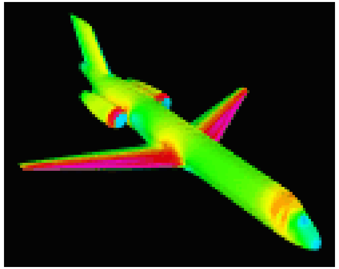
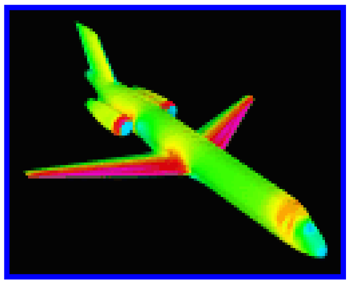
July - August 1996, Vol 2. No. 18





Flow Solver Developed With Extensive Industry Interaction

by Pieter Buning

A Navier-Stokes flow solver, OVERFLOW, has been developed by researchers in the Aeronautical Technologies Division, NASA Ames Research Center, to use overset (Chimera) grids to simulate the flow about complex aerodynamic shapes. It has been optimized to run efficiently on the NAS CRAY C90, and a production capability exists for the IBM SP2 and workstation clusters.

The OVERFLOW code is one representative of a collection of overset grid CFD software programs developed at Ames, including Collar Grid Tools (grid manipulation and surface grid generation), HYPGEN (volume grid generation), PEGSUS (overset grid hole cutting and boundary interpolation), DCF3D (an alternative to PEGSUS, especially for moving grids), and FOMOCO (force and moment calculation for overset grids). Typical customers use most of these tools for their CFD applications; the grid generation process represents by far the largest amount of "calendar time" of a project, though the flow solver consumes all but a fraction of the computer time.

The Chimera grid approach was jointly developed in the 1980s by the Air Force Arnold Engineering Development Center for store separation problems and tunnel support interference evaluation -- and by Ames for evaluating the Space Shuttle launch vehicle. Specifically, early development of OVERFLOW and a variety of the tools were funded by the Space Shuttle Program Office and NASA Johnson Space Center (JSC).

Customers From Industry, NASA

Primary customers of the OVERFLOW flow solver and related software include McDonnell Douglas and Boeing, as well as the NASA Focused Programs for Advanced Subsonic Technology (AST) and High Speed Research (HSR). Interaction with the helicopter industry is handled through the Joint Army/NASA Rotorcraft CFD group, co-located with the Ames Applied Computational Aerodynamics Branch.

Interaction with McDonnell Douglas is on two fronts. McDonnell Douglas Aerospace, Long Beach, CA, is responsible for the C-17 military transport, NASA AST and HSR interaction, and technology development. Douglas Aircraft Company (DAC), also in Long Beach, designs and manufactures commercial transports, including the MD-80 and -90 series and the MD-11 follow-on to the DC-10. OVERFLOW has been useful to McDonnell Douglas in NASA AST work, including propulsion/airframe integration (the effect of engine installation on aerodynamic performance), and high lift (landing and takeoff configurations). DAC's use for the code includes new projects such as the MD-95.

Use of NASA-developed overset grid technology includes military applications such as Joint Strike Fighter and Dark Star in Boeing's Defense and Space Group, and propulsion/airframe integration for HSR designs in their Commercial Airplane Group.

Within NASA, the AST Program "Integrated Wing Design" Element uses OVERFLOW heavily. Propulsion/airframe integration research, high-lift development, and aerodynamic design cycle methodology work all are focused on the overset grid approach.

Industry Interaction Model

In developing OVERFLOW into a production CFD capability, two issues are addressed: The first is to streamline the usability of the code, while maintaining access to enough input controls ("knobs and buttons") to allow the testing of new algorithms. Second is to integrate new capabilities into the code as soon as possible, once the usefulness of the options has been shown. Examples of this are new algorithms and better turbulence models. Recommended parameter settings also change with new applications or increased experience. With these changes, the code is somewhat of a "moving target" to users, occasionally leading to frustration.

Feedback from industry and NASA customers is very important in the development process. In the field of CFD, the two areas that always need improvement are speed and accuracy. Code speedups come from algorithm improvements, computer hardware advances, and code efficiency. Evaluation of code accuracy for a given application, including required grid densities and number of iterations needed for convergence, can best be determined by the customers themselves. Indeed, even with recommended settings, this validation process must be carried out by each company to build the confidence in the software necessary for use on a project. In particular, each company has configurations of interest on which they can try out code improvements, and wind tunnel data with which to compare results.

Often with experimental results (as with CFD), data accuracy can be judged best by those with knowledge of the specific test conditions and fidelity of the model. With OVERFLOW, the companies have shared the results of their experiences, allowing code improvements and tuning to be made. In this way, industry is an integral partner in the code development and validation process.

Recommendations from industry customers on computing platforms are also addressed. At DAC, for example, many Hewlett Packard workstations are dedicated to CAD during the day but are unused at night. The ability to use these as workstation clusters could significantly reduce CFD computing budgets. In addition, the need for robustness of such a code is stressed. To paraphrase a Douglas Aircraft engineer's question: "If our CAD designers can't sleep, come in at 2:00 a.m. and find their workstation busy, they may simply push the reboot button. Can the CFD job recover and still finish by morning?"

With one-day turnaround (or more), the answer to such a question can determine whether a CFD code is useful or not. For this reason, the PVM (Parallel Virtual Machine) library was chosen for development of a production workstation cluster capability for OVERFLOW. Earlier work by former Ames researchers Merritt Smith and Chris Atwood (sponsored by NASA's High Performance Computing and Communications Program) showed good performance using the OVERFLOW/PVM code, an advanced forerunner of the current production capability. The PVM version of OVERFLOW has also proven usable on the NAS parallel machines, including the IBM SP2 (babbage) and the Silicon Graphics POWER CHALLENGE cluster (davinci).

Many companies use NAS computer time to carry out this validation process for NASA-developed codes. This allows testing of codes for feedback to NASA, without impacting company computer resources, which may be dedicated to product design. Industry testing of new computer systems is also done using NAS systems such as babbage and davinci; decisions on future computer purchases and even design methodology may be made based on system performance.

Beyond the testing and validation stage however, production CFD is often focused on wall-clock time. Computer availability and raw throughput are critical to completing projects on time. For this reason, much effort has been expended on OVERFLOW development and tuning to run on the CRAY C90. The code has been specifically structured not only for good vector performance but has been multitasked to use multiple CPUs even for a single block grid.

Typical performance on vonneumann gives 450 megaflops per processor, with an average of 6 CPUs in use concurrently. Coupled with a low in-core memory requirement, this routinely allows the turnaround of an 8-hour job in 1.5 hours during peak time. Further advantages on vonneumann are the higher priority of the multitasking queue and the "discounted rate" charged for multitasking jobs (jobs are charged for four concurrent CPUs, regardless of how many they use; thus an 8-hour OVERFLOW job would typically only be charged for 6 hours).

While these types of features have proven extremely useful for NAS users, such priorities and billing practices are not universal. Indeed, throughput on any particular computer system is dependent on a variety of factors, including how heavily loaded the system is. In judging the usefulness of code characteristics such as memory, CPU, and I/O usage, the computing environment can create a "reality" which encourages use of one code over another. This makes code development decisions difficult.

Response to Customer Requirements:

Besides the implementation of Cray multitasking and a workstation cluster capability using PVM, several other code improvements have been recently. A multigrid algorithm has been implemented in the production version of OVERFLOW to improve convergence to a steady-state solution. Tests on simple problems have been encouraging, showing a factor of 2-4 speedup. While other cases have been disappointing, the code has been released to selected users for feedback. An industry HSR problem showed an immediate factor of 2 reduction in CPU time, with further improvements likely.

Another modification has added low Mach number preconditioning, a technique to accelerate convergence for cases with freestream Mach numbers below 0.2. This capability is specifically aimed at high-lift applications for the AST Program, and evaluation by industry partners is underway in conjunction with further testing at Ames. Other applications are already benefiting from the addition: recent tests by JSC for parachute recovery of a Space Station Crew Return vehicle have been assisted by the feature. "Works like a charm," was the response of Ray Gomez, a researcher at JSC.

Interaction with NAS

All of this work has involved significant interaction with the NAS staff. Clayton Guest (scientific consulting group) worked on improving Cray multitasking performance, and Jerry Yan (parallel tools group) is measuring communications efficiency for OVERFLOW on a workstation cluster. Parallel implementations of OVERFLOW from Sisira Weeratunga (applications and tools group) and Smith led the way to the current production code. It is important for the NAS Facility to maintain PVM on the parallel machines for codes where the improved performance of MPI is deemed less critical than robustness and portability to workstation cluster platforms.

