Running the ParNSS CFD code on Beowulf and Exemplar

Ralf Winkelmann
CLE, Salzgitter, Germany

and

Roy Williams
Caltech, Pasadena, California

This note reports the results of running a multiblock hypersonic flow code on two parallel supercomputers. The Beowulf-class machine is a "homegrown pile of PC's" made from commercial, off-the-shelf components: the objective is to provide cheap supercomputing for those willing to procure parts and connect them themselves. The other machine is the Exemplar machine from Hewlett-Packard; this NUMA (non-uniform memory architecture) machine is driven by the PA8000 CPU from HP, with a shared-memory programming model, sophisticated software tools, and binary compatibility with programs that run on HP workstations.

These two machines are very different both in architecture and philosophy, but the unifying element that we use in this report is that both can run MPI programs. Here, we are not presuming to assess the quality of the machines in some general sense, but to present the results of running an application (a loosely-synchronous multiblock PDE solver) which uses MPI messaging.

Beowulf

The Beowulf machine at the Caltech CACR has, in September 1997, 58 Intel Pentium Pro compute processors running at 200 MHz. Each processor has 128 MBytes memory, and 3.1 Gbytes disk, and it is packaged as a conventional PC box, with motherboard, power supply, disk, floppy drive, ports and so on. There is a host processor, the only one connected to the Internet, where users can log in, compile, and launch jobs. In addition, each PC is supplied with fast (100 Mbit/sec) ethernet for interprocessor communication, which is routed by four Gigalabs crossbar switches. The architecture of the machine at the Caltech CACR is being constantly optimized, but

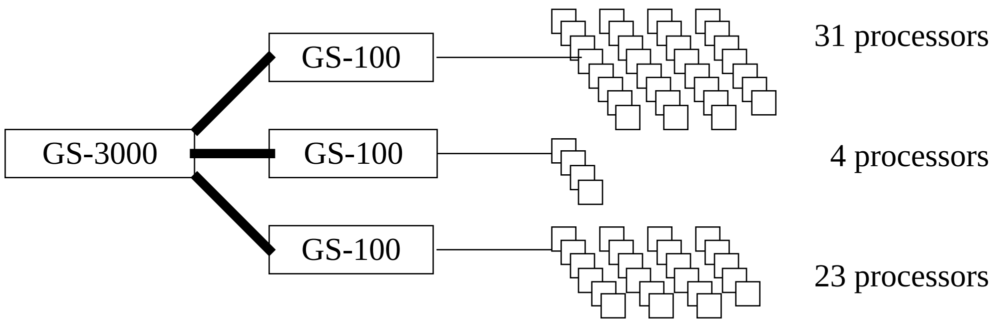


Figure 1: Architecture of the CACR Beowulf (Sept 1997). Each GS-100 switch can carry 1600 Mbit/s peak, and each processor can communicate locally at 100 Mbit/s. Non-local processors communicate through the GS-3000, which is connected at 800 Mbit/s(bidirectional) to each GS-100 switch.

a snapshot from September 1997 is shown in Figure 1.

Each processor of the Beowulf runs the Linux operations system, a freeware version of Unix, and provides TCP/IP sockets via the fast ethernet. Parallel computing can be done via message-passing, either with the free PVM software, or using the free MPICH implementation of MPI from Argonne National Labs.

Of course the widespread interest in the Beowulf-class machine is because of the promise of a supercomputer at a very good price: for the processors described above, each additional is about 2000 US dollars, which is cheaper by a large factor, gigaflop for gigaflop, than conventional ‘big iron’ supercomputers such as the Exemplar, the SGI Origin or the IBM SP2.

Exemplar

The HP Exemplar system at the Caltech CACR has, in September 1997, 256 HP PA8000 processors running at 180 MHz. Each processor has 256 MBytes memory and 4 GByte disk. The system is packaged as sixteen ‘hypernodes’, where each of these consists of 16 processors in a box with 4 GByte shared memory and 64 Gbytes disk.

