CRAY CHANNELS

SUMMER 1989 · A CRAY RESEARCH, INC., PUBLICATION

Parallel processing

CRAY CHANNELS

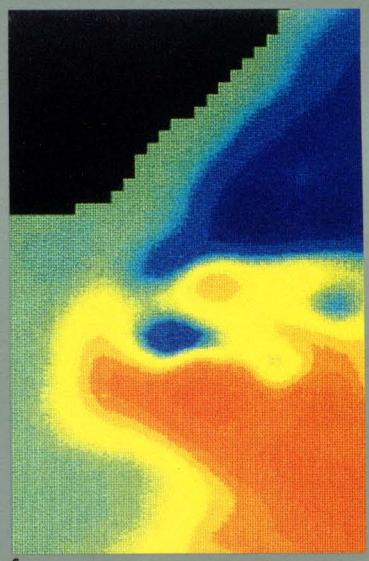
In this issue

Rowing a boat — painting a house — solving a complex problem. Each of these tasks can be accomplished more efficiently when the work is divided among several hands. This simple concept is at the heart of parallel processing, the simultaneous application of two or more of a computer's processors to a single problem. Cray Research's Autotasking capability allows users to exploit the advantages of parallel processing without expending the extra effort required by previous generations of parallel processing software. Applied effectively, parallel processing can increase total system throughput as well as performance on individual problems.

This issue of CRAY CHANNELS provides several perspectives on parallel processing using Cray systems. Experts from Cray Research, the National Center for Atmospheric Research, and Boeing Computer Services discuss parallel processing tactics, experiences, and performance benefits. This issue of CRAY CHANNELS also explores Cray Research's approach to computer networking and the use of Cray systems to study artificial neural networks: computing devices that apply principles of brain functioning to solve complex problems in pattern classification, speech recognition, and vision and signal processing. Our regular departments cover new system orders, networking and applications software releases, and other applications news.

As Cray Research continues to increase the number of processors in its systems, efficient parallel processing becomes increasingly important to users. To help users achieve their goals quickly and with the least amount of effort, Cray Research continues to develop new parallel processing capabilities, such as Autotasking. Parallel processing is an important strategy for solving large and complex problems, and its advantages are becoming clearer with each advance in its efficiency and ease of use.

Features



6

Departments

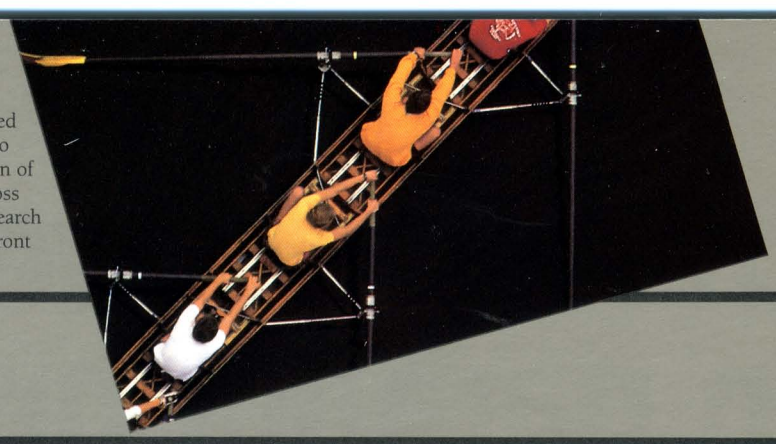
CRAY CHANNELS is a quarterly publication of Cray Research, Inc., intended for users of Cray computer systems and others interested in the company and its products. Please mail feature story ideas, news items, and "Gallery" submissions to CRAY CHANNELS at Cray Research, Inc., 1333 Northland Drive, Mendota Heights, Minnesota 55120.

Volume 11, Number 2

Editorial staff
Ken Jopp, editor
Elizabeth Knoll, associate editor

Design and production
Barbara Cahlander
Eric Hanson
James Morgan
Cynthia Rykken

On the cover: A rowing team represents a coordinated division of labor aimed at achieving the highest possible speed on water. Parallel processing aims to achieve the highest possible computational speed by a coordinated division of labor among multiple processors. By distributing individual problems across two or more processors, users can cut their run times significantly. Cray Research pioneered parallel processing on supercomputers and remains at the forefront of parallel processing hardware and software development.



Software developments in parallel processing

Mark Furtney, Cray Research, Inc.