While versions of OVERFLOW run on NAS platforms from the C90 to the SP2, specific differences in their operation remain. Usage on the C90 is the easiest, with fast wall-clock turnaround and low memory requirement with SSD (solid-state storage device) use. On the SP2 (and workstation clusters in general), there are two limits to efficient operation of the code. First, given the 128-megabyte memory size of the average node, individual grid blocks are limited to about 400,000 points. This partitioning requirement can be met by manual splitting of grids and input file modification, but hampers easy porting of applications from the C90.

The second limit is load balancing. The current OVERFLOW implementation puts one grid block on each node. If some blocks are significantly smaller than others, those processors will finish early and be forced to wait. This could be mitigated by either running several small blocks on a single node (not currently possible on the NAS SP2), or splitting larger blocks to achieve a more even set of block sizes. Again, this option currently involves manual intervention. Even with these changes, efficient usage of 50 SP2 nodes would be necessary to equal the throughput of a job on the C90.

With its capability for multitasking as well as parallel operation using PVM, OVERFLOW should perform very well on the new NAS cluster of CRAY J90s, called Newton. (See *CRAY J90 Cluster To Help C90 Users `Parallelize' Codes,* NAS News, March-April '96.)

Contributions from Many Researchers

OVERFLOW has a long line of ancestry, and includes input from many CFD researchers at Ames and elsewhere. Dennis Jespersen developed the PVM production code and the multigrid implementation. Jespersen and Tom Pulliam (both in the Ames Advanced Computational Methods Branch) are responsible for the low Mach number preconditioning algorithm -- with pioneering work done at Penn State. Tom Shieh (a researcher at JSC) did initial multigrid work. Contributions to expanding OVERFLOW's capabilities include the coupling of the CDISC inverse design code by Steve Krist at Langley Research Center, development of an automated grid partitioning scheme for the SP2 and workstation clusters by Dan Barnette (Sandia National Laboratories, Albuquerque) and addition of an aeroelastic deformation capability by Guru Guruswamy (Applied Computational Aerodynamics Branch).

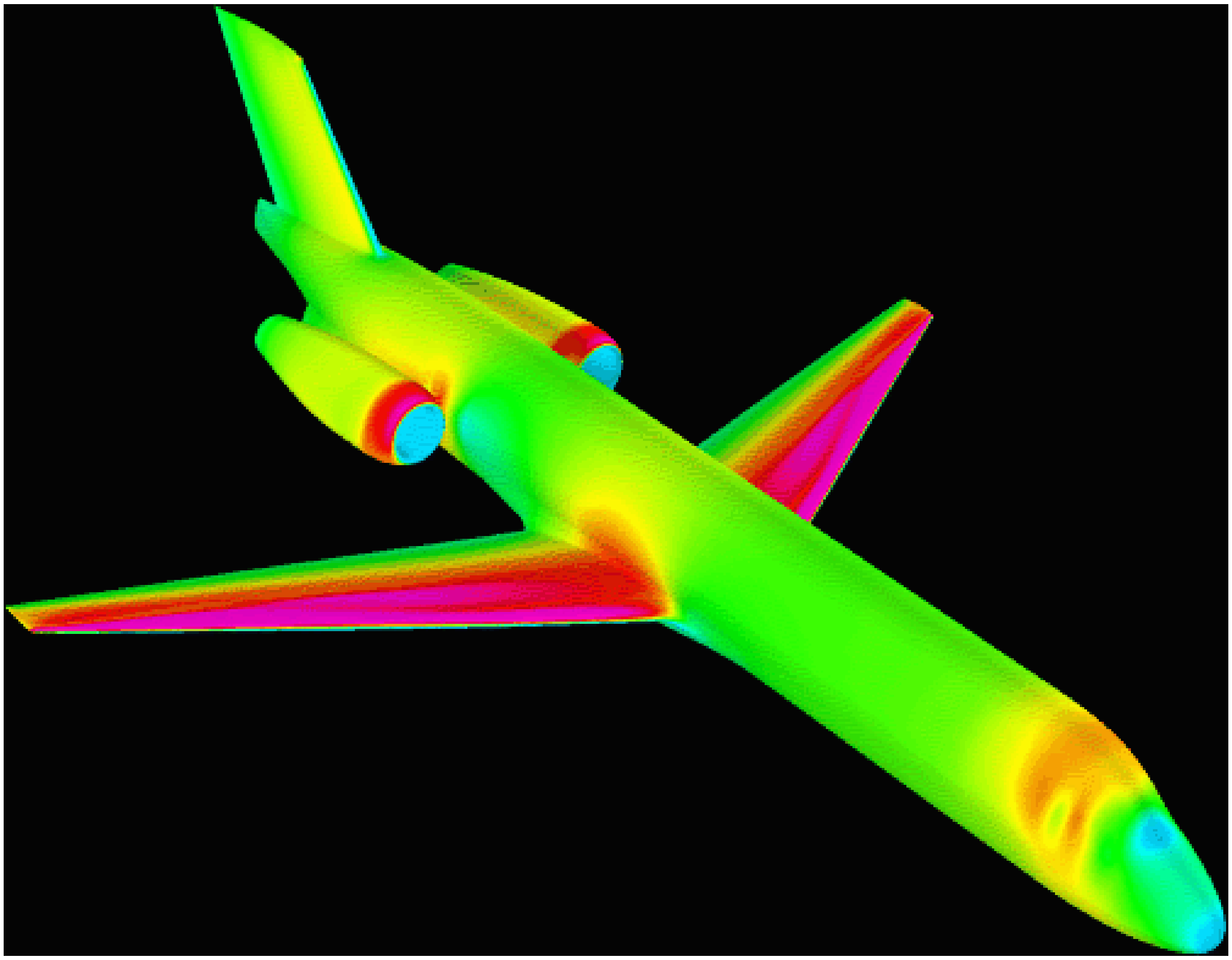
Further Increases in Speed Required

Future improvements must focus on capabilities in the area of design optimization and multidisciplinary applications, including the coupling of structural analysis, and determination of stability and control derivatives. With the increased computational requirements of these new goals, there remains a critical need for increases in computational speed. These improvements will come from new computer hardware, as well as algorithm changes.

In the application of CFD to industry and NASA problems in aerodynamics, our researchers will continue to strive for a better understanding of the design environment, and continue to listen to our customers.

Pieter Buning, a research scientist with the NASA Ames Design Cycle Technologies Branch, has been at Ames for 16 years, where he developed the PLOT3D graphics program. Work on the Space Shuttle launch vehicle in 1987 spurred his interest in aerodynamic applications and overset grid technology

Next Article | Contents
Main Menu | NAS Home

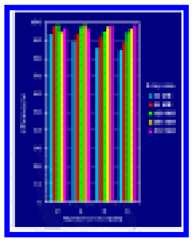


Surface pressure on a McDonnell Douglas MD-95, computed using OVERFLOW. The solution used 4 million grid points in 10 grids, and required 27 CPU hours and overnight turnaround on the CRAY C90.

Graphic courtesy of John Vassberg, Mark DeHaan, and Doug Friedman, Douglas Aircraft Co; created with ARTIS.

Back to the article

I/O Performance of Multidimensional Arrays Tests Well on IBM SP2 With Panda



by Elisabeth Wechsler

A group of scientists based at the University of Illinois who are developing a software library to improve I/O performance on multidimensional arrays have found the IBM SP2's "clean environment" a boon for their research.

"An environment that produces consistent results is very important," said Marianne Winslett, principal investigator and faculty advisor to the Panda (Persistence AND Arrays) project team. She attributed this outcome in part to the SP2's homogeneously configured processors.

"Usually, researchers working on parallel I/O use `best' times in reporting test results," Winslett continued, "but we've been able to use `average numbers' because the test results were so consistent. This has allowed us to get to the heart of the problem and make improvements much more quickly."

Panda uses data management techniques to improve I/O-intensive scientific computing applications. The project is funded by NASA, ARPA (Advanced Research and Projects Agency), and the National Science Foundation. Since 1993, the project team has run tests on several parallel systems around the country, including other SP2s.

Getting Data On and Off Disk `Fast'

"With these powerful parallel computations, we want I/O nodes to access data on disk as fast as the underlying disk and operating system technology will allow," Winslett said, adding that currently available software libraries for I/O on parallel platforms "often fail to give high throughput."

In addition to high-performance array I/O operations, the project's goals include ease of use and application portability, according to Kent Seamons, former Panda project team member now with a research group at Transarc Corp., Pittsburgh.

"Each node of the NAS SP2 has a local disk for I/O, in contrast to earlier parallel computers with external I/O nodes," said Seamons, who conducted Panda research on the SP2 and Intel iPSC/860 at the

NAS Facility in 1993-94 under an ARPA high-performance computing fellowship. "Being able to easily experiment with different numbers of I/O nodes on the SP2 allowed us to test the scalability of our approach."

Server Directed I/O

"The combination of Panda's easy-to-use, general interface with server-directed I/O techniques allows applications to run on different platforms and experience high performance, since Panda transparently exploits each platform's advantageous characteristics," Seamons said. Testing has shown that a wide range of applications, such as flow solvers doing time-dependent simulations and applications checkpointing their own data, can get good performance using Panda.