The processors of a hypernode communicate with each other at high bandwidth and with low latency via a crossbar switch. It is possible to effectively implement a virtual shared memory model, and indeed a popular programming model is to have a single multithreaded process that uses several processes from a hypernode. Processors in different hypernodes communicate via CTI (Coherent Toroidal Interface) in a ring, at a somewhat slower bandwidth, with somewhat higher latency. This model of communication between processors is called NUMA.

ParNSS

ParNSS is a hypersonic, multiblock, Navier-Stokes code that has been used for the ESA Hermes vehicle, the NASA Shuttle, the Huygens lander of the Cassini Saturn mission, and most recently for a model geometry similar to the Lockheed-Martin X33 vehicle. The X33 is widely perceived to be the predecessor to a commercial vehicle code-named ‘Venture Star’.

Because ParNSS is intended for complex geometries, it uses a multiblock grid. Each of these blocks is logically rectangular or cuboidal and there may be hundreds of blocks distributed among the processors of a parallel machine. The reason for the large number of blocks is geometric complexity combined with a requirement for high grid quality, so that the only possibility is a large number of blocks, with, in general, the blocks being very different sizes. Ghost points (a double layer) are exchanged across the face common to two blocks in order to compute fluxes and thus update the primitive variables stored in the cells. There is no communication between blocks that only share an edge or a corner point.

For this report, we have taken a difficult problem: a Mach 9.8 Euler flow over a modified X33 vehicle—modified because the real geometry is secret. This problem is difficult because of the geometric complexity with the resulting large number of blocks; there are a total of 342,361 grid points distributed among 274 blocks. Because of this large number of blocks, therefore a large number of faces, and since messages are exchanged at each face, there are a large number of messages.

In this version of the code, the blocks are distributed among the processors at the beginning of the run, using a heuristic for the knapsack problem that attempts to get the same number of gridpoints in each processor. In this study, with at most 48 processors, it was always possible to get a good uniformity of the number of grid points (within 1%).