Cray Research introduced its first multiprocessor supercomputer in 1982, the dual-CPU CRAY X-MP/2 system. To support the use of both CPUs by individual programs, a library of synchronization primitives was introduced, and it became the de facto industry standard. Parallel processing using these primitives is called *macrotasking*. In 1986, Cray Research introduced the UNICOS operating system, which is based on AT&T's UNIX System V. This system provided new, easy-to-use tools, such as background processes and pipes, for using multiple CPUs to accomplish complex tasks.

The macrotasking experiences of users and developers opened several unexplored avenues for parallel exploitation and led to the development of a collection of new techniques for parallel processing. This collection of techniques addressed some of macrotasking's weaknesses, generally was easy to use, and provided good performance in both batch and dedicated computing environments. The use of these techniques is called *microtasking*, and it provides substantial advantages over macrotasking, but still requires that programmers occasionally do some detailed data dependence analyses to use it safely. Unlike vectorization with Cray Fortran compilers, microtasking is not automatic.

To automate microtasking, Cray Research developed *Autotasking*, which grew from the excellent performance experiences with microtasking. Autotasking provides an automatic mechanism for exploiting parallelism without programmer intervention. However, Autotasking also provides a variety of techniques by which knowledgeable users can fine-tune a program and exploit levels of parallelism not visible to the Autotasking system.

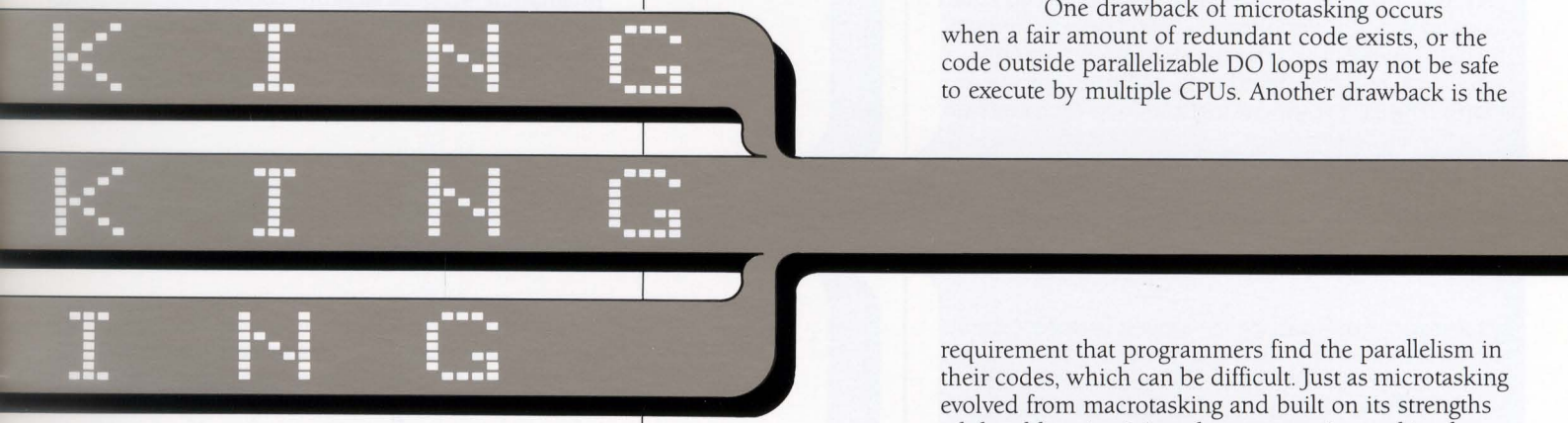
Macrotasking

Macrotasking is a technique whereby programs are modified with explicit calls to a special Fortran-callable library of synchronization routines (these routines are callable from C and other languages). This collection of synchronization routines, the *macrotasking library*, provides the primitives necessary for a single program to execute correctly on multiple CPUs. These library routines interact directly with the operating system to create extra *tasks*, and with the hardware to provide the necessary synchronization between concurrently executing tasks. The macrotasking library contains four sets of routines: tasks, which are used to create and manipulate tasks, and *locks*, *events*, and *barriers*, which are used for synchronization.

