ParNSS : An Efficient Parallel Navier-Stokes Solver for Complex Geometries

J.Häuser,
Center of Logistics and Expert Systems, Salzgitter, Germany

M. Spel, J. Muylaert,
ESTEC, Noordwijk, The Netherlands

R.D. Williams
California Institute of Technology, Pasadena, California

Extended Abstract [1]

The *ParNSS* Code

ParNSS is an acronym for Parallel Navier-Stokes Solver: it solves the Reynolds averaged Navier-Stokes equations on a multiblock grid topology. Our objectives in producing this code are for a solver that combines sophisticated fluid-flow simulation with sophisticated software engineering.

ParNSS can solve complex physics – meaning viscous, hypersonic and eventually reactive flow – with enough robustness that the the user need not be an expert to run the code, yet enough flexibility that the expert can manipulate the solution strategy in many ways. In addition to the physics of the flow, *ParNSS* can operate in complex geometrical domains: such as a complete aircraft configuration, combinations of internal and external flow, and with complex interactions of shock and slip surfaces.

Program Organization

The code has been written entirely in ANSI C, making full use of the advanced concepts provided by the language, such as dynamic data structures and pointers. Thus a much more compact, well-structured, and maintainable code has been obtained.

The individual tasks are completely encapsulated, following the programming philosophy of Object-Oriented-Programming. The code is organized as a collection of block objects and face objects; the blocks contain flow data in long vectors for maximum efficiency and the faces act as messengers between the blocks (which may be on different processors)[3].

As tests on an IBM Risc 6000/560 have revealed, by comparing *ParNSS* with another N-S solver written in Fortran, the code is is much more efficient both with respect to memory usage and computing time.

The code accepts grids in multi-block Plot3D grid format. The flow output is either in Tecplot or Plot3D format, enabling the usage of the major flow visualization packages, such as AVS and Visual3. The code provides the user with architecture-independent software which runs on a sequential machine, a workstation cluster, or a massively parallel system.

Parallelization

[1] Submitted to AIAA 25th Fluid Dynamics Conference, Colorado Springs, June 1994

	Flops per grid point	Floats per boundary point
Laplace	5	1
Grid∗	75	2
ParNSS	≈ 5000	10

Table 1: Approximate computation and communication requirements of 3 domain-decomposed PDE solvers

The code runs efficiently on clusters of workstations and on massively parallel machines; the results below report results from clusters of SGI and IBM workstations and also the 512-processor Intel Paragon machine.

Parallelization is achieved by domain decomposition. We take the position that the grid (both topology and block size) should be completely determined by the physics and geometry of the problem rather than by the machine used for the simulation. The grid-generation process delivers a certain number of blocks. There may be two problems with the given grid:

- Since each processor of the parallel machine takes one or more blocks, there may not be enough blocks to run the problem on a massively parallel machine. We have tools to automatically split the blocks to allow the utilization of more processors.
- The blocks may have very different sizes, so that the blocks must be distributed to the processors to produce a reasonable load balance. We have tools to solve this bin-packing problem by an algorithm discussed below.

ParNSS is implemented with an extremely simple message-passing model, consisting of only *send* and *receive*. The simplicity of this model implies easy portability, and *ParNSS* is currently ported to PVM and Intel NX message-passing.

The reason for the efficiency of *ParNSS* may be illustrated by comparing three domain-decomposed PDE solvers, being an elementary Laplace solver, an elliptic relaxation scheme for the grid generator *Grid∗* [1] and the *ParNSS* code. In each case, the computational part is proportional to the number of gridpoints, and the communication overhead is proportional to the number of gridpoints associated with the boundary of the domain. Thus the inefficiency of the code is proportional to the surface area to volume ratio for a processor; and also proportional to the ratio of the amount of communication per boundary gridpoint, to the amount of work (flops) per internal gridpoint.