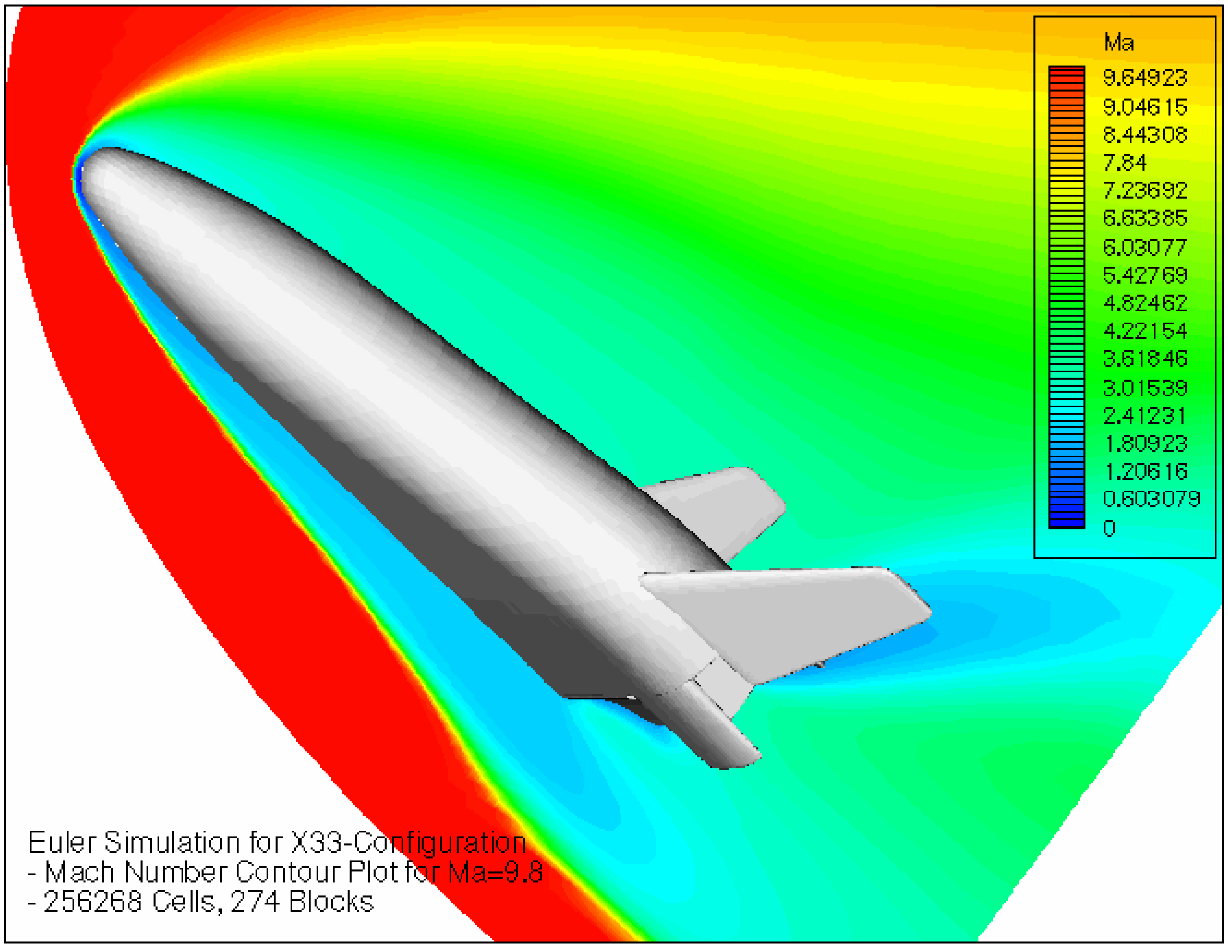


Figure 2: Mach 9.8 Euler flow over a model of the Lockheed-Martin X33 SSTO vehicle. The computational grid has 342,361 points in 274 blocks.

Explicit and Implicit Schemes

ParNSS runs either explicitly or implicitly. In the explicit mode, messages are exchanged frequently: there is a straightforward flux computation from the conservative variables, one per grid point, then ghost points are exchanged at the faces of the blocks. In the implicit mode, a sparse linear system is solved for each block with an iterative linear solver before ghost points are exchanged. Thus, there is much more computation for each exchange of messages, so we would expect much greater efficiency with the implicit mode.

Unfortunately, there is another source of inefficiency for the implicit mode, because the number of iterations of the linear solver varies considerably, meaning that the assumption underlying the load-balance (that the work per block is proportional to the number of grid points) is no longer correct—measurements indicate that on 48 processors, the implicit mode cannot be more that 60% efficient.

To run on a larger number of processors, we would split the larger blocks into two, for, or eight smaller blocks, in part to allow an equal division of the grid points among the processors, and also to reduce the inequality of the number of iterations of the linear solver.

Blocking and Nonblocking Messages

ParNSS is a message-passing code, and it is written with the industry-standard MPI (Message Passing Interface) library. The updating of the ghost points can be done with either of two message-passing strategies, which we call

- BSNR (Blocking Send, Non-blocking Receive). First the processor sets up non-blocking receives for all the messages that it expects to get. Then it loops over the blocks again, sending a message for each face of each block. When all the ghost points have been exchanged in this way, the processor loops over the blocks again, updating each.
- NSBR (Non-blocking Send, Blocking Receive). Here each processor makes a single loop over all the blocks. First the messages containing ghost points are received (with a blocking receive), then the block is updated, then the processor sends the values of ghost points. Thus messages are sent as soon as possible.

For the BSNR method, the communication is separated from the computation; all the receives are posted, then all the messages are sent. On the other hand, for the NSBR method, each message is sent as soon as it is ready, spreading out the communication traffic in time. Thus we would expect that the impact of a slow communication fabric is felt to the greatest extent for the BSNR method.

