical model widely employed in the present CFD is the Reynolds averaged Navier-Stokes equation together with some turbulence model and almost all CFD researchers at NAL conduct a calculation by Implicit Approximate Factorization (IAF) method using TVD scheme. Outline of the procedure of the IAF method is as follows.

Conservative form of the Navier-Stokes equations is written as

$$Q_t + E_x + F_y + G_z = 0 \qquad (1)$$

where Q is the vector representing the conserved quantities. The number of conserved quantities is usually 5 (density, momenta and energy) and increases by the number of quantities taken into account if we use a turbulence model represented by partial differential equations or if we consider chemical reactions. E, F and G are the flux vectors. Equation (1) is discretized as follows.

$$L_1 \Delta Q^n = RHS^n \qquad (2)$$

$$RHS^n = L_2(Q^n) \qquad (3)$$

$$Q^{n+1} = Q^n + \Delta Q^n \qquad (4)$$

Here L_1 and L_2 are some finite-difference operators. If L_1 is the identity operator, we have an explicit scheme. In case of implicit method with structured grid, L_1 is represented by a large banded sparse matrix. The implicit approximate factorization (IAF) method is employed to reduce the computation cost in solving the first equation. The linear operator is written as the product of three factors, one for each coordinate direction:

$$L_x L_y L_z \Delta Q^n = RHS^n \qquad (5)$$

where L_x, L_y and L_z are operators restricted to x, y and z-direction, respectively. These operators typically give rise to block tridiagonal matrices. To invert these matrices, one usually uses such efficient solution methods as LU-ADI and diagonalization. One can also use a direct method.

In the following, we explain a typical flow of CFD program using the IAF method. We use the terminologies x-direction and i-direction in the same meaning. After setting initial values, the program enters into the time iteration loop. The right hand side RHS is evaluated with the physical quantities at time n. The computation for RHS at a point (i, j, k) requires values at its neighboring points but can be completed concurrently for all i, j and k. Then the inversions of L_x, L_y and L_z are performed. In the inversion of L_x, the process can be executed concurrently in both j-direction and k-direction while the process is sequential in i-direction. This is the same for the inversions of L_y and L_z. We obtain Q^{n+1} at time $n+1$ for interior points by adding ΔQ^n to Q^n. The values Q^{n+1} at boundary points are calculated based on the boundary conditions. We repeat the above time iteration loop until numerical solutions converge if we want to obtain a steady state.

We consider a parallel computation model based on the flow of the CFD program mentioned above. For the following discussion assume that N_p is the number of processing elements (PEs). We also assume that N_i, N_j and N_k are the numbers of grid points in i-direction, j-direction and k-direction, respectively. Note that the CFD program has at least two directions in which the computation can be performed simultaneously. For example, in case of the inversion of L_x we can take j-direction and k-direction. Accordingly we can use one direction for vectorization and the other for parallelization. Keeping this fact in mind, we divide array data evenly along k-direction and assigns each part to each processor as in Fig. 1. We refer to this partition as P_k. With this partition x-derivatives and y-derivatives in RHS can be computed on each processing element without any communication. The vectorization can be made in either i-direction or j-direction. The computation of z-derivatives on PE_k needs the data on PE_{k-1} and PE_{k+1}. This implies that the communication with the neighboring PEs occurs every time the computation requires data on the neighboring PEs. In the computation model we propose, data should be transferred in advance into the array subdivided along j-direction. We refer to this partition as P_j. This partition allows us to compute z-derivatives on each PE independently. It is also useful to use the concept of wing, which will be explained in section 4.