Macrotasking works best when the amount of work to be partitioned over multiple processors is large. When the work to be partitioned is not large compared to the synchronization time, excess synchronization time may become noticeable. When applying macrotasking to an existing code in which multiprocessing was not a design consideration, a significant amount of code restructuring may be necessary, which can introduce new errors. When the work is not easily partitionable into equal-sized tasks, load imbalance may occur, producing lower speedups than anticipated. Furthermore, macrotasked programs may display markedly different performance characteristics when comparing batch to dedicated executions. Because of these and other reasons, few programmers macrotasked existing large applications codes. Microtasking is the result of efforts to address some of macrotasking's weaknesses.

Microtasking

Microtasking is a technique for multiprocessing programs that is based on exploiting parallelism in DO loops. The primary design goals of microtasking are to provide good performance over a wide range of problem sizes, ease of use, and efficiency in both batch and dedicated environments. Microtasking employs a master/slave relationship among CPUs. When the master processor enters a region that can benefit from parallel execution, it alerts the slaves, which then may enter the computation.



Programmers who use microtasking determine where parallelism exists, then place comment directives in the text of their programs. A preprocessor reads the directives and translates them and their associated DO loops into a form acceptable to the compiler. Code is generated that allows the program to use extra CPUs *if they are available.* If idle CPU cycles are available on the system, microtasked codes can use them as accelerators for completing a particular loop. If idle cycles are not available, the microtasked code (executing in the master processor) does not slow down to summon them or to wait for them. Loop iterations are handed out to the next CPU ready for work (self-scheduling), resulting in excellent load balancing. The code that performs this protected "handing out" of iterations is extremely efficient, requiring about 40 clock periods on a CRAY X-MP system. This permits the profitable exploitation of very fine-grained parallelism. Microtasking also makes good use of other CRAY X-MP and CRAY Y-MP features, such as hardware deadlock detection, to provide very fast mechanisms for getting CPUs to join a "fray" (a code segment that may utilize multiple CPUs) and to remove CPUs from a potential busy-wait situation as, for example, when a CPU is withdrawn from a fray by the operating system, and the other CPUs must wait for a DO loop to be completed before continuing.

Alternative methods were developed to microtask code on CRAY-2 systems, which do not include the same rich set of parallel synchronization hardware. These methods have proven to be so powerful that they are being moved to the CRAY Y-MP microtasking design for comparison.

Although microtasking generates its computational efficiency by spreading DO-loop iterations over multiple CPUs, it is based on subroutine boundaries, not loop boundaries. It has been designed this way because subroutines provide a natural break in a code where the scope of variables can be manipulated easily into a form that permits correct parallel execution. (The scope refers to which subroutines and tasks can "see" particular instances of particular variables.) This implies that some portions of microtasked subroutines are executed redundantly by whichever CPUs show up. In many instances this redundant computation merely sets up a local context in which the parallel loop may execute (without slowing down the master processor by forcing it to broadcast local context).

One drawback of microtasking occurs when a fair amount of redundant code exists, or the code outside parallelizable DO loops may not be safe to execute by multiple CPUs. Another drawback is the requirement that programmers find the parallelism in their codes, which can be difficult. Just as microtasking evolved from macrotasking and built on its strengths while addressing its weaknesses, so Autotasking has evolved from microtasking. Autotasking retains the many advantages of microtasking and overcomes some of its shortcomings.

Autotasking

Autotasking is a technique whereby the compiling system detects opportunities for parallel exploitation, and generates code to execute these parallel regions on multiple CPUs. Autotasking retains the advantages of microtasking, including self-scheduling for good load-balancing, very low synchronization overhead, use of idle CPU cycles when available, excellent performance in both batch and dedicated environments, and unmodified source code, while adding several new advantages. Autotasking can be completely automatic — the compiling system can do all the analysis and generate parallel code — or it can support the programmer in exploiting potentially higher levels of parallelism. Autotasking works on DO-loop boundaries, but easily is expandable to "parallel regions" and to subroutine boundaries.

Autotasking inside the compiling system can be thought of as a three-phase operation: *dependence analysis, translation,* and *code generation.* Programmers may pass directives to any of the three phases. For example, a programmer may know that a certain large program segment accounts for only a small percentage of total run time, and so may choose to disable the extensive dependence analysis done in this region of the code because no payoff will result (and to save compilation time). Or, a programmer may know that a DO loop that contains an external call may be safe to execute in parallel and wish to alert the compiling system to that fact (the dependence analyzer does not look beyond subroutine boundaries).