Even though the NSBR method has the advantage of spreading out the message traffic, it has a potential overhead in that messages are sent as soon as they are available, meaning that they are received before the receiving process is ready to use them. Thus the MPI implementation must be responsible for buffers that can hold the messages during the time before the receiving process can take them.

Development Environments

The Exemplar provides a software environment that includes optimizing compilers, parallel debugger and profiler, backup, accounting, scheduling, and job checkpointing. The Beowulf provides somewhat less, though optimizing compilers and a scheduler are currently available.

The Exemplar software is based on years of development, and presumably the same will be true of the Linux-based Beowulf tools in several years time. While the Exemplar provides (or will provide) binary compatibility with HPUX software, Beowulf software is contributed by groups that run Beowulf machines. We do not intend to discuss the ramifications of these models of software development here.

It took less time to make the ParNSS code run on the Exemplar than it did on the Beowulf. Some of the initial problems with Beowulf involved the Gigalabs switch failing under heavy message traffic, at least until some settings had been changed.

On Beowulf, the code would fail when compiled with strong optimization. Sometimes it would not start, sometimes get NaN results. We settled on a medium level optimization (-O2) that provided both speed and reliability.

The Exemplar is a more integrated system than the Beowulf, meaning that there is a single system image that is on a single hard disk—in addition, the shared memory adds to this comfortable image of a single system. The Beowulf, however is more explicitly distributed: the operating system is stored many times on many disks, so for example adding a new account means that the account is added for each processor in the system. Of course the administrator does not do this by hand because there are tools to execute a command on all processors of the system, but it is hard to imagine a Beowulf system providing the same level of integration and unity that is normal on a multiheaded SMP machine such as the Exemplar.

Results

Our results are shown in Figure 3, one panel for the explicit mode, being the time in seconds for ten iterations of the solver, and the other panel for the implicit mode, also ten iterations. In each panel there are four curves, for the two machines and the two messaging schemes, each for 1, 2, 4, 8, 16, 24, 28 and 48 processors.

Even for this difficult problem of 274 blocks, the efficiency remains good on both machines even with 48 processors. The Exemplar exhibits efficiency of 70% for the explicit mode and 50% for the implicit mode. There are about 1500 messages being sent at each time step, so there is inefficiency is not only from communication time in the machine but also from time that the processors spend waiting for messages, being interrupted, and buffering and copying. In addition, for the implicit mode, there is inefficiency because we are using a static load-balancing strategy for an inherently dynamic computation; we believe that a dynamic load-balancing strategy in ParNSS could improve the efficiency of the implicit mode significantly.

For the Exemplar, there is essentially no difference between the BSNR and NSBR messaging schemes, but for Beowulf there is considerable difference. We conjecture that the NSBR scheme is slower because the MPI implementation must find space for and store the incoming messages, rather than having them delivered directly to the memory location chosen by the programmer. For a larger number of processors, we would expect the behavior to be the other way, with the NSBR scheme faster than BSNR: in this case it is the switch fabric that is the bottleneck, because in BSNR all the messages tend to be sent at the same time, resulting in longer message delivery times.

Our first results for the Beowulf showed a most disturbing loss of parallel efficiency: when the number of processors increased from 24 to 28, the code slowed down by a factor of ten! After repeating this run several times under different conditions, we assured ourselves that it was real. More work, bringing in the Beowulf experts, showed that one of the switches was either misconfigured or faulty, and when corrected, we obtained the better results shown in the Figures. In doing this benchmark, we have unwittingly added a diagnostic to the Beowulf software collection that can debug the crossbar switches.

Conclusions

The Exemplar provides an excellent, scalable platform for MPI applications. It also provides a global shared memory model that, it is hoped, can scale to 256 processors, but we have not tested this.

Unlike the Exemplar, a Beowulf system is not a turnkey solution with a toll-free help desk. The hardware cost of a Beowulf system is a factor of five to twenty lower than a conventional supercomputer, for given achieved megaflops, making it an extremely attractive option to some.