For an elementary Laplace solver on a square grid, each gridpoint takes the average of its four neighbors, requiring 5 flops, and communicates 1 floating-point number for each gridpoint on the boundary. The *Grid∗* grid generator is a more sophisticated elliptic solver, where about 75 flops occur per internal grid point, while grid coordinates are exchanged across boundaries. The *ParNSS* code, in contrast does a great deal of calculation per grid point, while the amount of communication is still not large (see Table 1).

ParNSS exchanges information only across the faces of the blocks, rather than the more conventional approach where exchanges occur across faces, edges, and corners. In two dimensions, this means the *ParNSS* sends 4 messages per block rather than 8, and in three dimensions, it sends 6 rather than 26 messages, thus making a very substantial reduction in message traffic.

Thus we may expect *ParNSS* to be highly efficient in terms of communication, as shown by the timing results below. When the complexity of the physics increases, with turbulence models and chemistry, we expect the efficiency to get even better. This is why *ParNSS* is a viable parallel program even when running on a workstation cluster with slow communication (Ethernet).

Parallel Machines and CFD

For the kinds of applications that we are considering in this paper, we have identified four major issues concerning parallelism, whether on workstation clusters or massively parallel machines.

Load balancing As discussed above, the number of blocks in the grid must be equal to or larger than the number of processors. We wish to distribute the blocks to processors to produce an almost equal number of gridpoints per processor; this is equivalent to minimizing the maximum number of gridpoints per processor. We have used the following simple algorithm to achieve this.

The largest unassigned block is assigned to the processor with the smallest total number of gridpoints already assigned to it, then the next largest block, and so on until all blocks have been assigned.

Given the distribution of blocks to processors, there is a maximum achievable parallel efficiency, since the calculation proceeds at the pace of the slowest processor, i.e. the one with the maximum number of gridpoints to process. This peak efficiency is the ratio of the average to the maximum of the number of gridpoints per processor, which directly proceeds from the standard definition of parallel efficiency.

Convergence *ParNSS* uses a block-implicit solution scheme, so that with a monoblock grid, it is completely implicit, and when the blocks are small, distant points become decoupled. Increasing the number of processors means that the number of blocks must increase, which in turn may affect the convergence properties of the solver. It should be noted [1] that the physical fluid has a finite information propagation speed, so that a fully implicit scheme may be neither necessary nor desirable.

Performance It is important to establish the maximum achievable performance of the code on the current generation of supercomputers. Work is in progress with the Intel Paragon machine and a Cray YMP.

Scalability Massively parallel processing is only useful for large problems. For *ParNSS* , we wish to determine how many processors may be effectively utilized for a given problem size – since we may not always run problems of 10 000 000 gridpoints.

Test Cases

Following the same philosophy as Gaitonde and Shang [4], the test runs that we shall discuss are meaningful to real applications. The Space Shuttle symmetry plane shows the bow shock, boundary layer, and an angle of attack corresponding to a real flight situation. The hyperboloid flare simulates the influence of flap deflection in a generic space vehicle, with a shock-boundary layer interaction, recirculation, and reattachment of the flow.

Shuttle Symmetry Plane

Here we use a 37-block 2D grid which is taken from the symmetry plane of a 3D Navier-Stokes grid around the Space Shuttle. Although this geometry could be modeled with only 4 blocks, we would like a nearly orthogonal grid on the windward side of the craft, where the important physics occurs. Because of these requirements of grid-line topology, we have used the more complicated 37-block grid. The blocks of this grid have very different sizes.

The computation simulates Mach 10 flow, at angle of attack 18 degrees, and viscosity corresponding to an altitude of 50 km.

