Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines

Jochem Häuser
European Space Agency

and Roy Williams
California Institute of Technology

Abstract

The purpose of this paper is to predict the efficiency of the Navier-Stokes code NSS*, which will run on an MIMD architecture parallel machine. Computations are performed using a three-dimensional overlapping structured multiblock grid. Each processor works with some of these blocks, and exchanges data across the boundaries of the blocks. The efficiency of such a code depends on the number of grid points per processor, the amount of computation per grid point, and the amount of communication per boundary point. In this paper, we estimate these quantities for NSS*, and present measurements of communication times for two parallel machines, the Intel Touchstone Delta machine, and an Intel iPSC/860 machine, consisting of 520 and 64 Intel i860 processors respectively. The peak performance of the Delta machine is 32 Gflops. Secondly it is shown how, starting from a 7-block grid of about 5 000 000 points for the Hermes space plane, a mesh of 512 equally sized blocks is constructed, retaining the original topology. This example demonstrates that multiblock grids provide sufficient control over both the number and size of blocks. Therefore, it will be possible to simulate realistic configurations on massively parallel systems with a specified number of processors, while achieving good quality load balancing.

1. Computational Requirements in Aerospace Design

The design and analysis of new generations of aircraft and hypersonic vehicles requires accurate values for critical quantities. For example, a drag reduction of 0.5% for a new aircraft is considered substantial progress; surface temperatures for the European space plane Hermes must be known within 50 K during the reentry phase when maximum heat flux occurs.

The National Aerospace Plane (NASP) in the United States, and the Sänger two-stage-to-orbit vehicle in Germany form a new generation of airbreathing vehicles which demand integration of the airframe and propulsion systems. Wind-tunnel testing is very difficult in the extreme flight conditions of these vehicles [1], so that their design and analysis will be based to a large extent on numerical simulation including viscosity, nonequilibrium chemistry and combustion. As the air starts to react, new species are created, which are described by additional conservation equations. A multitemperature model may be needed to compute the vibrational temperatures of the diatomic species. At temperatures above 10000K, ionization occurs resulting in additional species. Combustion models for hydrogen/air may have 16 species.

Hence the problem is to solve a large system of coupled nonlinear PDE's in three dimensions with a complex geometry. In general the time scales of the flow and the chemistry are different, leading to a stiff system. To adequately resolve the viscous boundary layer, the grid must be clustered near the surface of the vehicle, leading to highly stretched meshes and thus reducing the convergence rate.

For airbreathing vehicles, we must also model the transition from laminar to turbulent flow. Turbulence modelling increases the computational cost by either calculating the turbulent viscosity locally, or by introducing a turbulent transport model such as the k-ε model (k is turbulent energy and ε the dissipation rate). The number of physical variables is also increased, but also more simulations are required because of the uncertainty of turbulence modelling.

From the foregoing it is clear that such an accuracy can only be achieved with very fine meshes. Present calculations use one to three million grid points, while a satisfactory solution would really need 10 million. We give as an example the computing time for the NSS* code for the Space Shuttle using 200 000 points. Running on one processor of a Cray YMP, each iteration takes eight seconds and convergence of the pressure is achieved in 2000 iterations, making a total running time of about five hours. Convergence of the heat flux solution would require a further factor of five.

The total number of floating point operations for a convergent heat flux solution on a mesh of 10 million grid points is about $5x10^{14}$. If a one Teraflop machine were available with an assumed sustained performance of 250 Gigaflops (optimistic) it would take approximately 2 000 seconds to solve the heat flux problem. An unsteady Navier-Stokes simulation on the same grid would take approximately 8 000 seconds because of the reduction in CFL number. However, as mentioned above inclusion of a turbulence model and nonequilibrium effects would demand a much higher computational effort. If aeroelasticity effects have to be accounted for, i.e. a structural code has to be coupled to the unsteady Navier-Stokes solver, the computational time will increase further. These calculations are valid for a single point of the flight trajectory. To simulate a complete flight envelope, 10 to 20 points may be needed. Recently, the coupling of electrodynamic effects (Maxwell's equations) with the Navier-Stokes equations has become a topic of research. In addition, if CFD is going to play a role in the aerodynamic design process, some sort of shape optimization for a vehicle has to be possible, demanding a new level of performance in supercomputing.