However, buying such a system means joining a collaborative community of other users, being ready and willing to contribute software, to keep an eye on newsgroups and other resources, and to evaluate which software is best for a given system. The Beowulf system at CACR is one of the largest, and the ParNSS code one of the first codes to run on the large system, so perhaps it is not surprising that the hardware and system software is not yet bulletproof. The hope is that as the community grows, reliability and flexibility will also.

References

CACR's Beowulf Machine,
http://www.cacr.caltech.edu/research/beowulf

The Beowulf Project at NASA Goddard,
http://cesdis.gsfc.nasa.gov/linux-web/beowulf/beowulf.html

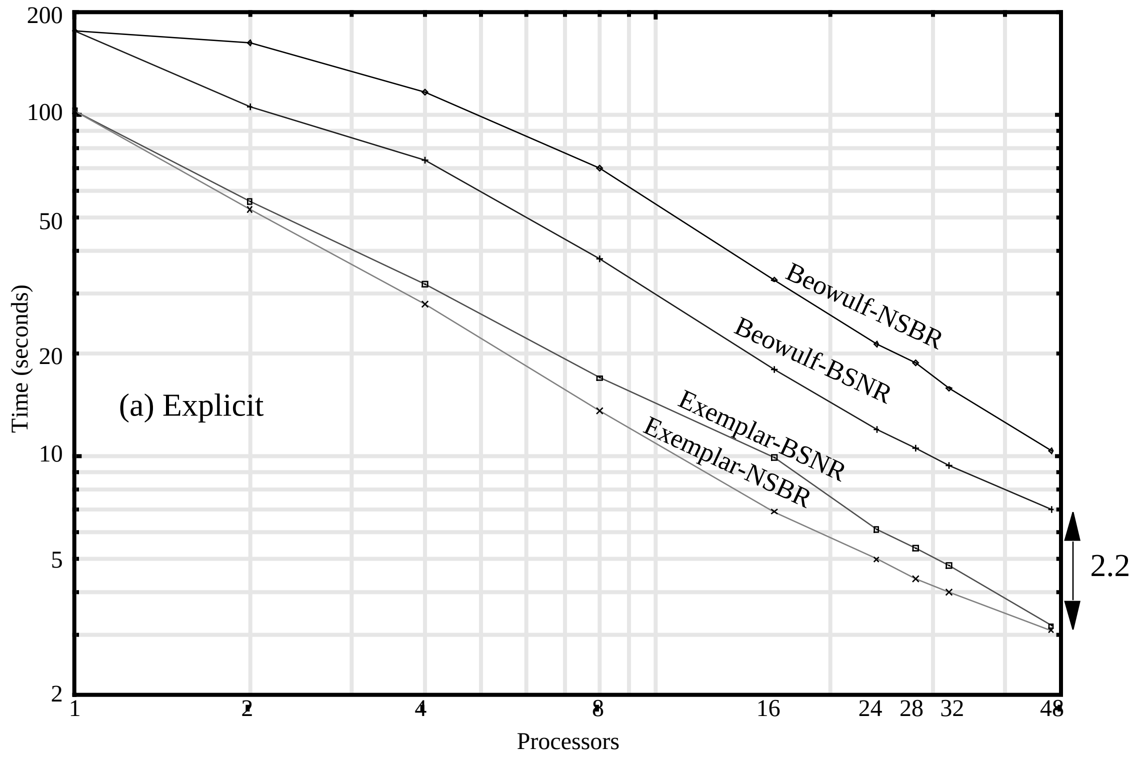
The HP Exemplar at CACR,
http://www.cacr.caltech.edu/local_docs/exemplar/

The Exemplar from Hewlett-Packard,
http://www.hp.com/wsg/products/servers/exemplar/sx-class/index.html

ParNSS: a Compressible Navier-Stokes Solver,
http://www.cacr.caltech.edu/SIO/APPL/eng02.html