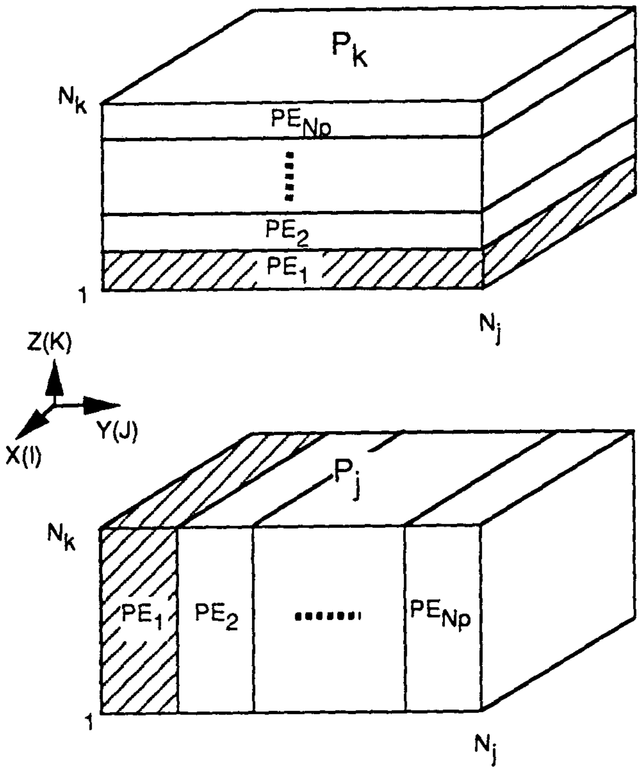

Figure 1: Partitions P_k and P_j

Next step is the inversions of L_x, L_y and L_z. We assume that the order of the inversions is not changed in every time iteration. Before the inversion of L_x, we execute data transfer from partition P_j to partition P_k. As a result the inversion of L_x and L_y can be done on each PE independently and the vectorization can be done in j-direction in case of the inversion of L_x and in i-direction in case of the inversion of L_y. In inverting L_z, we find that the process is sequential in z-direction. Since the direction coincides with the partition direction of the parallelization, effective process cannot be done. This is the most serious problem for the parallelization of the CFD program. To solve the problem several methods have been proposed including wave front methods. Here we transfer the data from partition P_k to partition P_j, which is called transposition. This data transfer pattern puts heavy loads on the interconnection network. But if the network is able to deal efficiently with the transposition, the network could flexibly treat other data transfer patterns. Then we can compute the inversion of L_z on each PE with vectorization in i-direction.

In this way we obtain the correction ΔQ^n on the partition P_j. Then by adding ΔQ^n to Q^n, we obtain Q^{n+1}. At the same time, we compute the boundary conditions independent of j-direction. Finally going back to the partition P_k, we compute the boundary conditions dependent of j-direction.

3 Basic Specification of the NWT

From the computation model and studies on the trend of computer technologies we sum up the following basic specification expected for the NWT:

1. actual performance of more than 100 times faster than the VP400 for NAL CFD programs
2. flexibly applicable to computing methods NAL CFD researchers employ
3. parallel computer with distributed memory
4. each PE displays more performance than the VP400
5. short- and equi-distance interconnection network
6. low power consumption
7. flexible manageability

By analyzing the NAL CFD activity, we concluded that the target performance should be more than 100 times faster than the VP400. As far as we insist on such high performance, it might be impossible to achieve the performance with a supercomputer of shared memory type and we should select a parallel computer with distributed memory. Since our computation model admits one direction left for vectorization in addition to parallelization, we can adopt a high performance vector computer as a processing element. Large granularity of performance of a processing element implies that existing NAL CFD codes can run at a high speed on 1 PE without any modifications. This benefits the continuation of CFD researches.

One of the causes to degrade the actual performance of parallel computers with distributed memory is the contention of communication. It occurs typically in the data exchange between all pairs of processing elements when we change the partitions P_k and P_j in our computation model. In order to cope flexibly with various computing methods a strong interconnection network are required. These requirements lead to the short- and equi-distance interconnection network.

On the other hand, the fact that we can adopt a high performance vector computer as a processing element implies that we can do with a relatively small number of processing elements to achieve the target performance. As a result, this enables us to consider such strong networks as crossbar network in contrast with microprocessor-based massively parallel processors. Then we would obtain an applicability to a wide variety of algorithms and gain a flexible allocation of PEs to multiple jobs to realize a high total utilization ratio of PEs. Furthermore, since the adoption of a crossbar network can produce the system I/O with the equal distance and equal throughput from every PE, the flexibility for the management as a center machine is secured.