The availability of commodity software, such as Message Passing Interface (MPI), contributed to the SP2 as a productive test platform. "We could debug Panda on local workstations, then move our code directly to the SP2 for final testing -- enabling us to spend time conducting our experiments rather than debugging the application," Seamons continued.

"When reading and writing very large multidimensional arrays, we routinely achieve a utilization rate above 90 percent of the underlying peak AIX [IBM UNIX] filesystem throughput with server-directed I/O on the SP2," said Ying Chen, a doctoral candidate working on the Panda project. "Typically, applications get less than 50 percent of the filesystem utilization when using traditional parallel filesystems." She added that there have been few performance results available from other parallel I/O systems and Panda's results on the SP2 are "impressive."

The most recent version of Panda (written in C++) will allow the team to plug in different algorithms for a variety of system architectures and applications. Chen hopes that in the future Panda can automatically select an optimal strategy for providing high-performance I/O. "If you have applications with certain characteristics, Panda can choose from among several algorithms to give the best I/O performance," Chen said.

Load Balancing for Black Holes

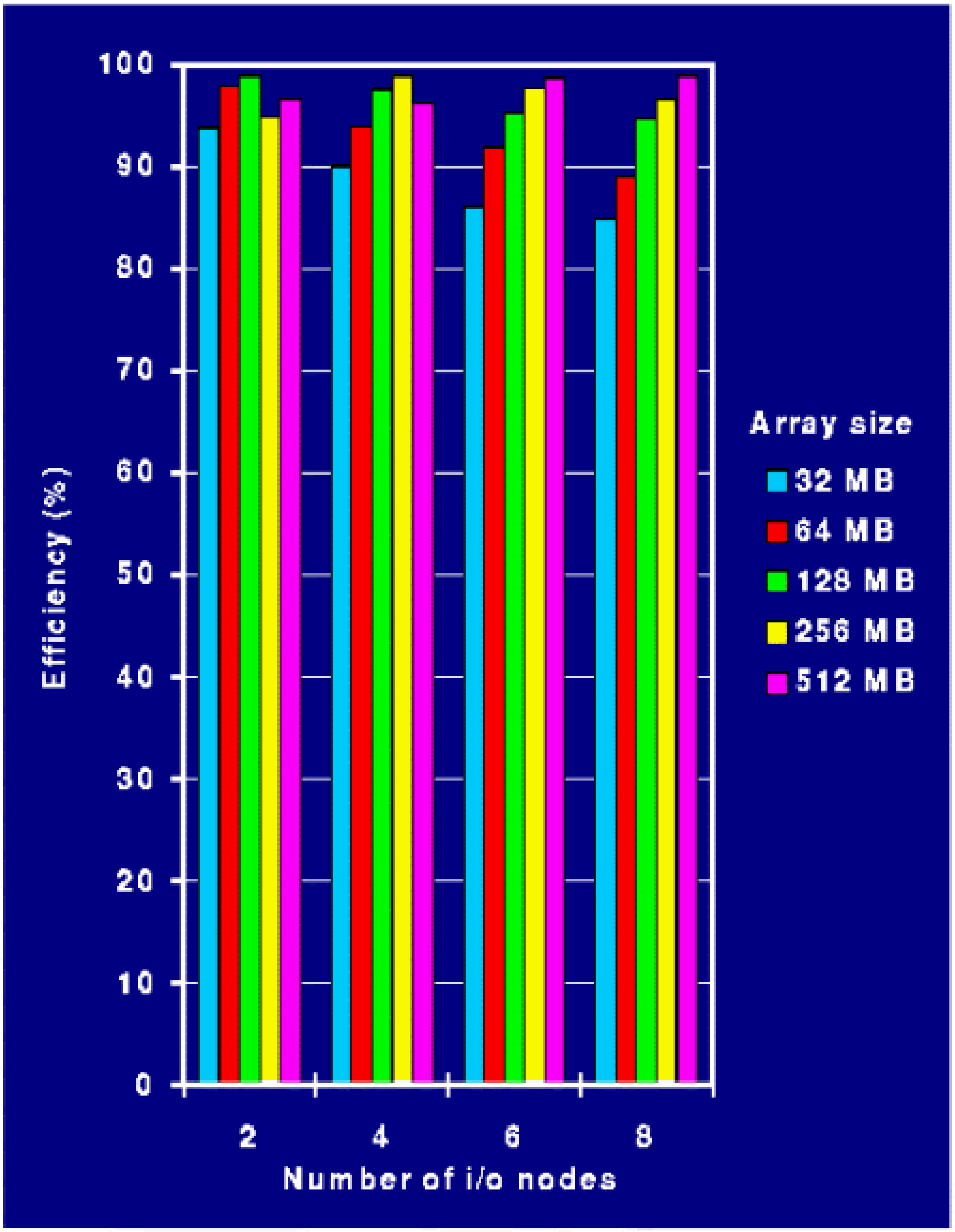
The Panda team is currently experimenting with black hole applications at National Center for Supercomputing Applications (University of Illinois) utilizing advanced techniques such as adaptive mesh refinement (AMR), Winslett said. Among other challenges, this effort involves unevenly distributed data in which load balancing is an issue.

"When an AMR application wants to output its data, Panda must dynamically determine which data is located on which processors and somehow balance the I/O load across all I/O nodes," Winslett said. "To complicate matters, the black hole scientists want to run such large simulations that they need to harness several parallel computers at once. This raises issues of heterogeneity and load balancing for I/O that we hadn't needed to address when we ran only on the NAS SP2. Fortunately, the clean environment on the

NAS SP2 prepared us to address the real-world challenges posed by heterogeneity," Winslett added.

Editor's note: *The IBM SP2, operated and maintained by NAS Facility staff, is funded by the High Performance Computing and Communications Program.*

Next Article | Contents | Main Menu | NAS Home



This graph shows how the Panda I/O library has achieved 85-99 percent efficiency using the AIX filesystem to output multi-dimensional arrays in parallel on the IBM SP2. The application program used for these experiments requested that Panda output a 3D array spread across 48 processors to multiple I/O nodes, so that the data is written in parallel. The experiments used five different array sizes (32MB, 64MB, 128MB, 256MB, 512MB) and four combinations of I/O nodes (2, 4, 6, 8) to show that Panda scales well.

Graphic courtesy of Kent Seamons.

Back to the article

p2d2 Simplifies Debugging on the IBM SP2

by Robert Hood

Editor's note:*In an ongoing project supported by the Computational Aerosciences Project of NASA's High Performance Computing and Communication Program, the NAS parallel tools team has designed and built p2d2, a portable parallel/distributed debugger. Currently, p2d2 is running on the IBM SP2, and there are plans for porting it to the other parallel testbeds at the NAS Facility. (See* NAS News, *January-February '96 for background.)*

One of the principal challenges facing the p2d2 design team was making the debugger usable in situations where many processes are being controlled. This usability problem boils down to handling the two primary functions of a debugger: The first is process control, where the user can start execution and describe circumstances under which it should stop. This is done by issuing commands to set breakpoints, execute a single source statement, or continue to the next breakpoint. The second function is state examination, where variable values can be scrutinized.

Any debugger used to control a computation involving many processes must provide these functions in a way that scales. In p2d2, scalability is achieved by applying process control directives to more than one process. Similarly, state examination requests can be applied to multiple processes simultaneously, and then the results can be abstracted.

Process Control

In a computation involving many processes, users need a simple mechanism for sending a control directive (such as Continue and StepOver) to a set of processes. In p2d2, the collection of processes that will receive control directives is called the control set. Users can toggle a process's inclusion in this set by clicking the mouse in its grid location. When a control button (such as Continue) is pressed, the command is sent to the processes in the control set.

State Examination

p2d2 provides several tools for examining state. The simplest one is the Evaluate command, which permits users to select or type an expression and have it evaluated on all processes in the control set. The result shows up as a simple list in the output area. For example, evaluating x+1 might yield:

```
(for selected processes) x+1 =
0> 11
1> 21
3> 1
8> 3
(done)
```

where the selected processes are 0, 1, 3, and 8.

In addition to this simple one-time evaluation mechanism, there is a tool for watching data values continuously. The scalar data display is essentially a table that lists a collection of expressions in one column and their associated focus process values in the next column. The values get updated every time the focus process stops.

p2d2 also provides an array-viewing tool that presents a two-dimensional "slice" of an array in a scrollable window. The values in the focus process are displayed and are updated whenever the focus process stops. Users can change the slice that is being viewed and can transpose the values in the display.

One way to compare data values between processes is to use the process grid. Instead of representing stopped and running processes with red and green squares (as in a previous example), the grid can be configured to represent a process with an icon that depends on the values of variables within the process. For example, processes where the expression "x.eq.10" is true could be represented with checkmarks. This allows users to get a quick overview of a small part of the program state in all of the processes at once.