R.D. Williams, J. Häuser and R. Winkelmann, *Efficient Convergence Acceleration for Parallel CFD Codes*, Proceedings of the Parallel CFD '96 conference, Capri, Italy, July 1996, also
http://www.cacr.caltech.edu/~roy/papers/capri.pdf

J. Häuser, R.D. Williams, H.-G. Paap, M. Spel, J. Muylaert and R. Winkelmann, *A Newton-GMRES Method for the Parallel Navier-Stokes Equations*, Proceedings of the Parallel CFD '95 conference, Pasadena, California, July 1995, also
http://www.cacr.caltech.edu/~roy/papers/pcfd95.pdf



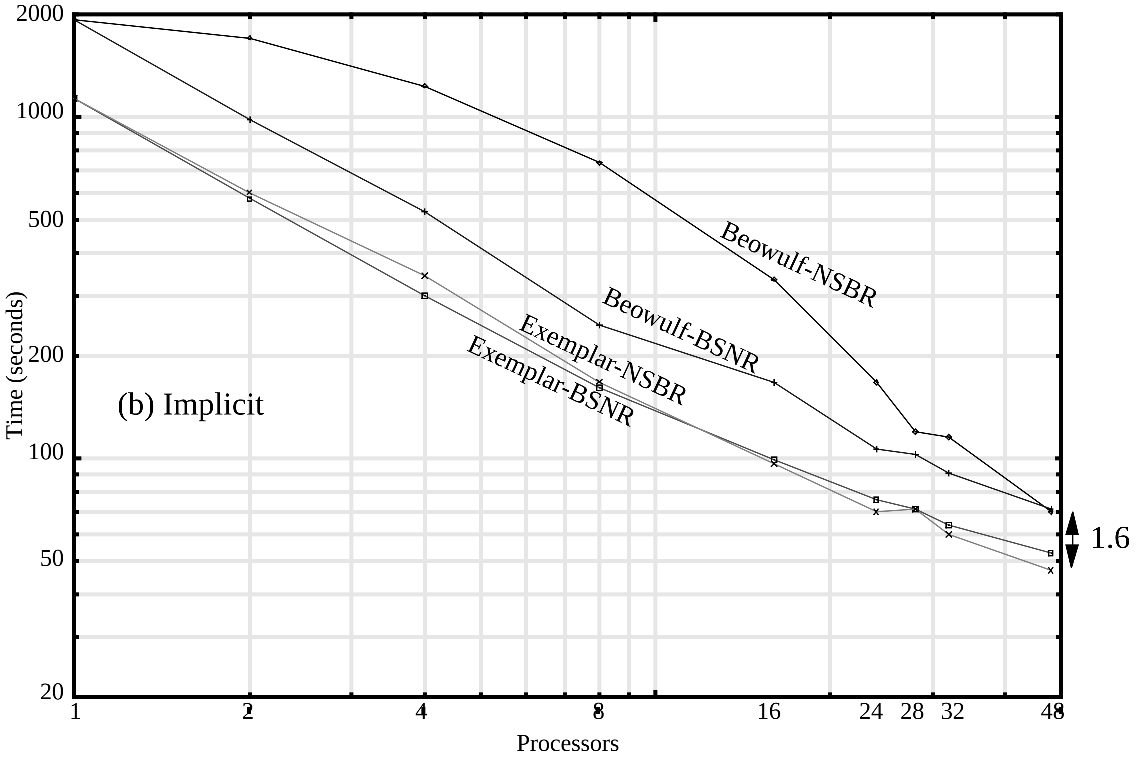


Figure 3: Run times for 10 steps of the ParNSS code, (a) Explicit scheme and (b) Implicit scheme, solving the Euler equations at Mach 9.8 for the X33 model (342,361 point multiblock grid). Each panel shows times for Beowulf and Exemplar, and for the two messaging schemes. In both cases, the Exemplar is roughly twice the speed for the same number of processors.