Functions of the dependence analyzer

The dependence analyzer performs a wide variety of program optimizations. It looks for parallel constructs and may perform some source code transformations to produce faster-executing code. Payoffs from dependence analysis come in four major areas: enhanced vectorization, recognition and generation of parallel constructs (concurrentization), automatic in-line expansion, and special code sequence recognition.

Enhanced vectorization

The dependence analysis phase recognizes vectorization opportunities and uses various techniques to try to produce vectorizable code. These techniques include statement reordering, ambiguous subscript resolution, reference reordering, splitting calls out of loops, loop nest restructuring, and loop exchanges (to get a stride of one and/or the longest vector length on the innermost loops). In a recent vectorization study of 100 loops, the vectorized loop count went from 51 to 72 when using the new compiling system, a substantial improvement over the vectorization achieved with the previous compiling system release.[1]

Concurrentization

The dependence analysis phase recognizes parallelization opportunities and inserts directives to the next phase (translation) that tell it how to exploit this inherent parallelism. Again, the dependence analyzer may do some code transformations to produce parallel opportunities. In general, it will attempt to concurrentize on the outermost possible loop.

Automatic in-line expansion

Automatic in-lining of subroutines by the dependence analyzer also has a big payoff. In-line expansion not only removes the overhead of the call, but also then possibly can concurrentize on a more outer loop (from the original code before in-line expansion). In many cases this is very profitable.

Specific code sequence recognition

The dependence analyzer also recognizes some special code sequences, including matrix multiply, first and second order linear recurrences, dot product, and search for maximum or minimum. It then generates calls directly to optimized library routines that can perform these functions in parallel.

Functions of the translator

The primary function of the translation phase is to rewrite the Fortran-code-with-directives into pure Fortran for use by the code generation phase. Directives (whether written by the dependence analyzer phase or the human programmer) are expanded into a series of special function calls and compiler intrinsics that together implement the requested parallel processing functionality. A primary consideration in this rewriting exercise is enforcement of the scoping requirements detailed in the directives. Every variable in each parallel segment of code has a "scope" which is either *shared* or *private*, as declared on the directive. Only one copy exists of each shared variable, and it is available to all processors that contribute to the computation. Potentially many copies exist of private variables, one per contributing processor.

Unique features of Autotasking

The three-phase compiling system gives programmers a great deal of freedom in selecting the forms of parallel processing most efficient for individual types of computation. Users also find great range in the forms of directives that can direct the dependence analyzer and the translator, and they are encouraged to combine their own knowledge of the problem domain with the dependence analyzer's output to create faster-running programs. The concept of a parallel region allows the computational overhead of processor start up to be minimized and amortized over multiple exploitable sections of code. Parallel regions also allow the efficient parallel exploitation of many forms of reduction computations. The *savelast* parameter on the DO ALL construct allows parallel loops that must carry scalar (or array) elements from the last iteration out of the loop body to be executed correctly and efficiently in parallel. The in-line expansion feature of the dependence analyzer is performed before parallelism analysis is performed, leading to many cases of outer parallel loops surrounding inner (in-lined) loops. The SOFT EXIT construct allows loops that contain a jump to outside their range to be safely processed in parallel. The CONTINUE construct permits an important cross-subroutine optimization to occur under user direction. The introduction of the five forms of parallel loop iteration partitioning (*single, chunksize, numchunks, guided,* and *vector*) adds a new dimension to tuning particular DO loops. Guided and vector partitioning methods are particularly useful for a variety of programs.

Autotasking results

Results of autotasking applications reveal the advantages of this parallel processing option. As expected, the amount of exploitable parallelism inherent in a code sets a limit on the performance improvements that can be obtained. Some codes contain little or no parallelism, some contain a modest amount, and some are almost entirely parallel, and many codes of each type exist. Regardless of the amount of parallelism in a given program, users can benefit almost always from the messages generated by the dependence analyzer that describe why vectorization or concurrentization has been inhibited. The program in the first example is a large magnetohydrodynamics code. When working

out the maximum theoretical speedups possible for a code with this level of parallelism (just over 89 percent), the implementation of Autotasking produces speedups very close to the maximum possible (Table 1). This is an example of the very low overhead cost associated with synchronization in Autotasking.