Navigating the Program Source

p2d2 provides a source browser called the Program Navigator to help users cope with the information management problems presented by large scientific programs. This tool provides an overview of the program source that is essentially a configurable program listing.

By default, the Program Navigator contains a list of all user files that make up the program. The display also lists the subroutines in each file and the breakpoints in each subroutine. The entry for a breakpoint includes its line number in the file and the text of that particular line of code.

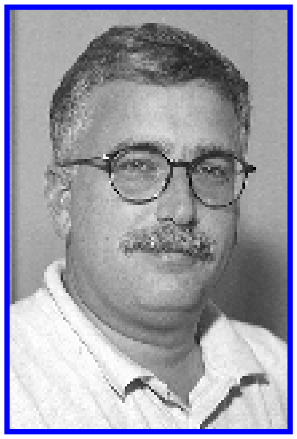
Besides showing file names, subroutine names, and breakpoints, the Program Navigator supports a search operation that will insert all program source lines matching a pattern into the listing. This can be used to generate a list of all program locations that refer to a particular variable or call a function.

For convenience, when a line of the Program Navigator display is selected by a mouse click, the corresponding location is shown in the focus process source area of the main window.

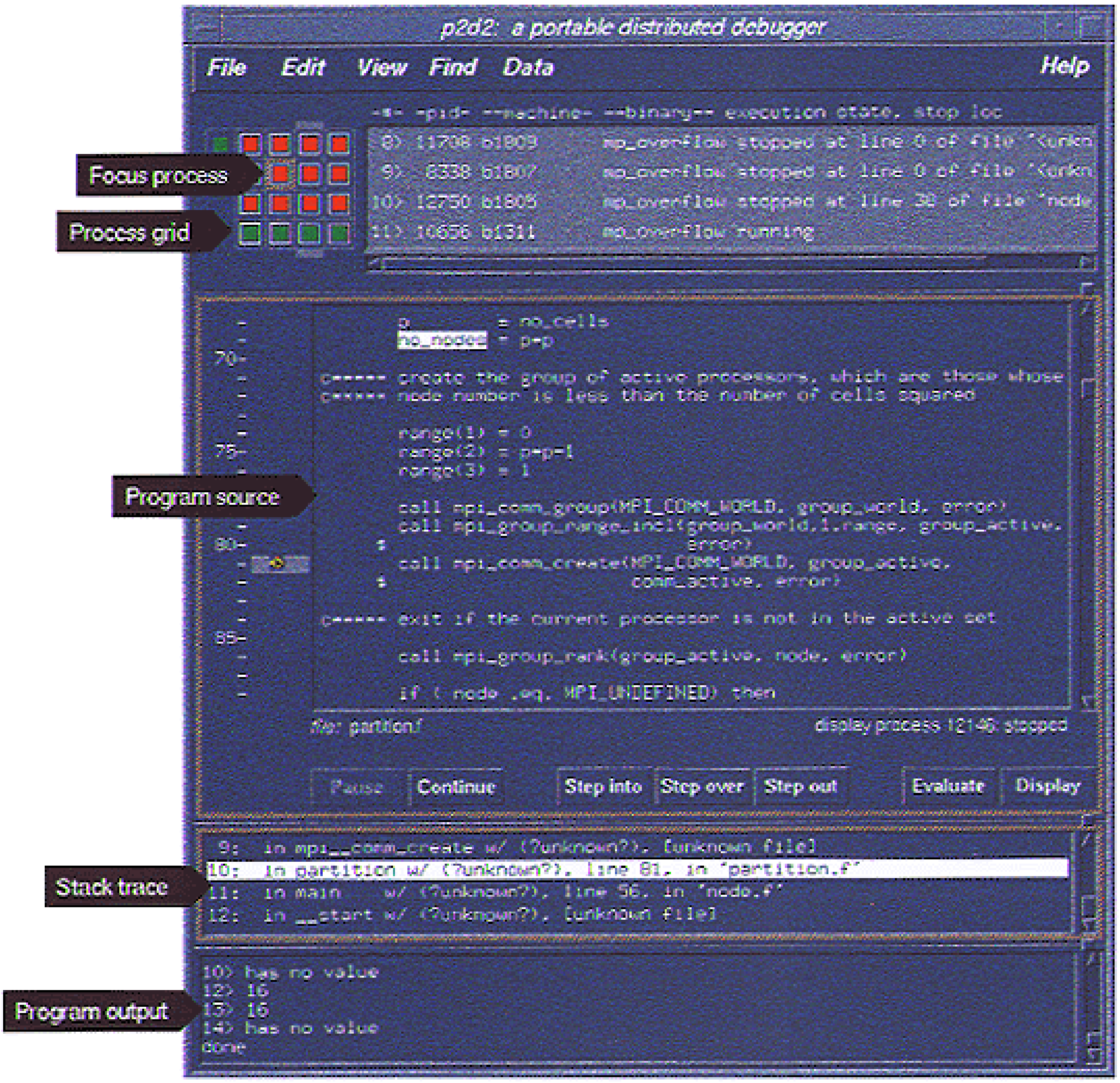
Second Test Phase

At present, p2d2 is undergoing a second phase of select user testing, and an open testing period was tentatively scheduled to begin in June.

Robert Hood has been a member of the NAS parallel tools team for three years. He began researching tools for debugging computationally intensive programs in 1986 while on the faculty at Rice University.



Next Article | Contents | Main Menu | NAS Home



```
-#- -pid- --machine- --binary-- execution state, stop loc
 8) 11708 b1809      mp_overflow stopped at line 0 of file `<unkn
 9)  8338 b1807      mp_overflow stopped at line 0 of file `<unkn
10) 12750 b1805      mp_overflow stopped at line 38 of file `node
11) 10656 b1311      mp_overflow running
```

```
            p          = no_cells
            no_nodes   = p*p

c***** create the group of active processors, which are those whose
c***** node number is less than the number of cells squared

            range(1) = 0
            range(2) = p*p-1
            range(3) = 1

            call mpi_comm_group(MPI_COMM_WORLD, group_world, error)
            call mpi_group_range_incl(group_world,1,range, group_active,
     $                                error)
            call mpi_comm_create(MPI_COMM_WORLD, group_active,
     $                           comm_active, error)

c***** exit if the current processor is not in the active set

            call mpi_group_rank(group_active, node, error)

            if ( node .eq. MPI_UNDEFINED) then
```

file: partition.f   display process 12146: stopped

```
 9:  in mpi__comm_create w/ (?unknown?), [unknown file]
10:  in partition w/ (?unknown?), line 81, in `partition.f'
11:  in main     w/ (?unknown?), line 56, in `node.f'
12:  in __start w/ (?unknown?), [unknown file]
```

```
10> has no value
12> 16
13> 16
14> has no value
done
```

Main Window For Primary Interaction

p2d2's main window (depicted in graphic at right) is the primary location for user interaction with the running program and is divided into several components. The process grid (upper left corner of window) represents all processes in the computation. Each process in the computation is represented by an icon. The particular icon used depends on the process's characteristics and how the grid has been configured. For example, the default grid configuration represents running processes with green squares and stopped processes with red squares.

The process grid is also used to select a single process, called the focus process, for detailed examination.

In the main window, the focus process area consists of a program source display and a stack trace display. The program source display shows the code being executed in the focus process. The stack trace display lists all the active calls to functions and subroutines in that process.

The program output area (bottom area of window) displays the output of the program that would normally appear on the terminal if run outside the debugger. In addition, it is used to display the results of certain debugger commands, such as Evaluate.

Back to the article

Early MPI-IO Implementation Demonstrates Portability and Performance

by Parkson Wong

The March-April and May-June issues of *NAS News* reported on NAS's role in defining MPI-IO and how MPI-IO could be used to write out distributed arrays. This article describes the architecture and performance of PMPIO, a portable MPI-IO implementation developed by the NAS parallel systems group to serve as a tool for designers and early "adopters" of MPI-IO.

PMPIO was designed to be extremely portable so that it could be tested on a wide variety of massively parallel processors and cluster systems. It also maintains compatibility with the underlying UNIX file system so that users' files can still be accessed by other programs not using MPI-IO. To date, PMPIO has been ported to the IBM SP2, Silicon Graphics Inc. (SGI) and Sun Microsystems shared-memory workstations, an Intel Paragon, and the NAS CRAY J90. This portability allows the group to measure the performance characteristics of MPI-IO on vastly different architectures.

PMPIO is divided into the following independent and replaceable components: collective buffering, client-server protocol, communication transport, support routines, and server implementation. Multiple implementations of the different components can co-exist. Calls from one component to another are done through call tables, so they can be set to default values or changed at run time (when a file is open). For example, a different collective buffering algorithm could be used for each open file, depending on the data distribution pattern. Another use for this feature is to allow the client to talk to different server implementations. For example, this allows an MPI-IO program to write temporary files to a parallel file system, and to write result files to a mass storage system through a server for later analysis or visualization.