Hyperboloid Flare

This problem simulates hypersonic flow at Mach 8.7 past a hyperboloid flare, which is a generic configuration

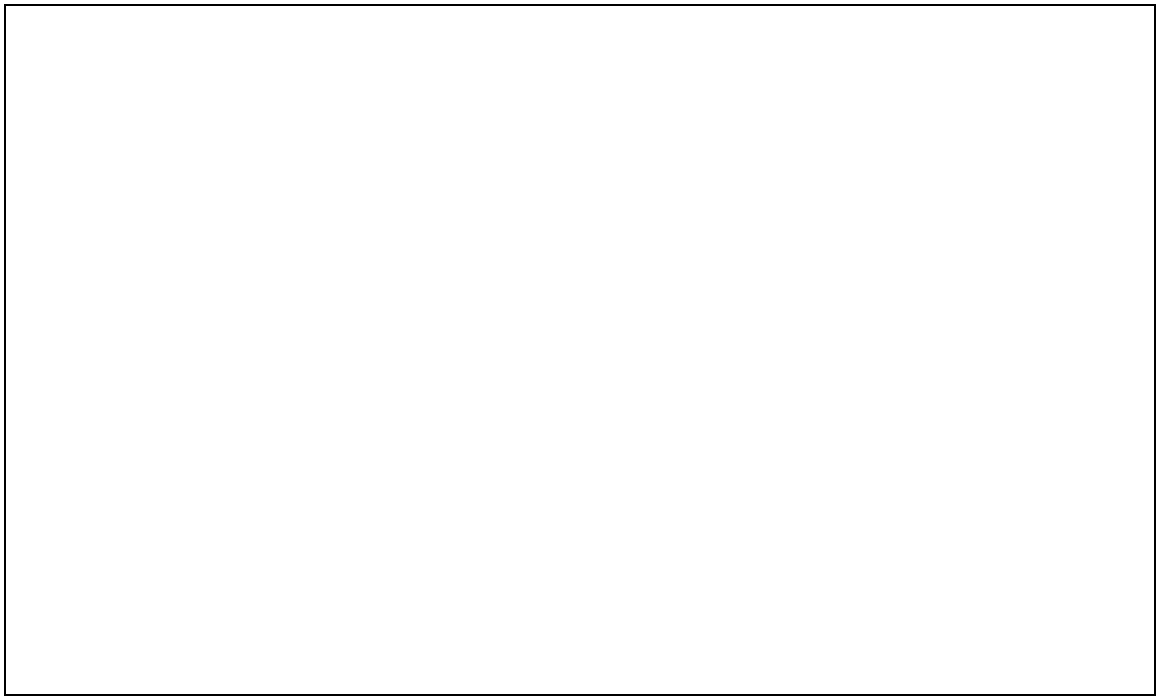

Figure 1: 37 block grid for the Shuttle Symmetry Plane.