The second example is representative of many programs that have been microtasked and then autotasked (the microtasking directives were removed for the autotasked run) (Table 2). The programmer took about two weeks to do the microtasking, whereas the Autotasking system took about 210 milliseconds. Nevertheless, it is very difficult for an automatic system to compete with human programmers. As in many similar cases, the human programmer microtasked a series of DO loops that contained external calls. The programmer checked the called routines and found it was safe to execute them in parallel. The Autotasking system does not look beyond subroutine boundaries, so it was forced to make the "safe" judgment. It judged that the external call was not safe to execute in parallel, so it marked the loops as nonparallelizable because of the external calls. Programmers confronted with this situation can utilize the CFPP$ CNCALL directive to the dependence analyzer to indicate that an external call in the next loop should be considered safe for parallel execution.

The third example also compares microtasking and Autotasking (Table 3). However, the situation in this example occurs much less frequently than that in the second example. As in the second example, the time required for microtasking was about two weeks, versus 190 milliseconds for Autotasking. In this case however, Autotasking found several areas for parallel execution not found by the programmer. In particular, the Autotasking system found and exploited some concurrentizable reductions (microtasking has no mechanism for this construct).

CPUs	Run time (secs)
1	424.8
2	239.3
3	176.7
4	146.0

Table 1. Run times for a magnetohydrodynamics code on a CRAY X-MP/48 system, showing performance improvements obtained with Autotasking.

CPUs	Microtasking time (secs)	Autotasking time (secs)
1	258.311	250.219
2	131.582	157.210
3	92.721	124.939
4	73.638	109.824
5	55.516	97.847
6	53.712	93.981
7	51.927	89.466
8	50.861	87.003

Table 2. Microtasking and Autotasking run times obtained on a CRAY Y-MP/832 system. Although the microtasked code performed better, the entire microtasking process took about two weeks for a human programmer, whereas the Autotasking system took only about 210 milliseconds to generate parallel code.

CPUs	Microtasking time (secs)	Autotasking time (secs)
1	166.122	171.724
2	86.113	87.651
3	59.753	59.601
4	46.819	45.603
5	40.364	37.440
6	34.145	32.026
7	33.999	28.268
8	28.863	25.628

Table 3. Microtasking and Autotasking run times obtained on a CRAY Y-MP/832 system. In this case, the Autotasking system found several areas for parallel execution not found by the programmer.

Building on the best

Increasing numbers of high-performance CPUs that can timeshare a large workload as well as cooperate on individual user programs are a fundamental part of Cray Research's hardware plans. The software to support the use of multiple processors on individual user programs has evolved through three generations: macrotasking, microtasking, and Autotasking. These three generations of parallel processing software form an integrated parallel processing package and can be used in any combination in a single program. Each generation provides some special services not available from the others, and Cray Research will continue to support all three. The company also will continue to develop more powerful parallel processing tools that will make the use of parallel processing easier and more efficient on existing hardware systems as well as on future, more highly parallel, systems. ▬

About the author

Mark Furtney is group leader for Cray Research's multiprocessing software group. He also has responsibility for the CRAY X-MP, CRAY Y-MP, and CRAY-2 libraries. Furtney earned a B.S. degree in mechanical engineering from Clarkson College of Technology in New York in 1968, an M.S. degree in nuclear engineering from the Massachusetts Institute of Technology in 1970, and a Ph. D. degree in computer science from the University of Virginia in 1983. He joined Cray Research in 1982.

Reference

1. Callahan D., Dongarra, J., and Levine, D., "Vectorizing Compilers: A Test Suite and Results," Proceedings, IEEE Supercomputing '88.

Microtasking climate models at NCAR

Robert M. Chervin
National Center For Atmospheric Research
Boulder, Colorado

Climate system science involves many areas of traditional science, including astronomy, meteorology, oceanography, hydrology, chemistry, and physics. Computer models of global climate systems integrate many of the forces that determine the state of the environment and govern its evolution. The application of numerical models to socially relevant environmental problems, such as the greenhouse effect, acid rain, and ozone depletion, can test the capabilities of any supercomputer system.

Climate modeling at the National Center for Atmospheric Research (NCAR) has advanced from one-month atmospheric simulations by numerical weather prediction models to multidecadal simulations of coupled ocean-atmosphere models that address the impact of enhanced carbon dioxide (the greenhouse effect). The investigation of longer-time-scale phenomena requires the incorporation of more complex interactions among the various components of the climate system and lengthier model integrations. The result is a need for ever-increasing amounts of computer power. When computer resources are limited, available resources must be used efficiently.