Outperformed Standard Fortran I/O

The performance of the NAS Application I/O benchmark (BTIO) on various systems is shown in the table below, using PMPIO as well as Fortran I/O. Using this benchmark, PMPIO has outperformed standard Fortran I/O on all systems tested -- and on some systems the speedup is dramatic. The I/O portion of BTIO was implemented using three different approaches, which were run using the class B problem sizes.

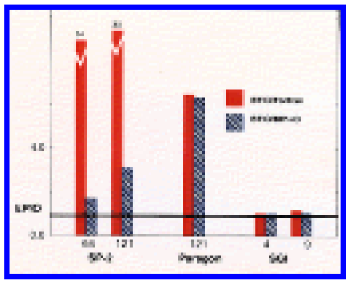
BTIO/Fortran uses standard Fortran unformatted direct I/O to write the solution matrix to a single file. The record size was set to the largest unit (40 bytes) that can be used to address the file and still support the data distribution used by BT. To use Fortran I/O, it was necessary to make repeated write operations of small units of data, since Fortran can only write out contiguous chunks of the file.

BTIO/MPI-IO used MPI-IO's collective write operation (MPIO_Write_all) to write out the entire solution matrix at once to a single file. MPI datatypes were created using MPI-IO's subarray type constructor to describe both the layout of the array in memory (buftype) and the layout of each processors pieces of the array in the file (filetype).

The parallel systems group also implemented an idealized benchmark, EPIO, which mimics the BTIO benchmark in terms of the volume of data transferred, but does none of the detailed data placement required of the actual problem. In this way, EPIO provides a best possible performance gauge for comparisons.

These experiments were run on the 160-node IBM SP2 and the SGI POWER CHALLENGE cluster at the NAS Facility, as well as a 512-node Paragon located at Oak Ridge National Laboratory. All nodes on the SP2 were "wide" nodes, with 128 megabytes (MB) of RAM running AIX 4.1.3 and PSSP 2.1. The data was stored on the SP2's Parallel I/O File System (PIOFS), which is served by three I/O server nodes. The Paragon ran OSF/1 AD R1.4 beta. Each node had 32 MB of RAM, and files were stored on Intel's Parallel File System (PFS), served by 16 I/O server nodes. The SGI cluster was a 10-processor, shared-memory R8000 based system with 4 GB of RAM running IRIX 6.2. Files were stored on a local XFS file system.

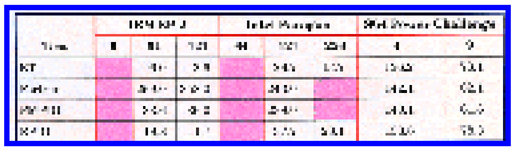
The codes used were not vendor optimized and, since PMPIO only runs under the collaborative Argonne National Laboratories/Mississippi State University implementation of MPI (MPICH), the vendor-optimized MPI libraries available on these systems were not used. In addition, only a few machine-specific optimizations were performed on PMPIO -- so, it is expected that these results could all be improved with further optimization.



Effect of Collective Buffering

The effect of PMPIO and collective buffering on different types of parallel file systems is shown in the figure at left. These results are normalized for the optimal (EPIO) implementation to compare the improvement that collective buffering provides for the benchmark. In addition, they show how close the performance of collective buffering is to ideal performance.

As can be seen in the figure at right, the performance of PIOFS improves dramatically with collective buffering. This is due to severe degradation caused by small write operations (approximately 100-800 bytes) used in the Fortran version of BTIO.



	IBM SP2			Intel Paragon			SGI Power Challenge	
Time	[illegible]	64	121	[illegible]	121	256	4	9
BT		[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	[illegible]
Fortran		[illegible]	[illegible]		[illegible]		[illegible]	[illegible]
MPI-IO		[illegible]	[illegible]		[illegible]		[illegible]	[illegible]
EPIO		[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	[illegible]

The Paragon did not perform as well as the SP2, but collective buffering still improved performance significantly. The SGI is different than the other two architectures in that it has a very large (4 gigabytes) shared memory, and disk buffer space is shared. Although not as significant, performance was still enhanced by collective buffering on this system.

Latest Developments

At an April meeting of the MPI Forum, members decided to add an I/O chapter to the MPI-2 standard. At the Forum's June meeting, NAS staff presented MPI-IO as a starting point for this effort. The Scalable I/O Initiative has already adopted MPI-IO as its mid-level API in May, and will work with the MPI Forum to create a common parallel I/O interface standard.

Parkson Wong has been a member of the NAS parallel systems group since 1993. Prior to joining the NAS Systems Division, he worked for over 10 years in the computer industry as a systems architect.

Simulations on C90 Predict Rapid Regeneration of Earth's Ozone Layer -- With Caveats

by Elisabeth Wechsler

A Stanford University researcher running an atmospheric model on the NAS CRAY C90 to study the Earth's ozone layer has found that, if the ozone layer were entirely destroyed and the depleting substance removed, half of the layer could be regenerated within ten days, and the rest within nine months. Of course, there are many caveats to these findings and more research needs to be done, said Mark Z. Jacobson, assistant professor of civil engineering at Stanford.

One caveat is the assumption that all chlorine can be removed from the atmosphere. In reality, natural removal of chlorine is a long-term process. Artificial removal is "not feasible" and chlorine is continuously being added to the present stratosphere. Still, running the simulation without chlorine "gives an upper limit to the rate at which the ozone layer can regenerate if chlorine is removed. If chlorine could be removed from the present-day stratosphere, for example, the ozone layer should regenerate to its level of 30 years ago in less than a year," he said.

To test the effect of chlorine on the rate of regeneration, a simulation was run with ozone removed but with the present-day chlorine levels in the atmosphere. The result was that the ozone layer regenerated at the same rate, but to a level approximately 5-10 percent lower than the level found without chlorine. Thus, the presence of chlorine destroyed part of the regenerated ozone layer -- which was expected, Jacobson said.

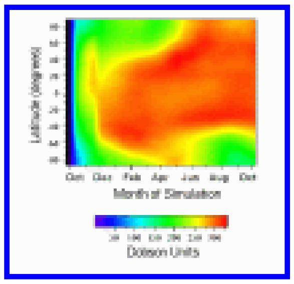
Meteorology, Chemistry Affect Ozone

Both meteorology and chemistry play a role in the distribution and buildup of ozone in the stratosphere. To test the relative effect of each on the rate of ozone regeneration, Jacobson compared results from a simulation without ozone transport (primarily, wind) to one with ozone transport. The comparison showed that, when ozone transport was ignored, 90 percent of the ozone layer still regenerated chemically. "This shows that, while transport was responsible for shifts for the geographic distribution of ozone, it had a relatively small net effect on ozone production in the stratosphere," he continued.

The simulations described were carried out with an atmospheric computer model called GATOR/AGCM (gas, aerosol, transport, and radiation/atmospheric general circulation). Three primary processes -- gas chemistry, radiation, and meteorology -- were interactively combined for the simulations, which placed heavy demands on computer time. The first set required 5 days of computer time per year of simulation to run on the C90. Subsequently, when his NAS allocation ran out, Jacobson has used a smaller CRAY J90 at Stanford, which requires 25-30 days per year of simulation.

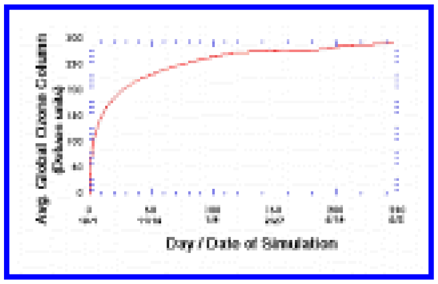
Sunlight is Primary Producer

The simulation divides the atmosphere into 44 north-south by 72 east-west grid boxes stretching around the globe, as well as 17 vertical layers stretching from the Earth's surface up to about 52 kilometers in altitude. The simulations were initialized for October conditions, when peak sunlight intensity occurs south of the equator.



Immediately, ozone began to form most intensely over latitudes receiving direct sunlight; however, it also formed in regions receiving less intense light. As it formed, transport moved some of the ozone towards the poles, Jacobson explained, showing that "a combination of chemical reactions and transport contributed to the reformation and redistribution of ozone in the stratosphere." (Figure 1 shows the regeneration rate of ozone over time at each latitude.)

Preliminary results indicate that if both ozone and ozone-depleting gases are removed from the atmosphere, ozone would regenerate almost completely in less than a year (see Figure 2). However, Jacobson said, "In reality, ozone-depleting gases such as chlorine cannot be removed easily from the atmosphere."



Ozone Blocks Radiation

The stratospheric ozone layer protects life on the Earth's surface from harmful solar radiation by absorbing these rays before they penetrate below the surface. Without the ozone layer, the most intense radiation passing through the atmosphere would immediately destroy phytoplankton and other small organisms, and would eventually destroy exposed higher organisms, Jacobson explained.