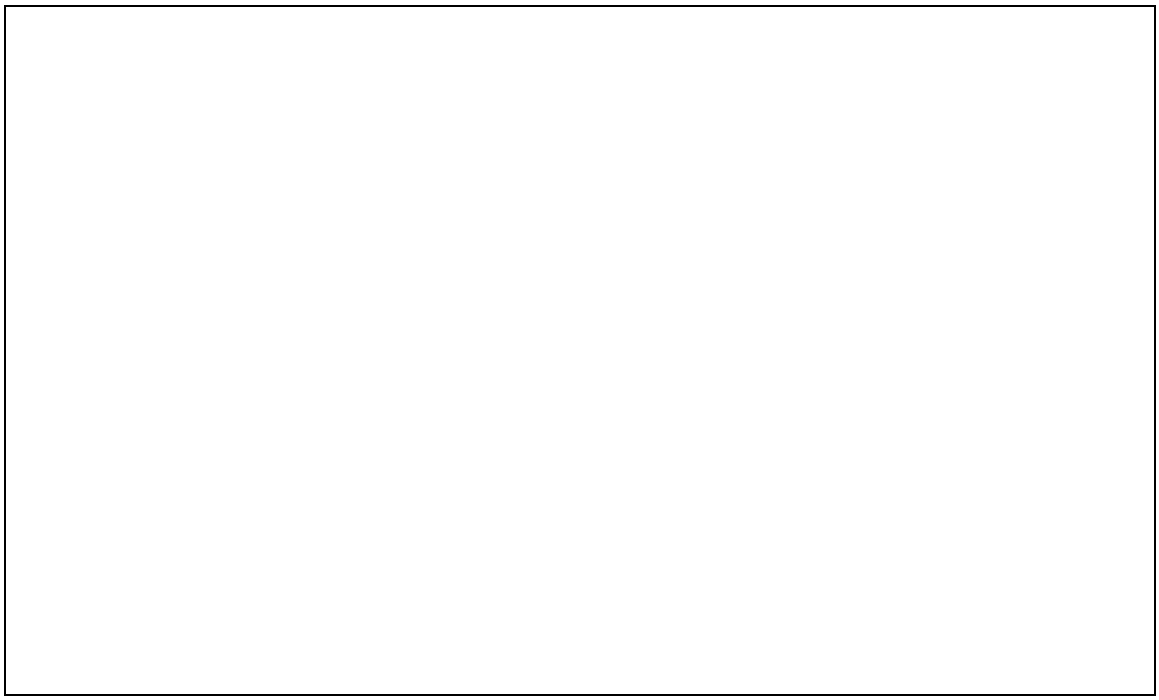

Figure 2: 32 block grid for the hyperboloid flare.