For instance, NCAR has been adapting global atmosphere and ocean models for multiprocessor execution. As a result, a variety of computationally intensive climate research endeavors are feasible on supercomputers such as NCAR's CRAY X-MP/48 system. Additionally, NCAR is preparing to meet even greater climate modeling challenges with supercomputers such as the CRAY Y-MP system and future systems that are expected to have larger main memories, different memory hierarchies, and additional processors.

The following sections describe NCAR's model conversion strategies, use of Cray diagnostic software and microtasking directives, degrees of parallelism obtained, and model performance. Also summarized are some new results obtained from these multitasked atmosphere and ocean models.

The atmosphere model

We multitasked the annual cycle version of the NCAR Community Climate Model (CCM),[1] which evolved from the original CCM.[2,3] The original model was based on the spectral transform atmospheric general circulation model developed at the Australian Numerical Meteorology Research Centre.[4] The spectral transform technique is used for horizontal discretization with rhomboidal truncation of the spherical harmonic basis functions at wave number 15. Vertical derivatives are represented by finite differences in the nine sigma, or normalized-pressure, layers that define the vertical structure. At this resolution, the model and all the requisite data are memory resident, even on a CRAY-1A computer system.

We compute the model's "physics" (such as convection, clouds, radiative transfer, and ground-surface energy balance) and "dynamics" (such as horizontal and vertical advection and other nonlinear products) for vertical slabs composed of longitude/sigma sections in physical space. The prognostic variables of temperature, humidity, divergence, vorticity, and surface pressure are advanced in time by means of a semi-implicit procedure for the complex wave amplitudes. We convert grid-point space data to wave-number space data and vice versa with a sequence of Legendre and Fourier transforms.

Multitasking conversion strategy

The conversion plan for the model was based on the principle of data-space partitioning to effect concurrency while maintaining the level of vectorization existing in the original code. To apply this principle successfully, we identified disjoint subsets of the model's total data space for which independent, parallel calculations would be possible. The model's numerical algorithms and prescriptions of physical processes dictated appropriate partitioning strategies.

A natural double-data-space partitioning evolved for parallel computing. In physical space, a vertical slab for a latitude line was selected as the basic computational element for an individual processor. The second partitioning arrangement involved dividing the wave number space (in the shape of a rhomboid) into equal-area zonal wave number bands

to achieve concurrency in the time stepping and in Legendre transformations, which convert the Fourier coefficient into wave amplitudes.

Our next step was implementing the above design. At this point, the "venerable" nature of the original code, with its approximately 100 subroutines, 100 common blocks, and 10,000 lines of Fortran code composed by a multitude of authors, became quite apparent. Communal data management was the most time-consuming and tedious part of the process. We used Cray Research's FTREF utility for global common block cross-referencing to make this chore more bearable.

The first implementation featured only high-level parallelism (that is, at the subroutine level only) and employed explicit Cray subroutine calls to start (TSKSTART) and synchronize (TSKWAIT) the individual tasks. This implementation resulted in 3.2 percent sequential code, in terms of CPU time. A second implementation built upon the first and added lower-level parallelism (that is, at the outer loop level) in several routines formerly in the sequential part of the code. Microtasking directives, with their inherently lower overhead, were used in this case at the subroutine and loop levels. This implementation reduced the sequential portion of the code to just 0.5 percent.[5]

Computer performance

The original version of the model required 11 CPU hours on a CRAY-1A computer system for a one-year simulation. Thus, multidecadal integrations required hundreds of hours, and few such experiments were attempted in that era. During the course of the conversion effort, some further internal optimizations were achieved in addition to code restructuring for multitasking. Cray Research's SPY and PERFTRACE utilities aided in the optimization and conversion efforts. The result was a microtasked model that could simulate one year of atmospheric activity in 1.4 wall-clock hours on a dedicated CRAY X-MP/48 computer system. Now, multidecadal integrations are possible in tens rather than hundreds of hours. Furthermore, a speedup of 3.7 was achieved with the dedicated CRAY X-MP/48 computer system, compared to its dedicated uniprocessor performance. The difference between this actual speedup and the theoretical value (3.94, from an application of Amdahl's Law for 0.5 percent sequential code) can be attributed to operating system interference, task imbalances, memory bank conflicts, and the slight additional overhead incurred with microtasking.