Although from these mostly qualitative arguments it is obvious that, for advanced aerospace applications, the Teraflop machine can only be the first major step to bring up CFD to a level where it can be used as a design tool. It is also safe to claim that the distributed memory MIMD architecture is highly suitable for complex aerodynamic simulations.

2. Interprocessor Communication

MIMD parallel computers offer great increases in speed over conventional supercomputers, and at a much lower cost per megaflop, but with increased software cost. Before making the effort to parallelize a code such as NSS*, it is of great interest to predict the inefficiency, which is defined as

$$inefficiency = 1 - \frac{PT_1}{T_P}$$

where P is the number of processors, T_P is the time taken to run on these P processors, and T_1 is the time taken to run on a single processor. If the reasons for inefficiency are independent, inefficiencies may be added. Many parallel codes, including NSS*, have an inner loop structure of loosely synchronous cycles of computing and communicating, where the communication is exchanges of data between processors whose physical domains are adjacent. If this is the case, there are usually two contributions to inefficiency: load-balance inefficiency and communication inefficiency [2,3].

Load-balance inefficiency occurs because the processors have different amounts of computation to perform, which for NSS* means different numbers of grid points. The algorithm then runs at the speed of the processor with the largest number of grid points, so the load balance inefficiency is the maximum number of grid points divided by the average number of grid points minus one.

Communication inefficiency arises from the time taken for messages to pass between the processors. We first measured the communication rates between just two processors simply exchanging messages. The Intel NX programming environment supports two ways of receiving a message, `crecv` and `irecv`. A call to `crecv` blocks until a message has arrived and been copied into the buffer provided by the application code; thus an exchange of data between two processors consists of a `csend` then a `crecv` call. In contrast, the `irecv` mechanism consists of a processor first "posting a receive", nominating a buffer into which incoming messages are to be placed, then sending to the other processor with `csend`, then calling `msgwait`, which blocks until the expected message has actually been received.

Figure 1 shows the measured communication bandwidth for message exchange between two processors, in megabytes per second, against the size of the exchanged messages. The iPSC/860 machine achieves 2.80 Mbyte/sec, and the Delta 9.80 Mbyte/sec, for exchange of messages between two adjacent processors.

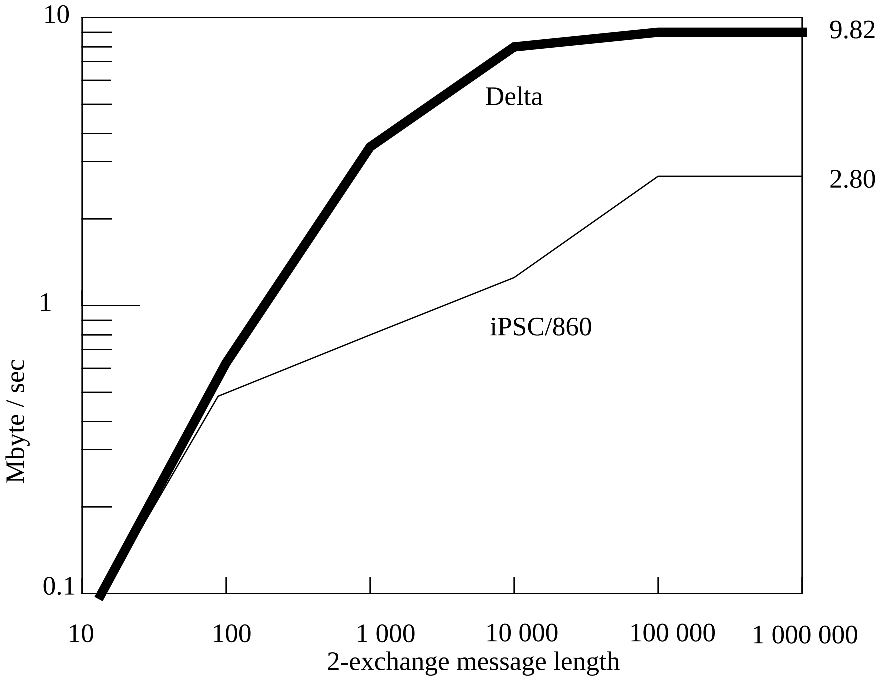

Figure 1: Bandwidth for message exchange between 2 processors with the Delta and iPSC/860 machines,

3. Parallel Multiblock

The NSS* code [4] uses an implicit cell-centered finite-volume scheme, combining an oscillation-free low-order scheme with a third-order MUSCL scheme by a local flux limiter function. Since the finite volume technique and the multiblock approach are widely used, the same or similar efficiency can be expected for a large class of Navier-Stokes solvers.