A 1-2 percent decrease in ozone levels increases the presence of dangerous ultraviolet rays in increments of 2-4 percent, he continued. From 1979 to 1994, the global stratosphere ozone layer was depleted at a rate, ranging from approximately 4-4.5 percent for the band between 60 degrees N and 60 degrees S latitude, to 5 percent for 30-60 degrees N-S latitude, and only about 2 percent near the equator. In October 1993, measurements over the South Pole indicated a 70 percent depletion of ozone, covering a hole the size of North America.

Jacobson believes that concern about the reduction of the ozone shield over the globe is still "very appropriate." While the simulations showed that, if chlorine is removed from the atmosphere ozone can regenerate quickly, removing chlorine is not a trivial task. "The first step is to further reduce human-made emissions of chlorine," he said.

The Montreal Protocol, an international agreement signed in 1987 (with amendments in 1990 and 1991), has resulted in scheduled limitations for chlorofluorocarbon emissions. For more information, send email to **jacobson@cive.stanford.edu**.

Next Article | Contents | Main Menu | NAS Home

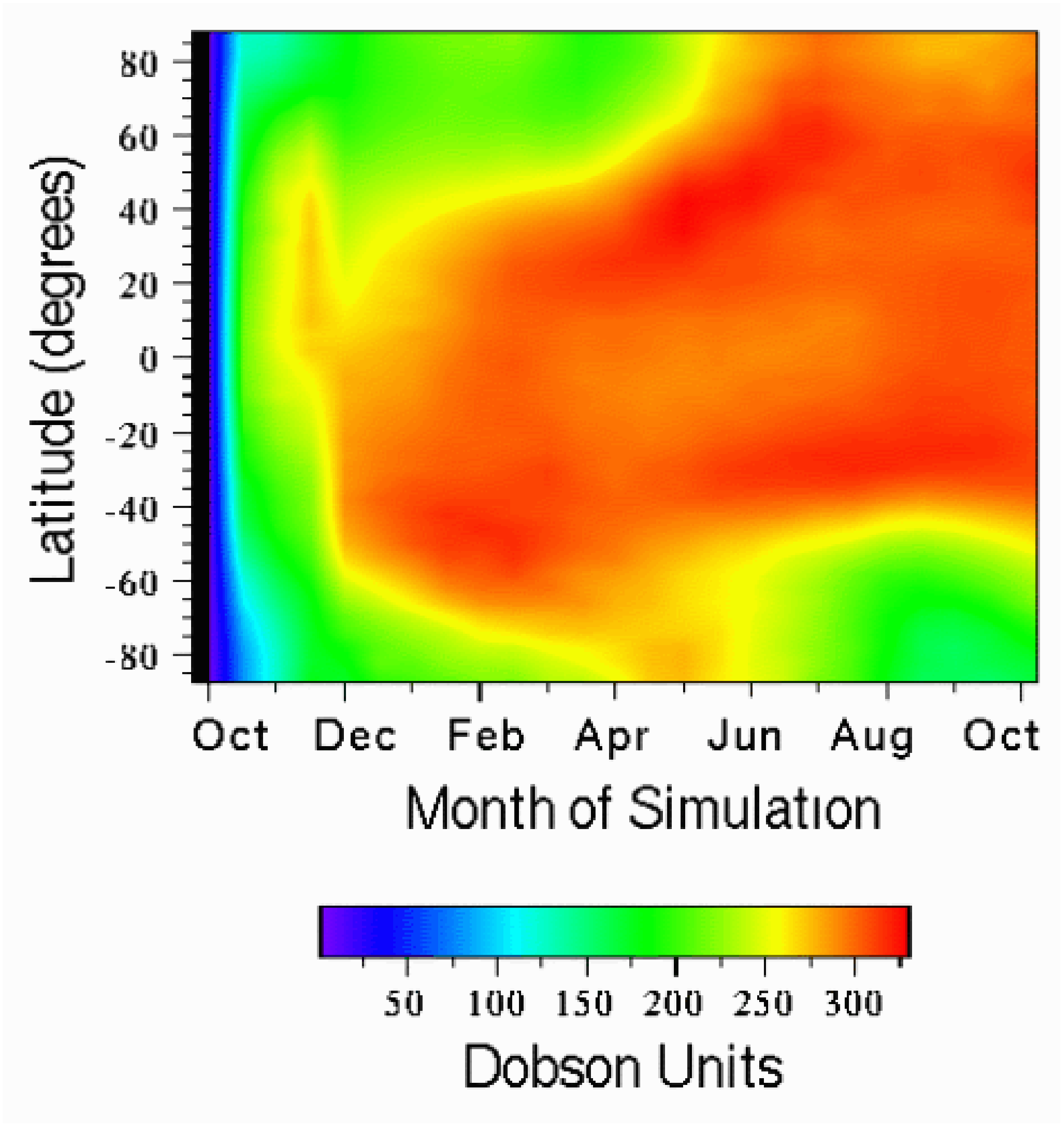


Figure 1 shows the change in ozone level over time in the atmosphere -- measured in terms of Dobson Units (DUs) -- at each latitude. The figure shows that initially no ozone was present at any latitude but that it regenerated at a different pace at each latitude. Seasonal variation of the sun's location, chemistry, and meteorology affected the location of peak ozone levels.

Graphic courtesy of Mark Jacobson

Back to the article

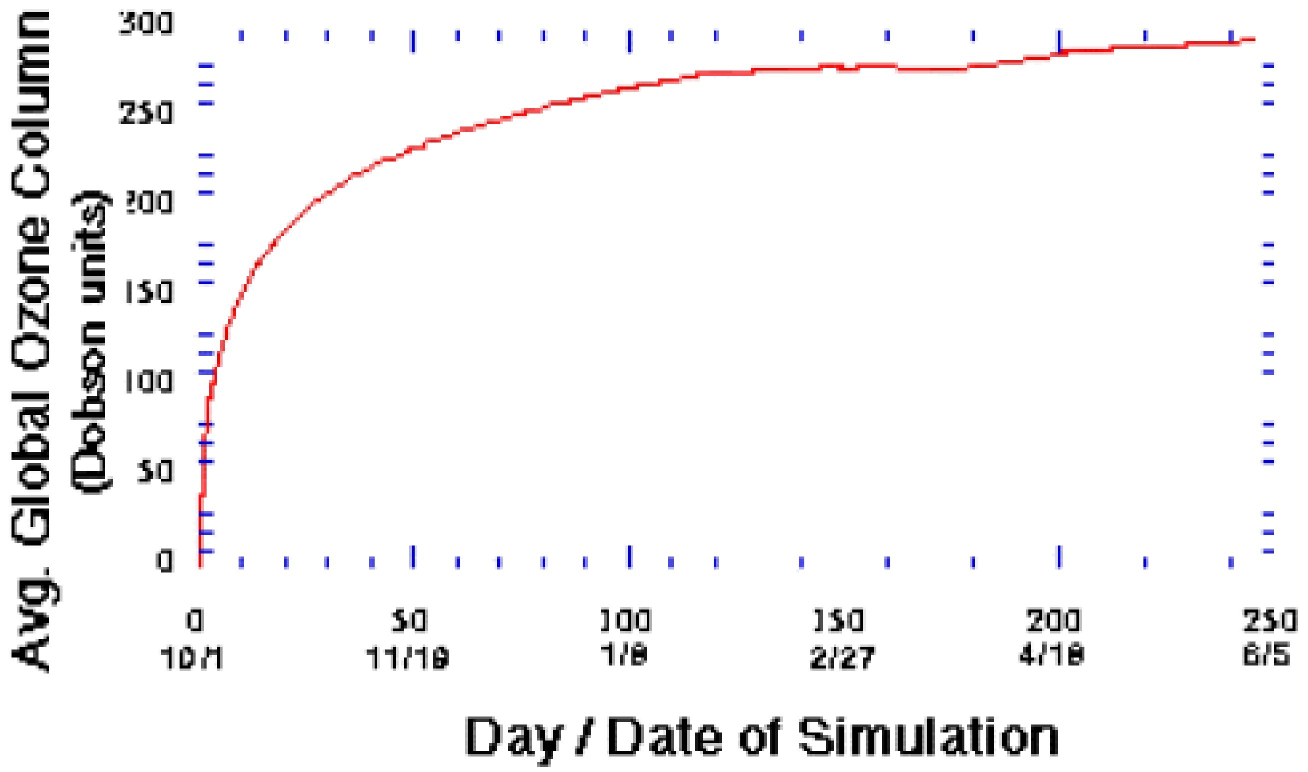


Figure 2 shows the pace at which the average global ozone layer regenerated in the model. The present day globally-averaged ozone level is 290-300 DUs. The figure shows that within 10 days after the start of the simulation the average level increased to about 150 DUs. The level reached 200 DUs after 27 days and increased at a slower rate thereafter.