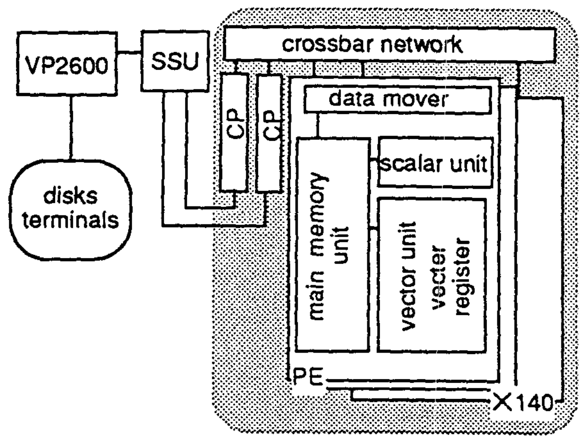

Figure 2: Configuration of the NWT

Eventually the NWT was realized through the joint research and development between NAL and Fujitsu. It is a parallel computer with 140 vector processors connected by a crossbar network. I/O is managed by two control processors connected with the crossbar network. Fujitsu VP2600 is used as a frontend processor. Each processing element is a vector computer with pipelines of multiplicity 8. Add, multiply, load and store pipelines can be operated simultaneously. Hence, each PE has the peak performance of 1.7 GFLOPS. The capacity of vector register is 128kB. Scalar instruction is a long instruction word (LIW) which issues 3 scalar instructions or 2 floating point instructions at once. Data transfer is managed by the mover and can be done asynchronously. Each PE has main memory of 256 MB. The total peak performance is 236 GFLOPS and the total main memory capacity is 35 GB. Fig. 2 shows the configuration of the NWT.

```
!XOCL   PROCESSOR PR(3)
        α
!XOCL   PARALLEL REGION
        β
!XOCL   SPREAD REGION /P(1)
        γ
!XOCL   REGION /P(2)
        δ
!XOCL   REGION /P(3)
        ϵ
!XOCL   END SPREAD
        ζ
!XOCL   SPREAD DO
        DO 1 I=1,30
        η
    1   CONTINUE
!XOCL   END SPREAD
        θ
!XOCL   END PARALLEL
        κ
```

Figure 3: Basic execution method

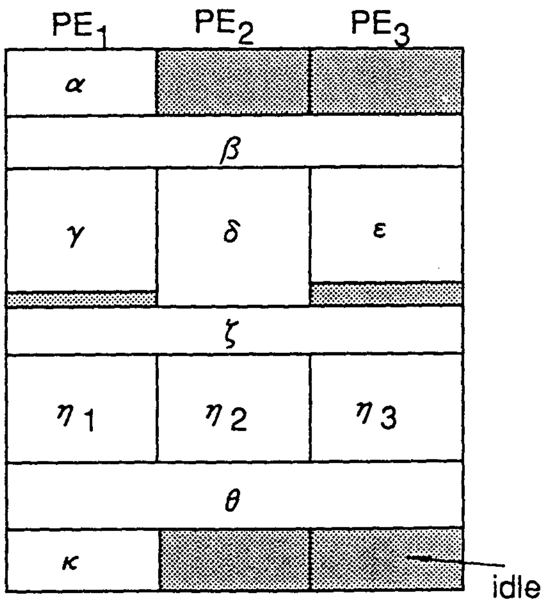

Figure 4: Flow of the execution

4 NWT Fortran

The language to describe parallel processing is the NWT Fortran. It is based on the Fortran 77 enhanced with compiler directives.

The NWT is a distributed memory computer and main memories are physically distributed. However, the logical model assumed for programming hierarchical memory parallel processor system, namely, language processor in conjunction with operating system offers virtually global space to ease programming. Only the data declared as global data are recognized as data on the global space. When global data are referred to, compiler and OS judge where the data are and transfer them through mover into the local space.

Basic parallelization with NWT Fortran is to describe process decomposition, especially decomposition of DO loops, and data decomposition corresponding to the process decomposition to perform parallel processing effectively.

Basic execution method is called the spread/barrier execution method. Fig. 3 shows an outline of the execution method.

The PROCESSOR statement declares the name and the shape of a processor group. In this example, three processor named PR are used. Only the program portion between a PARALLEL REGION statement and an END PARALLEL statement is executed in parallel. The program portions α and κ are executed by one processor. If SPREAD statements are specified, each processor executes only the program portion assigned to that processor. The program portions β, ζ and θ are commonly executed the three processors. The program portions γ, δ and ϵ in the SPREAD REGION structure are separately executed by the assigned processors. The program portion η in the SPREAD DO structure is executed 30 times, 10 times by each of the three processors. At each END SPREAD statement, a barrier synchronization is peformed. Fig. 4 shows the process flow on the NWT corresponding to Fig. 3.