The solution domain consists of a connected set of logically rectangular blocks [5] in three dimensional space. Each block has at most six neighbor blocks, although some at physical boundaries have fewer neighbors. However, because of the complex geometry of the vehicle, the connectivity of these blocks is irregular. If there are fewer blocks than processors, the blocks may be split judiciously until each processor has one or more of the subblocks. The splitting should be made so that each processor is in charge of approximately equal numbers of grid points. As an example, we shall consider a multiblock grid around the Hermes space plane, made with seven blocks. To use this grid with a massively parallel machine, with perhaps 520 processors, we must split these into many more smaller blocks, and decide which processor is to take which of these smaller blocks.

The code runs in loosely synchronous cycles of computation followed by data exchange across the faces of the blocks. Each processor computes with the data of its block independent of the others, then exchanges data with its neighboring blocks. The exchange consists of sending messages to each of the neighbor blocks, where the size of each message is proportional to the number of grid points on the common face. The processor then waits to receive the corresponding messages from the neighbor processors, and when these are received, the cycle starts again with the computation phase.

The load balance inefficiency results from processors having different numbers of grid points, and the communication inefficiency results from the time required for message-passing between the blocks. We should note at this point that even the sequential code contains a kind of communication inefficiency because of the multiblock structure; the data from the face of the sending block must be taken from memory and ordered, then unpacked into the memory associated with the receiving block. Even if no message is to be sent, such as with a sequential code, this packing and unpacking must still take place.

Figure 2 shows a logical or computational view of the seven blocks from the Hermes grid with their connectivity and the numbers of grid points in each direction. The total grid size is 4 160 000, obtained by multiplying together the numbers of grid points for each block and summing over the seven blocks. The thick lines connecting the blocks indicate that two blocks have the same grid dimensions on the corresponding faces,which is necessary to guarantee a slope-continuous grid. Figure 3 shows the way in which this 7-block grid may be split into 512 subblocks. We have chosen

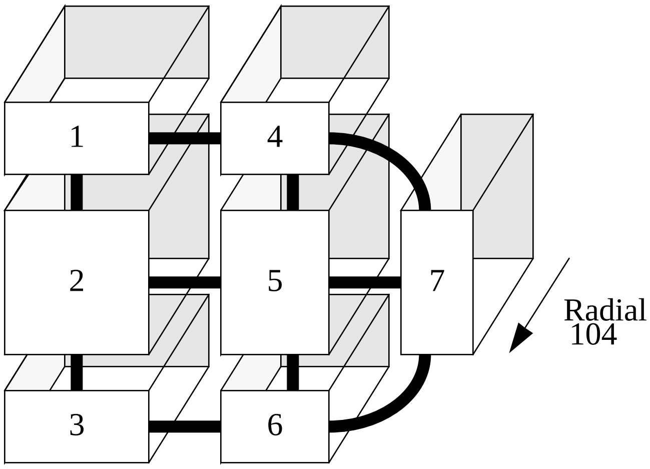

Figure 2: Logical structure of the Hermes 7-block grid.

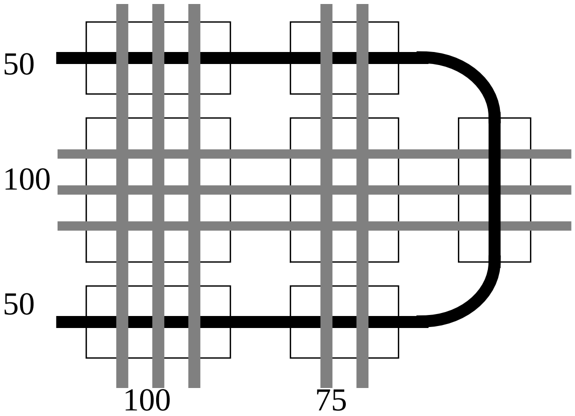

Figure 3: Subdivision of the 7-block grid into 64 equal subblocks. A further splitting in the radial direction produces 512 equal subblocks.

512 processors because this is just less than the 520 processors of the Intel Touchstone Delta machine. Each of the line textures in this figure represents a different way of splitting the grid, being five splitting planes in all, including the splitting on the radial direction. With the splitting as shown, each block has size 25x25x13, with 13 grid points in the radial direction. By splitting only four instead of eight times in the radial direction, we could also make 256 blocks of size 25x25x26.