Graphic courtesy of Mark Jacobson

Back to the article

Next Article Contents Main Menu NAS Home

Cray Fortran 90 Version 2 Offers Many New Options For Users

by Clayton J. Guest

NAS was one of the selected sites that beta tested version 2 of Cray Research Inc.'s Fortran 90 compiling system (f90). The larger CRAY C90 system at the NAS Facility, vonneumann, was used for the testing. Beta testing was completed at the end of 1995, and since then Fortran 90 version 2 has been the default f90 compiling system on vonneumann. The upgrade will be available on the other Cray systems (eagle, the Aeronautics Consolidated Supercomputing Facility C90, and Newton, the new CRAY J90 cluster) in the near future. Several major enhancements have been added to f90. These enhancements are summarized here to introduce them to users.

New Command Line Options

The new options to the f90 command line are: -*a*, -*D*, -*U*, -*F*, and -*R*.

-*a* is a *taskcommon* option that permits users to specify the task-common block storage allocation mechanism. This option converts all common blocks in the compilation to task-common blocks. Local variables named in SAVE are stored in taskcommon in that each processor (task) has a private copy of saved variables. This includes variables in data declaration or DATA statements. The COMMON compiler directive, described later in this article, will override this specification.

The options-*D* and -*U* define and undefine variables for conditional compilation. The form is: -D *var[, var]*... and -U *var[, var]*.... Each requires that at least one variable be named. If the input file name ends in .F or .F90, the GPP (generic preprocessor) is invoked and it recognizes the -*D* and -*U* options on the f90 command line. If GPP is not invoked, these options are recognized by the CF90 compiler.

The -*F* option enables macro expansion throughout the entire source file. Normally macro expansion occurs only on directive lines. This option is ignored, if the suffix of the source file is neither .F nor .F90.

The -*R runchk* option permits users to specify any group of run-time checks for their programs. *runchk* can be set to: compare the number and type of arguments passed to a procedure with the number expected; perform bounds checking of arrays; perform conformance checking of array operands in array expressions; check character substring bounds; or any combination of these checks. See the *man f90(1)* page for permutations of *runchk*.vision results upward. Before, it caused rounding of only integer division results.

New Arguments for enable and disable

Four additional arguments have been added to the repertoire of compiling options allowed with *-e* (enable) and *-d* (disable). By default, all four of these new arguments are disabled. A brief summary of the arguments *I*, *P*, *R*, and *Z* follows:

I - treats all variables as if an IMPLICIT NONE statement had been specified. Does not override any IMPLICIT or explicit type statements. This option forces "strong typing" and all variables must be typed.

P and *Z* both cause preprocessing on the source file and the generation of *file.i*. Upon completion of preprocessing, if *P* is specified the code is not compiled, but if *Z* is specified the code is compiled.

R - causes all subroutines and functions to compile as if they were declared RECURSIVE.

Modified Argument for enable and disable

The definition of the *u* argument for *-e* or *-d* has changed since the initial release of f90. The *u* argument now causes the CF90 compiler to round floating-point division results upward. (Before, it caused rounding of integer division results.) When disabled the results can be slightly less accurate. The result of the following example could produce an unexpected value:

```
X = 6.0
R = X / 3.0
...
I = R
```

Enhancements to Optimization

Several additional arguments are available to the *-O* option. These are mainly related to inlining subprograms, inlining modules, optimization messages, and Autotasking of innermost loops. These new optimization options are:

- inline*n* - where *n* can be set to: 0, 1, 2, or 3 for no, conservative, moderate, or aggressive inlining, respectively.
- *inlinefrom=source*: where *source* specifies a file or directory that contains procedures for inline code expansion into the Fortran source file to be compiled.
- *fastint, nofastint* - causes faster compare sequences, multiplication and division for INTEGER (KIND=8) and INTEGER*8 data objects.

- *modinline*, *nomodinline* - if *modinline* is specified, modules compile so that any procedures contained within them can be inlined if the module is used. The default, *nomodinline*, instructs the compiler not to inline procedures contained within the module.
- *msgs*, *nomsgs* - if *msgs* is specified, the CF90 compiler writes optimization messages to stderr. Default: *nomsgs*.
- *negmsgs*, *nonegmsgs* - if *negmsgs* is specified, the CF90 compiler writes negative optimization messages to stderr. Default: *nonegmsgs*.
- *taskinner*, *notaskinner* - *taskinner* specifies Autotasking for innermost loops and requests that a threshold test be performed prior to Autotasking. Default: *notaskinner*.
- *threshold*, *nothreshold* - if *threshold* is specified, the CF90 compiler tests to determine whether there is sufficient work in a nest of DO loops before Autotasking. Default: *threshold*.
- *vsearch*, *novsearch* - the default *vsearch* vectorizes search loops, where *novsearch* causes search loops not to vectorize.

Addition to the List Options

CF90 will now produce a 132-column-wide output list when the *-r* option contains the new argument *w*, in conjunction with an *s* or *x* list argument. By default, the listing is 80 columns.

Extensions to f90 File Names

The early version of f90 only understood files with a suffix of ".f90" or ".f". In addition, version 2 allows ".F90" and ".F" as suffixes. The uppercase "F" causes GPP to be invoked. As with the earlier version the ".f" implies fixed source format and ".f90" indicates free format.

Version 1.0 of CF90, when requested, produced a listing file with a ".l" suffix. Now a suffix of ".lst" is generated for the listing file.

The ".i" file (mentioned above in "New Arguments for enable and disable," above) does not seem to be documented yet. This file is only an informational file, which contains the Fortran source code that is compiled; that is, it contains inserted compiler directives, inlined code, and included code.

Fortran 90 Compiler Directives

Several additional compiler directives have been incorporated into this release of CF90.

The BOUNDS directive causes checks for most array references at both compile and run time to ensure that each subscript is within the array's declared size. Optionally names of selected arrays may be declared for checking. If none are selected then all arrays are checked when the BOUNDS directive is used. The NOBOUNDS directive (the default) is used to declare arrays not to be checked.

The new external naming directive NAME allows the user to specify a case-sensitive external name in the Fortran program. This directive can be useful when writing calls to a C subprogram. The name is specified on the NAME directive as, NAME (*fort_name="ex_name"*).

Several inlining directives have been implemented into f90 version 2. First is INLINE/NOINLINE. These cause automatic inlining analysis to be performed. One of these directives remains in effect until the opposite directive is encountered.

The MODINLINE directive allows all module procedures to be inlined. It causes the compiler to store more information in the module information table, which allows inlining of any module procedure declared within the scope of the directive. On the contrary, the NOMODINLINE, causes the compiler to produce a small module information table. Hence, any module procedure specified within the scope of the NOMODINLINE directive cannot be inlined. This holds true even if the module procedures are incorporated into a different compilation by a USE statement.

The directive INLINE ALWAYS *name* [,*name*] ..., always attempts to force the named procedure(s) to inline; where the directive INLINE NEVER *name* [,*name*] ..., suppresses inlining of the specified procedure(s). The directive INLINE NEVER overrides the directive INLINE ALWAYS.

Four new compiler directives have been implemented in the area of tasking and vectorization. PREFERTASK allows loops with large iteration counts to be considered as candidates for Autotasking. This directive shuts off threshold checking. It can be used if there is more than one loop in the nest that can be autotasked. PREFERTASK has no effect if Autotasking is not enabled.

PREFERVECTOR indicates that the loop following the directive should be vectorized. This directive can be used if there is more than one loop in the nest that could be vectorized.

UNROLL and NOUNROLL inform the compiler how to handle the unrolling of DO loops. NOUNROLL inhibits loop unrolling and overrides a *-o unroll2* command line specification. UNROLL uses an optional *n*, value to specify the total number of copies to be generated. If *n* is specified, the compiler does not attempt to determine the number of copies to generate based on the number of inner loops in the nested DO loops. Place this directive immediately before the DO statement of the loop to be unrolled.

Additional storage directives are now available. The compiler directive STACK causes the default storage allocation to be placed on the stack in the program unit that contains this directive.

When declaring common tasks the new *COMMON name* [,*name*]..compiler directive overrides the *-a taskcommon* command line specification for a given procedure. The information is not inherited from the parent scoping unit. All common blocks remain as single-copy common blocks when this directive appears in a procedure. The directive must contain at least one *name* and that *name* must also appear in a COMMON statement.

Additional Environment Variable

The new environment variable CRI_F90_OPTIONS specifies options to be appended to the f90 command line. The options are inserted following the options specified directly on the command line. File names cannot be included. This string is inserted at the right-hand side of the command line before the names of the binary and input files. For example if the following commands were issued:

```
setenv CRI_F90_OPTIONS -r1
f90 -c program.f90
```

it would cause the command *f90 -c program.f90 -rl* to be used, and the f90 compiling system would produce all output specified by *-r1*.