Data transposition is easily treated by SPREAD MOVE statement. The array A in Fig. 5 is subdivided along i-direction while the array B along j-direction. Simple substitution between "!XOCL SPREAD MOVE /(PR)" and "!XOCL END SPREAD (ID)" implies the data transposition.

Next we explain a concept "WING" adopted by the NWT Fortran for the first time which is very efficient in referring to array data with indices $I \pm 1$ characteristic of difference schemes. This allows us to specify a partition so that adjacent partitioned ranges overlap and have some common indices. The overlapping part of the index range is called wing.

```
!XOCL   PROCESSOR PR(10)
        DIMENSION A(100,100), B(100,100)
!XOCL   LOCAL A(/(PR),:)
!XOCL   GLOBAL B(:,/(PR))
        .........
!XOCL   SPREAD MOVE /(PR)
        DO 10 I=1,100
        DO 10 J=1,100
        B(I,J)=A(I,J)
   10   CONTINUE
!XOCL   END SPREAD (ID)
!XOCL   MOVEWAIT (ID)
        .........
```

Figure 5: Data transposition

```
        PARAMETER (N=80)
!XOCL   PROCESSOR PR(4)
        DIMENSION X(80), Y(80)
!XOCL   LOCAL X(/(PR,WING=(1,1)))
!XOCL   LOCAL Y(/(PR)
        .........
!XOCL   PARALLEL REGION
        .........
!XOCL   WINGFIX (X) (ID)
!XOCL   MOVEWAIT (ID)
!XOCL   SPREAD DO /(PR,INDEX=1:N)
        DO 10 I=2,N-1
        Y(I)=X(I-1)+X(I)+X(I+1)
   10   CONTINUE
!XOCL   END SPREAD
        .........
!XOCL   END PARALLEL
```

Figure 6: WING and WINGFIX

The statement "!XOCL LOCAL X(/(PR, WING =(1,1)))" in Fig. 6 implies that the data from X(1) to X(21) are allocated to PE_1, as in Fig. 7, The data from X(20) to X(41) are allocated to PE_2. Similarly, the data from X(40) to X(61) are allocated to PE3 and the data from X(60) to X(80) are allocated to PE_4. "!XOCL SPREAD DO /(PR,INDEX=1:N)" together with "DO 10 I=2,N-1" implies that I=2,20 are executed on PE_1, I=21,40 are executed on PE_2, and so on. Hence, I=61,79 are executed on PE4. The procedure in "DO 10" requires the data from X(20) to X(41) on PE_2. If we use a simple equal data decomposition, since the datum X(20) are on PE_1 and X(41) is on PE_3, we need data transfer. But the overlapped decomposition allows PE_2 to execute the DO loop independently.

The WINGFIX statement specifies the replacement of the overlap range of partitioned local array with overlap with the value of the range corresponding on another processor. The MOVEWAIT statement is required to secure the compatibility of the data on wing.

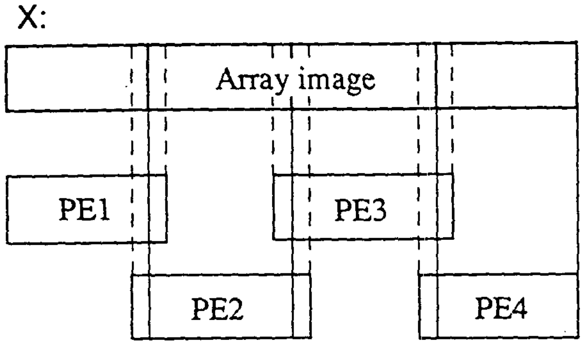

Figure 7: Array image with wing

5 Performance evaluation of the NWT

In this section we discuss the performance measured on the NWT.

For the LINPACK highly parallel benchmark test the NWT attained 124.5 GFLOPS with 140 PEs by using only Fortran language.

5.1 NS3D

NS3D is a CFD program which solves numerically the Navier-Stokes equations without turbulence model by using a TVD scheme and the IAF method. The block tridiagonal matrices mentioned in section 2 are inverted directly by the Gauss elimination. As a programming technique, it employs the minimum number of 3-dimensional arrays and computes such values as pressure and sound speed every time they appear. Consequently the number of floating point operations per grid point is as large as about ten thousands. 5 arrays are transferred. We have to consider the trade-off between main memory usages and computing times when we attempt to compute a large scale problem.