4. Communication Modelling

In addition to the block of data owned by each processor, there are two surrounding layers of “ghost” points, as shown in Figure 4. These ghost points are updated by the message-passing phase of the calculation, and provide data conti-

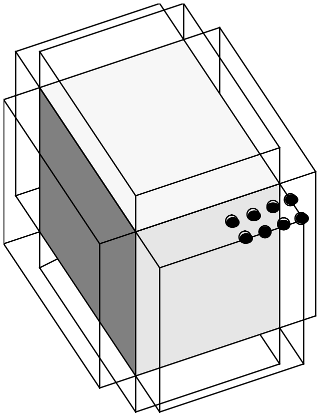

Figure 4: A block of grid points surrounded by two layers of ghost points.

nuity from the neighbor block for the computation. There are two layers of these ghost points because of the third-order nature of the solution scheme used by NSS*.

If the processors have different numbers of grid points, we may expect some load balance inefficiency; indeed we expect that the time for the computational phase of the algorithm cycle is determined by the processor with the largest number of grid points. The splitting of the Hermes grid discussed above has the same number of grid points in each block, so that inefficiency results only from the communication part of the cycle. If the computational time is more accurately modelled, we must include the time taken to prepare and interpret the outgoing and incoming messages: This time is different for the different blocks because interior blocks have six neighboring blocks, whereas those at the physical boundary may only have one or two neighbors.

For the purpose of estimating efficiency, we need to know some information about the algorithm:

N, the total number of gridpoints;

P, the number of processors being used;

F, the number of floating point operations (flops) per grid point per iteration;

B, the number of bytes per grid point to be exchanged between blocks;

and we also need some information about the machine on which the algorithm is to run:

T_{flop}, the time for a processor to do a flop;

T_{cube}, the time per byte for a cubic exchange.

Since the blocks are equal, each block contains *N/P* points. Thus we may define the linear dimension of each block to be $l = (N/P)^{1/3}$. The amount of data to be communicated for each block is then $12Bl^2$ per iteration; a factor 2 comes from the two layers of grid points to be exchanged at each face, and a factor 6 because each block has at most six faces. The number of flops per iteration is Fl^3. Thus the communication inefficiency is then the communication time divided by the computation time, which is

$$\frac{12Bl^2}{Fl^3} \cdot \frac{T_{cube}}{T_{flop}}$$

For the NSS* code, the algorithm parameters might be estimated as follows. As noted in Section 3, we take the number of grid points $N = 4\ 160\ 000$, and number of processors $P = 512$, so that $l = 20$. The number of flops per grid point per iteration is about $F = 5000$ for this low/high order code with a real gas model. The data exchanged across the boundaries are the five primitive variables: density, energy and three components of momentum; so that $B = 5$ words = 40 bytes. We have taken the time for a floating point operation to be $T_{flop} = 0.2$ µs, corresponding to a conservative 5 Mflops for the i860 processor. The computation time per cycle is thus $Fl^3 T_{flop} = 8$ seconds.

We have measured T_{cube} with the following model of the parallel NSS* code. Our model consists of a cubic lattice of equal blocks, so that each processor has exactly the same number of grid points. Each block has six faces, and on the other side of each face there may or may not be another processor.

To simulate the irregularity caused by the geometry of the solution domain, we have made random assignments of the blocks to the processors of the machine. Clearly random assignment is not optimal; there are schemes [6] for assignment of processors to blocks which tend to place adjacent blocks in adjacent processors, and such a scheme would presumably improve the communication performance.

Figure 5 shows the results for bandwidth. For the parameters given above, the number of bytes to be exchanged at each face is $12Bl^2 = 0.192$ Mbyte, so that T_{cube} is 0.16 seconds on 512 processors of the Delta machine.

Thus our measurements predict that each iteration of the NSS* code with 4 million grid points takes about 8 seconds, of which about 0.16 seconds is taken with communication, so that the efficiency is greater than 95%.