Reference Materials Available

For more details on these new (and older) command options and compiler directives see the *Cray Programming Environments* book of CF90 commands and directives, Reference Manual, SR-3901 2.0. For users with craydoc on their workstations, an early version of the *CF90 Commands and Directives Reference Manual* is available in the "pvp" collections. The man pages for f90 also contain useful information.

Clayton Guest is a member of the NAS Scientific Consulting team. He joined the NAS program in 1993.

Next Article | Contents
Main Menu | NAS Home

NAStore Upgrade Doubles Storage, Performance

by Jill Dunbar

An upgrade to NAStore, the NAS mass storage system, has increased storage capacity to nearly 200 terabytes of data and increased tape performance by more than a factor of two. The upgrade has "signficantly improved the quality of our (storage) environment," said John Parks, project coordinator. Parks, a member of the NAS mass storage group, is optimistic that performance should show even greater improvement once some of the kinks in the system are worked out.

The three-part upgrade, begun in February and completed in mid-June, was accomplished collaboratively with several NAS groups, which, in addition to mass storage, included those involved with high-speed processors, local area networks, and operations.

First came a "massive" system reconfiguration on the storage silos: 40 Storage Technology Corp. (STK) Timberline tape drives were installed in April, followed by 16 STK Redwood D3 drives. These additions brought NAStore into a SCSI environment -- which is much faster and more flexible than the older, slower technology, Parks said. He estimated that throughput could achieve I/O rates that are 2-3 times higher with this newer technology -- jumping from 2.5 megabytes per second (MB/sec) to as high as 6 MB/sec on the Timberlines and 10.5 MB/sec on the Redwood drives. However, he warns that end users aren't likely to see this increase, since no network changes have been made yet. (Network platforms are scheduled for replacement sometime in FY97.)

In the second phase, a new STK robotic silo (currently configured with eight Redwood drives) was installed offsite to serve as redundant storage for NAStore.

Finally, the total disk storage capacity was doubled to 4 TB on the two Convex systems (named chuck and scott) that comprise NAStore. For NAS users, this dramatically increases user data storage space on each system -- from .8 TB to 1.6 TB. A transparent benefit to users, as a result of the increased storage capacity, is a significant reduction in the number of manual tape mounts, which allows the NAS control room staff to spend more time helping users resolve questions and problems.

The upgrade has given NAS a "different base from which to leap into the next generation of storage," Parks said. The mass storage group is currently rethinking NAS's long-term approach to mass storage, in general. "What's important is that we take the time to develop a credible data storage strategy and determine what the next mass storage engine ought to be," he added.

That includes, in Parks' view, considering the many commercial off-the-shelf products offered today that weren't available 10 years ago, when NAStore was developed.

At present, the group is developing a set of requirements benchmarks to determine what the next-generation system should look like. Current plans tentatively call for a new system within two years.

Ultimately, Parks said, the NAS mass storage model will head in the entirely new directions. "That means new tools and technologies and a strong commitment to managing data in ways that we have historically ignored."

"We want to be able to provide not only an incredible archiving resource, but to be able to do the things that technology today demands and that the technology of tomorrow will demand, too," he said.

Next Article Contents Main Menu NAS Home

[Next Article] [Table of Contents] [NAS News Home Page] [NAS Home Page]

July -- August 1996, Vol. 2, No. 18

Executive Editor: Marisa Chancellor

Editor: Jill Dunbar

Senior Writer: Elisabeth Wechsler

Contributing Writers: Pieter Buning, Clayton Guest, Robert Hood, George Myers, Parkson Wong

Image Coordinator: Chris Gong

Other Contributors: Ying Chen, James Donald, Sam Fineberg, Mark Jacobson, John Lekashman, Louis Lopez, Terry Nelson, Bill Nitzberg, R.K. Owen, John Parks, Alan Poston, Kent Seamons, Cathy Schulbach, Dani Thompson, Marianne Winslett

Editorial Board: Marisa Chancellor, Nick Cardo, Jill Dunbar, Chris Gong, Mary Hultquist, David Lane, Chuck Niggley, Elisabeth Wechsler

NAS NEWS
Volume 9, Number 18 July – August 1996

I/O Performance of Multidimensional Arrays Tests Well on IBM SP2 With Panda

by Elisabeth Wechsler

A group of scientists based at the University of Illinois who are developing a software library to improve I/O performance on multidimensional arrays have found the IBM SP2's "clean environment" a boon for their research.

"An environment that produces consistent results is very important," said Marianne Winslett, principal investigator and faculty advisor to the Panda (Persistence AND Arrays) project team. She attributed this outcome in part to the SP2's homogeneously configured processors.

"Usually, researchers working on parallel I/O use 'best' times in reporting test results," Winslett continued. "But we've been able to use 'average numbers' because the test results were so consistent. This has allowed us to get to the heart of the problem and make improvements much more quickly."

Panda uses data management techniques to improve I/O interaction with the computing applications. The project is funded by NASA, ARPA (Advanced Research and Projects Agency), and the National Science Foundation. Since 1995, the project team has run tests on several parallel systems around the country, including other SP2s.

Getting Data On and Off Disk 'Fast'
"With these powerful parallel computations, we need I/O codes to access data on disk as fast as the underlying disk and operating system technology will allow," Winslett said, adding that currently available software libraries for I/O on parallel platforms "often fail to give high throughput."

In addition to high-performance array I/O operations, the project's goals include ease of use and application portability, according to Kent Seamons.

Continued on page 7

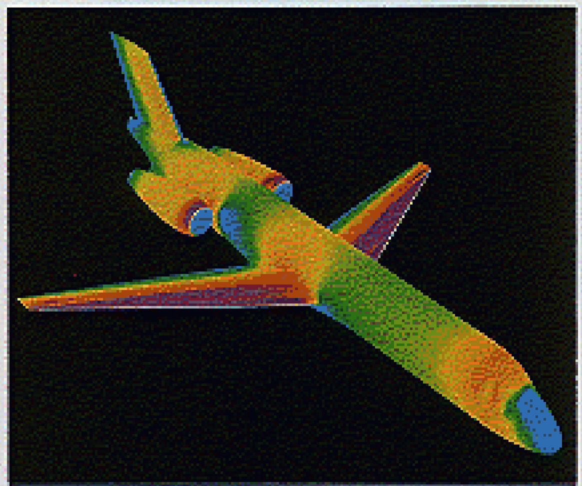


Figure 1. Surface pressure on a proposed Douglas MD-95, computed using OVERFLOW. The solution used 4 million grid points in 15 grids, and required 27 C90 hours and overnight turnaround on the Cray C90. (Graphic courtesy of John Vassberg, Mark DeHaan, and Doug Bowman, Douglas Aircraft Co., working with NAS.)

Flow Solver Developed With Extensive Industry Interaction

by Pieter Buning

A Navier-Stokes flow solver, OVERFLOW, has been developed by researchers in the Aeronautical Technologies Division, NASA Ames Research Center, to use overset (Chimera) grids to simulate the flow about complex aerodynamic shapes. It has been optimized to run efficiently on the NAS CRAY C90, and a production capability exists for the IBM SP2 and workstation clusters.

The OVERFLOW code is one representative of a collection of overset grid CFD software programs developed at Ames, including Collar Grid Tools (grid manipulation and surface grid generation), HYPGEN (volume grid generation), PEGSUS (overset grid hole cutting and boundary interpolation), DCF3D (an alternative to PEGSUS, especially for moving grids), and FOMOCO (force and moment calculation for overset grids). Typical customers use most of these tools for their CFD applications; the grid generation process represents by far the largest amount of "calendar time" of a project, though the flow solver consumes the bulk fraction of the computer time.

The Chimera grid approach was jointly developed in the 1980s by the Air Force Arnold Engineering Development Center for store separation problems and tunnel support interference evaluation—and by Ames for evaluating the Space Shuttle launch vehicle. Specifically, early developments of OVERFLOW and a variety of the tools were funded by the Space Shuttle Program Office and NASA Johnson Space Center (JSC).

Customers From Industry, NASA
Primary customers of the OVERFLOW flow solver and related software include McDonnell Douglas and Boeing, as well as the NASA-focused Programs for Advanced Subsonic Technology (AST) and High Speed Research (HSR). Interaction with the helicopter industry is handled through the joint Army/NASA Rotorcraft CFD group, co-located with the Ames Applied Computational Aerodynamics Branch.

Interaction with McDonnell Douglas is on two fronts. McDonnell Douglas Aerospace, Long Beach, CA, is responsible for the C-17 military transport, NASA AST and HSR interaction, and technology development. Douglas Aircraft Company (DAC), also in Long Beach, designs and manufactures commercial transports, including the MD-80 and -90 series and the MD-11 follow-on to the DC-10. OVERFLOW has been useful to McDonnell Douglas in NASA AST work, including propulsion/airframe integration (the effect of engine installation on aerodynamic performance), and high lift (landing and takeoff configurations). DAC's use for the code includes new projects such as the MD-95 (see figure above).

Continued on page 2