Table 2 shows the effect of the number of PEs in case of NS3D run with 64x128x128 grid points. Fig. 8 shows the computing time and communication delay. The number of processors employed changes from 2 to 128. The computation time becomes half as the num-

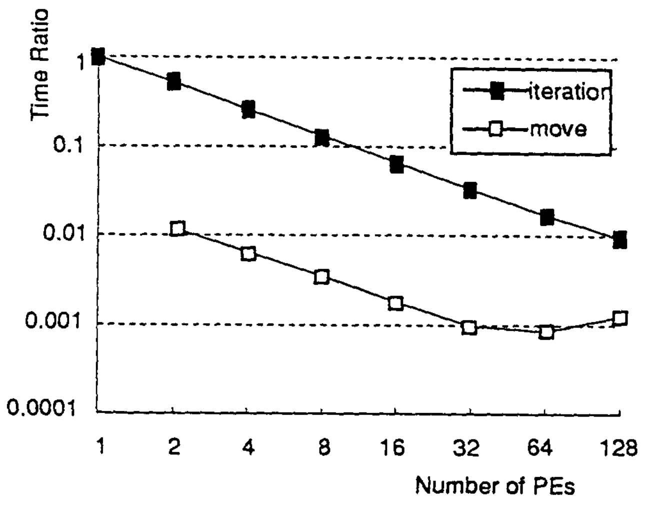

Figure 8: Computing time and communication delay with NS3D

ber of PEs doubles. The communication delay also becomes half as the number of PEs doubles up to 32. However the communication delay becomes longer as the number of PEs increases in case of 64 and 128 PEs since the packet size becomes small and the packet generation time becomes long. The elapsed time decreases almost linearly.

Table 2a Effect of the number of PEs with NS3D

# of PEs	2	4	8	16
Speed-up	1.89	3.72	7.49	14.96
Effect	0.95	0.93	0.94	0.93

Table 2b Effect of the number of PEs with NS3D (continued)

# of PEs	32	64	128
Speed-up	29.77	57.65	101.7
Effect	0.93	0.90	0.79

To see the maximum performance with NS3D we treated the $512 \times 420 \times 280$ case on 140 PEs. Then we have the actual performance of 116 GFLOPS for the time iteration loop.

5.2 TVDSD

TVDSD is a CFD program discretized by using a TVD scheme and the IAF method. It employs an algebraic turbulence model. The inversions of the block tridiagonal matrices are solved by diagonalization. It uses about 90 work arrays in contrast with the NS3D. Therefore the number of floating point operations per grid point is as small as 2400. In this case, 16 arrays are transferred. Since the amount of data to be transferred is larger, the weight of the computation part is smaller compared to NS3D. It takes 18 minutes to iterate 2000 times with $1001 \times 100 \times 50$ grid points on 50 PEs. It takes 37 minutes to iterate 2000 times with $1001 \times 100 \times 100$ grid points on 100 PEs. In this case the weight of the data transfer is 15.5%.

5.3 FFT3DX

Finally we discuss 3-D FFT program FFT3DX as an example of the small number of floating operations in DO loops and large weight of data transfer. In this program one direction is used for the FFT while the computation can be executed concurrently in the other two directions. Hence we can use one direction for vectorization and the other for parallelization. This program is made so as to be able to perform the turbulence simulation in 512^3 case. It calls three times a subprogram which execute the FFT with respect to the third variable and the vector process on first variable. Table 3 shows the performance varying the scale of the problem. 512^3 FFT is executed in about 270 milliseconds and attains about 63 GFLOPS.

Table 3 Performance with FFT3DX

size of problem	number of PEs	time (ms)	performance (GFLOPS)
128^3	32	23.1	8.7
256^3	128	60.8	30.6
512^3	128	269.3	62.8

6 CFD applications

6.1 Direct numerical simulation of isotropic turbulence (cf. [5])

The DNS with 512^3 Fourier components for the Reynolds number of 2000 has been conducted by the NWT with 128 PEs. The dynamical equations of Fourier components are calculated by the de-aliased Fourier-spectral methods. It requires about 24 hours of its CPU time. The key technique of the Fourier-spectral method is to calculate the convolution terms by use of the fast Fourier transform (FFT). The main part of the computation is spent by the calculation of the FFT.