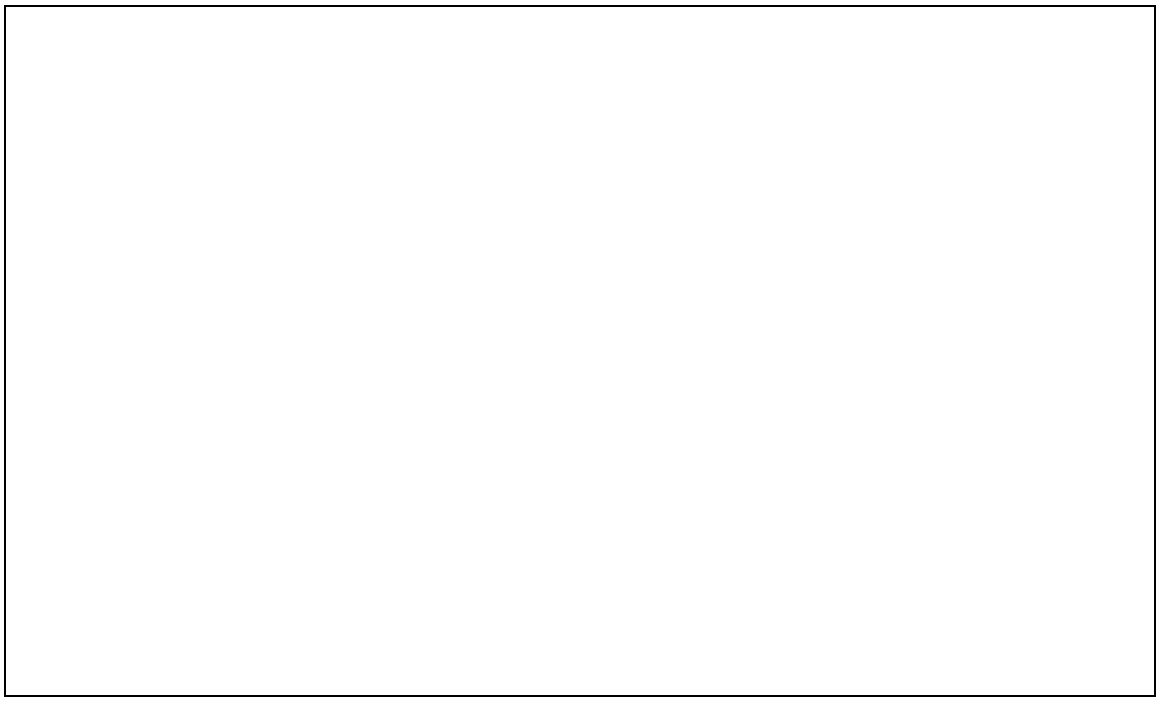

Figure 3: Mach distribution for Shuttle Symmetry Plane after 500 iterations

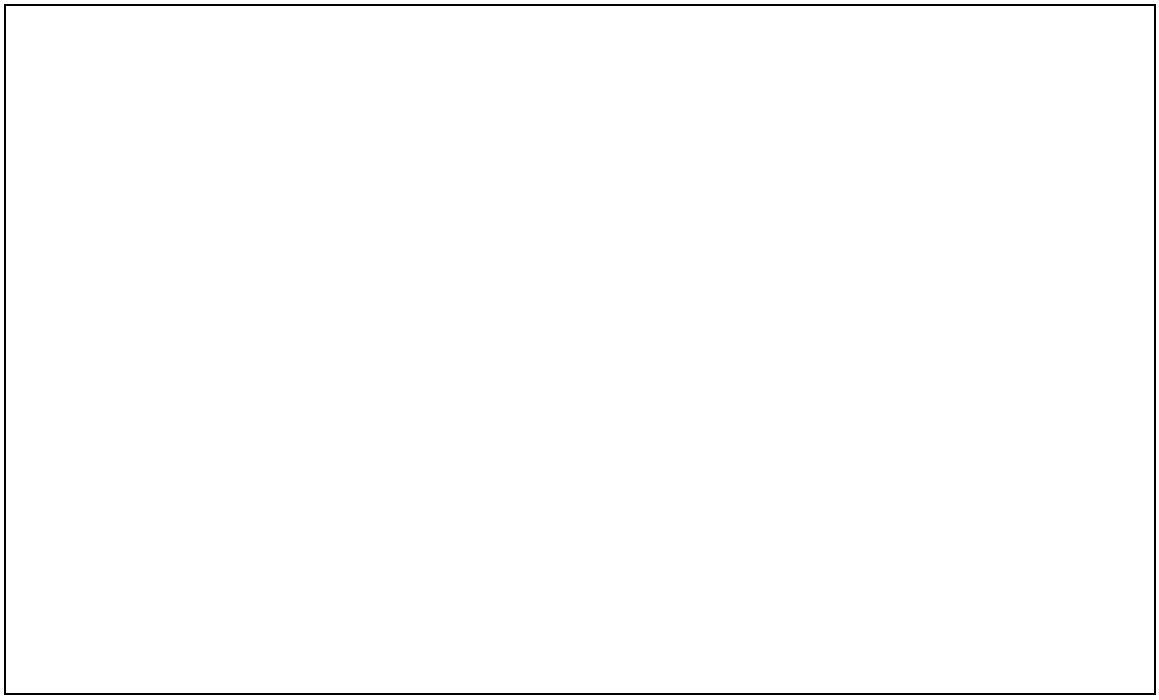

Figure 4: Mach distribution for the hyperboloid flare after 500 iterations

CPU Type	Network	Nodes	Time (sec)	Efficiency
SGI 380	SGI Bus	1	—	—
	Peak bandwidth	2	2691	1
	120 MByte/sec	4	1439	0.935
		8	751	0.90
Indigo 3000	Ethernet	1	4565	1
		2	2321	0.935
		4	1181	0.966
		6	792	0.96
Indigo 4000	—	1	2188	1
SGI Challenge L	SGI Bus	1	423	1
(Indigo 4400)	Peak bandwidth	2	217	1.02
100 MHz	1.2 GByte/sec	4	111	0.999
		8	62	1.01
IBM Risc 6000/580	Ethernet	1	222	1
		2	117	1
		4	62	0.995

Table 2: Time and efficiency for hyperboloid flare problem, 12231 gridpoints in 32 blocks, 100 steps.

Nodes	Blocks: 8, Size: $20 * 20$		Blocks: 32, Size: $20 * 41$	
	CPU Time	Efficiency	CPU Time	Efficiency
1	198	1	1665	1
2	102	0.985	839	0.999
4	51	0.988	421	0.999
8	27	0.985	214	0.994

Table 3: Efficiency measurements for two different block sizes on the IBM 6000/580 cluster for the hyperboloid flare.

for the underbody (windward side) of a spaceplane, e.g. Space Shuttle or Hermes. We assumed viscous laminar flow, which is realistic for the altitude simulated. The flow was axisymmetric and all inflow and outflow boundary conditions were considered to be supersonic.