New scientific results

Shortly after the CRAY X-MP/48 computer system arrived at NCAR in the fall of 1986, we performed a pair of extended-term integrations of this multitasked atmospheric model to assess the impact of interannually varying global ocean surface temperatures on the interannual variability of time-averaged atmospheric states. The ocean temperature data, which provided lower boundary forcing for the model, came from the Comprehensive Ocean Atmosphere Data Set (COADS) for the interval January 1950 through December 1979. Seven moderate-to-strong El Niño events, unusually warm surface waters in the eastern tropical Pacific, are contained within this interval. One model integration featured forcing by the 360-month evolution of the

The model recently shattered the GFLOP barrier on the CRAY Y-MP/832 system at Cray Research's computer center in Mendota Heights, Minnesota, as it performed at a sustained rate of 1.1 GFLOPS on a lightly loaded, but nondedicated system.

COADS ocean surface temperatures. The second was characterized by the monthly mean COADS climatological temperatures repeated through 30 identical annual cycles. Thus, the two experiments had the same mean forcing (in terms of ocean surface temperatures), but the first provided interannual variability in the forcing through realistic global patterns of ocean surface temperature anomalies.

Objective univariate and multivariate statistical tests were applied to assess the effect of variable forcing on the interannual variability of monthly, seasonal, and annual means for a wide variety of atmospheric variables. In general, enhanced atmospheric variability resulted in the tropics from the variable boundary forcing but not in the extratropics. Statistical analyses of composites of atmospheric mean states for El Niño and non-El Niño events (from within the evolving COADS experiment) also were applied to determine the extent to which the atmosphere typically exists in different states for these different classes of boundary conditions. Clear and statistically significant differences were found in the tropical atmosphere but not in the higher latitudes. The latter results agreed with similar analyses performed with observed data.

The ocean model

Next, we multitasked the Semtner[6] version of the Bryan and Cox ocean model developed at the Geophysical Fluid Dynamics Laboratory and first described by Bryan.[7] The model is formulated in spherical coordinates and allows irregular domains with realistic coastlines and bottom topography. We used a second-order accurate, energetically consistent finite differencing scheme in space. Leapfrog differencing is used for the time integration, with an occasional Euler backwards-cycle time step to prevent possible splitting of the model solutions. The prognostic variables are potential temperature, salinity, zonal velocity, meridional velocity, and the horizontal volume transport stream function. Model resolution in both horizontal and vertical directions is trivial to vary. The only resolution constraints are available memory and computer time.

Multitasking conversion strategy

The experience gained by converting the atmospheric model facilitated this effort considerably. We selected slab partitioning in terms of independent longitude/depth sections for concurrent execution, with vectorization maintained in the longitudinal direction.