This realistic simulation time enables us physical studies of turbulence by using 512^3 DNS.

6.2 An unsteady Navier-Stokes Solver for cascade flows (cf. [6])

A three dimensional Navier-Stokes solver has been converted for the NWT. This code is based on the Chakravarthy-Osher's TVD scheme and differentiable flux limiter. LU-ADI approximate factorization method is applied for the time integration and Baldwin-Lomax algebraic turbulence model is also used. This solver has been applied to unsteady rotor-stator interaction problems of centrifugal compressors. The number of grid points for each blade pitch is 100 for streamwise, 48 for spanwise, and 49 between blades. The accomplished speed-up by parallel code is more than 5 times with 8 PEs and about 9 times with 16 PEs. Total time to get a periodic solution is about 10 hours with 8 PEs.

This code can be employed for single stage analysis of cascades with stator and rotor to capture circumferentially nonperiodic phenomena such as inlet distortion and rotating stall.

6.3 Aerodynamic design of HYFLEX by hypersonic parallel CFD code (cf. [7])

From the viewpoint of design, a great number of parametric studies are needed. In the present parallel parametric study, the analysis of hypersonic aerodynamic characteristics of HYFLEX (Hypersonic Flight Experiment) are performed. Computation were made by the flux-split TVD Navier-Stokes code. Computational region are equally divided into 4-8 zones circumferentially and parallel computations using 4-8 PEs are performed. Computational speed using 4 PEs are 3.5 times faster than that of 1 PE calculation. 10 cases of the different flow conditions are computed at one computational cycle with totally 40-80 PEs. About 35-70 times improvement of computational speed has been attained in the present numerical analysis. These multiple parallel processing enables us to perform several hundreds of numerical computations and produce sufficient numerical data necessary for the hypersonic aerodynamic design.

7 Conclusion

NWT is a parallel vector computer developed to promote CFD activities. It is devised based on the structures of CFD codes. Flexible applicability to a wide variety of algorithms is taken into account.

We have attempted the performance evaluation using several programs. As a result we obtained more than 90% of the effect of the number of PEs up to 64 with the program NS3D which has provided a guiding computation model through the development of the NWT. We attains about 190 times faster performance with 140 PEs than VP400. For the program TVDSD which is used for the viscous simulation of the flow around an aircraft configuration it takes about 27 to 45 minutes to complete the 5 million grid point simulation on 50 PEs. Although the VP400 cannot treat this size of problem due to the lack of main memory capacity, it might take about 50 hours if the VP400 had enough memory. For the FFT program where the weight of the data transfer is large, we obtain the 512^3 FFT results in about 63 GFLOPS on 128 PEs.

Through the operation of the NWT, it is proved to be flexible to use for CFD application ranging from basic researches such as DNS and the simulation of unsteady flows to application-oriented works such as parametric studies of aerodynamic design.

References

[1] Fukuda, M., Iwamiya, T. and Miyoshi, M., "UH-SNWT Initiative at National Aerospace Laboratory", *Notes on Numerical Fluid Mechanics*, Vol.37,1993,pp.157-198.

[2] Nakamura, T., Yoshida, M., Fukuda, M., Murase, T. and Matsuzaki, T., "Performance Evaluation of NWT with CFD programs" (in Japanese), to appear.

[3] Iwamiya, T., Fukuda, M., Nakamura, T. and Yoshida, M., "On the Numerical Wind Tunnel (NWT) Program", *To appear in Proseedings of Parallel CFD '93.*

[4] Iwamiya, T. and Fukuda, M., "National Aerospace Laboratory Numerical Wind Tunnel", *Workshop on Benchmarking and Performance Evaluation in High Performance Computing.*

[5] Yamamoto, K., "Direct Numerical Simulation of Isotropic Turbulence Using the Numerical Wind Tunnel", *Preprints for Parallel CFD '94.*

[6] Yamane, T. "The transplantation of an Unsteady Navier-Stokes Solver for Cascade Flows onto the NWT System", *Preprints for Parallel CFD '94.*

[7] Yamamoto, Y., "Aerodynamic Design of HYFLEX by Hypersonic Parallel CFD Code Using NAL NWT System", *NAL Research Progress 1993.*