In Figure 2 the grid used for the hyperboloid flare computation is shown. The grid comprises eight blocks, with a total of 12231 gridpoints. In Figure 4 the Mach number distribution is shown.

Efficiency Measurements

The results of Table 2 show that *ParNSS* obtains efficiencies near 1 for small clusters (up to 8) of workstations, even with Ethernet communications. Indeed, we feel confident that the code would still be efficient on clusters of 20 or more workstations. It should be noted that the efficiencies shown as greater than 1 are not due to superlinear speedup, but rather to the statistical errors in timing.

Table 3 shows efficiency measurements for two different block sizes on the IBM 6000/580 cluster for the hyperboloid flare. Notice that even with the blocksize scaled down to 400 internal points, efficiency is still close to 1.

Results for runs on the Intel Touchstone Delta machine at Caltech are shown in Table 4. As well as the measured efficiency, taken from execution times, the maximum theoretical efficiency is shown. This is the ratio of average to maximum gridpoints per processor, as discussed above. It can be seen that a significant part of the inefficiency can be accounted for by this load imbalance, the rest presumably coming from

CPU Type	Network	Nodes	Time (sec)	Measured Efficiency	Theoretical Efficiency
i860 RX	Delta mesh	1	959	1	1
		2	486	0.99	1.00
		4	247	0.97	0.99
		8	126	0.95	0.97
		16	67	0.89	0.94

Table 4: Time and efficiency for 37-block, 6957 gridpoints, Shuttle problem, with Intel Delta. Computing time is for 100 timesteps.

CPU Type	Network	Nodes	Time (sec)	Measured Efficiency	Theoretical Efficiency
i860 RX	Delta mesh	64	2516	1	1
		256	638	0.98	1

Table 5: Time and efficiency for 512-block, 419 328 gridpoints, hyperboloid flare, with Intel Delta. Computing time is for 100 timesteps.

communication time between the processors.

Table 5 shows some results from running a large grid on 64 and 256 processors of the Delta; this grid is 419 328 total gridpoints, split into 512 equally sized blocks. Since the blocks are all the same size, and the number of processors divides the number of blocks, the theoretical upper limit on the load balance is exactly 1. The ratio of measured computing times for the two runs show that the code still performs at high efficiency even for this much larger problem.

Work in Progress

Although systematic testing has been performed, the code is not fully mature yet. First, more samples have to be run, investigating the influence of grid topology on the parallel efficiency. Also the convergence history of implicit solution schemes for decomposing a domain into a very large number of blocks has not been fully investigated.

To summarize, it has been demonstrated so far that *ParNSS* can be used on workstation clusters and MIMD computers to obtain a computation speed exceeding that of a Cray Y-MP but the limits of neither the code nor the parallel machines have not been determined.

References

[1] J. Häuser, H. D. Simon, and H.-G. Paap, *Features of Architecture Independent Parallel CFD Software*, in *Parallel Computational Fluid Dynamics '92*, ed R. B. Pelz et al., Elsevier North-Holland, Amsterdam 1993.

[2] J. Häuser and R. D. Williams, *Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines*, Int. J. Num. Meth. Fluids, **15** (1992) 51-58.

[3] J. Häuser, H.-G. Paap, H. Wong and M. Spel, *Grid∗ : A General Multiblock Surface and Volume Grid Generation Toolbox*, pp 817-838 in *Numerical Grid Generation in CFD*, eds A.-S. Arcilla et al., Elsevier North-Holland, 1991.

[4] D. Gaitonde and J. S. Shang, *Accuracy of Flux-Split Algorithms in High-Speed Viscous Flows*, AIAA Journal, **31** (1993) 1215-1221.

[5] G. C. Fox, P. Messina and R. D. Williams, *Parallel Computing Works*, Morgan Kaufman, San Mateo, California, 1993.