5. Conclusions

The communication timings indicate that a complex practical calculation, such as the accurate solution of the Navier-

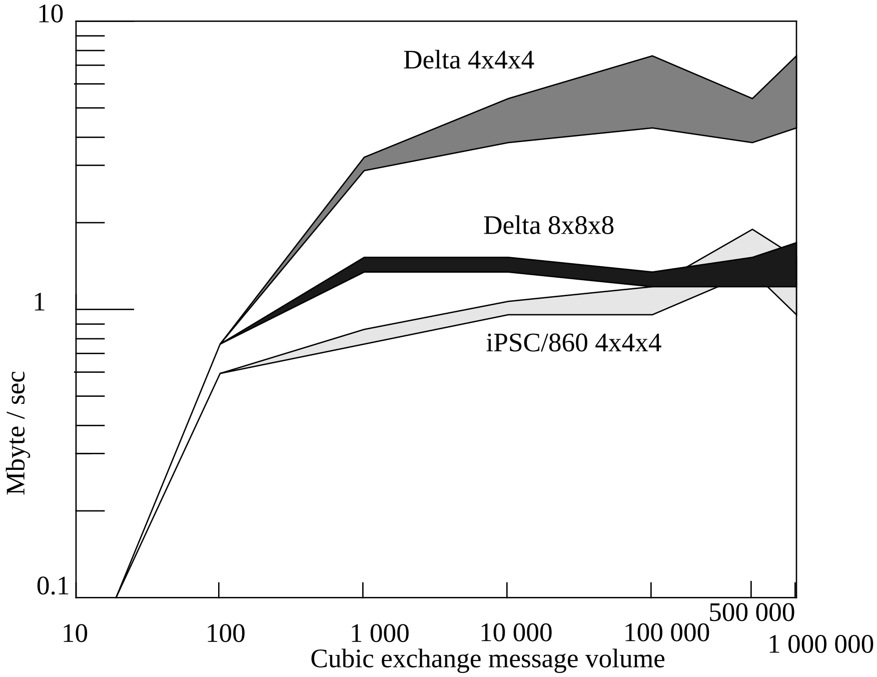

Figure 5: Bandwidth for cubic exchange. The width of the band shows minimum and maximum values for different random assignments of processors to blocks.

Stokes equations with a complex geometry, can be expected to run on a massively parallel machine with high efficiency. There are four main reasons for this high predicted efficiency:

Complex algorithm: Solving the Navier-Stokes equations rather than the simpler Euler equations means that derivatives of the physical fields are required, increasing the computation per grid point. Furthermore, using a real gas model rather than an ideal gas, as well as the use of simultaneous low- and high-order schemes, and the semi-implicit nature of the code all increase the amount of computation per grid point, and hence improve the efficiency.

Hardware speed: We have made preliminary measurements of the communication rate on the Intel Touchstone Delta machine, the result being an increase of a factor of 3.5 over the older generation iPSC/860 machine.

Memory: Each processor of the Delta machine has a memory of 16 megabytes, so that each processor may work on a large block, thus reducing the surface area to volume ratio of the block, thus decreasing the amount of communication relative to calculation.

Grid Partitioning: Along with inefficiency caused by communication overhead, the code would also be inefficient if the processors were unbalanced by having different numbers of grid points in each. We have shown that it is possible to split a 7-block grid into 512 equal subblocks, so that this cause of inefficincy is eliminated. Furthermore, it is possible to make this splitting such that the subblocks have reasonably low surface area to volume ratios, so that the communication inefficiency is low.

Acknowledgements

We would like to thank the Concurrent Supercomputing Consortium for our use of the Delta and iPSC/860 machines, and to Horst D. Simon of the Computer Science Corporation, NASA Ames Research Center, for useful discussions.

References

1. H. Hornung and B. Sturtevant, *Challenges for High-Enthalpy Gasdynamics Research During the 1990's*, Caltech Graduate Aeronautical Lab. Report FM 90-1, January 1990.

2. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors*, Prentice-Hall, New York, 1990.

3. R. D. Williams, *Express: Portable Parallel Programming*, pages 347-352 in Proc. Cray User Group, Austin, Texas, October 1990.

4. J. Häuser and H. D. Simon, *Aerodynamic Simulation on Massively Parallel Systems*, to be published in *Parallel CFD '91*, eds. W. Schmidt, A. Ecer, J. Häuser and J. Periaux, Elsevier North-Holland, 1991.

5. J. Häuser, H. G. Paap, H. Wong and M. Spel, *A General Multiblock Surface and Volume Grid Generation Toolbox*, in N*umerical Grid Generation in Computational Fluid Dynamics and Related Fields*, eds. A. Arcilla, J. Häuser, P. R. Eiseman and J. F. Thompson, Elsevier North-Holland, 1991.

6. R. D. Williams, *Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations*, Concurrency, Practice & Experience, to be published.