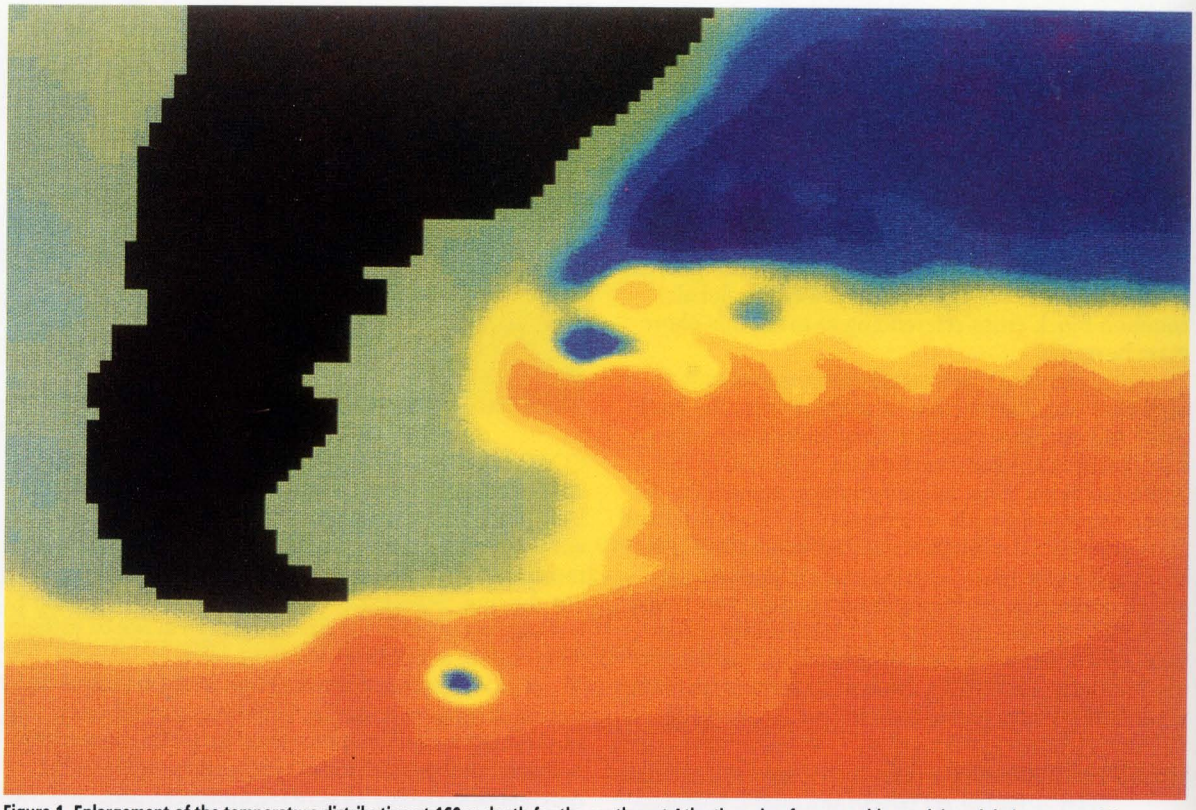


Figure 1. Enlargement of the temperature distribution at 160 m depth for the southwest Atlantic region from an eddy-resolving global ocean model. Warmest temperatures are shown in red; coldest temperatures are shown in blue.

We used stack allocation of memory to ensure independent data sets for each processor. Microtasking directives were implemented at both the subroutine and loop levels. Asynchronous, queued input/output (AQIO) routines were employed to effect efficient data transfer between main memory and the SSD solid-state storage device of the CRAY X-MP/48 computer system. (Out-of-memory considerations were required by the relatively small spatial scales needed to resolve the energetic scales of oceanic activity, or the "eddies.") Finally, to reduce performance degradation due to task imbalances, we imposed a minimum of synchronization points.

Computer performance

As with the atmospheric model, we used the full complement of Cray performance and diagnostic tools to convert and optimize the ocean model. The first application of the Cray hardware performance monitor to the ocean model revealed a rather unimpressive performance of 30 MFLOPS on a single processor of a CRAY X-MP/48 computer system. After additional vectorization, restructuring, algorithm modification, application of the Cray Fortran compiler CFT77, and parallelization, the model performed at a *sustained* rate of 450 MFLOPS during dedicated four-processor execution. We also obtained a multiprocessor speedup of 3.74 for a 0.36 percent sequential fraction.

The model recently shattered the GFLOP barrier on the CRAY Y-MP/832 system at Cray Research's computer center in Mendota Heights, Minnesota, as it performed at a sustained rate of 1.1 GFLOPS on a lightly loaded, but nondedicated system. On the CRAY Y-MP system we achieved a speedup of 7.1, in terms of wall-clock time, and average I/O performance of 200 Mbytes/second. These performance figures should improve when the model is run in a dedicated environment on a 6-nsec clock instead of a 6.3-nsec clock.

New scientific results

We achieved a simulation of the global ocean with resolved eddies using a version of the multitasked ocean model with a horizontal resolution of one-half degree in both latitude and longitude and 20 levels in the vertical.[8] Scale-selective biharmonic forms of frictional and diffusive parameterizations were used to allow the spontaneous generation of mesoscale eddies with realistic energy levels.

Figures 1 and 2 show an example of the simulated eddy activity for the 160-meter depth temperature and volume transport stream function, respectively, for a particular instant of time in the southwest Atlantic Ocean. The stream-function chart is color coded in contrast to the temperature. Notice that anticyclone or counter-clockwise (warm core) disturbances, which appear red in the temperature plot, show up blue in the stream-function plot. This color reversal is advantageous because some structures that are hard to see as blue shades in one field appear more conspicuous as red shades in the opposite field, and vice versa. A confluence of the Brazil and Falkland Currents emerges in the figures for the simulated southwest Atlantic, which leads to the separation of each current, with possible distinct axes for ring pair generation associated with the two currents. At some longitudes, four distinct cores of circulation in the stream-function chart may be found. The location of the eddy activity of this confluence region is consistent with data obtained from satellite observations.

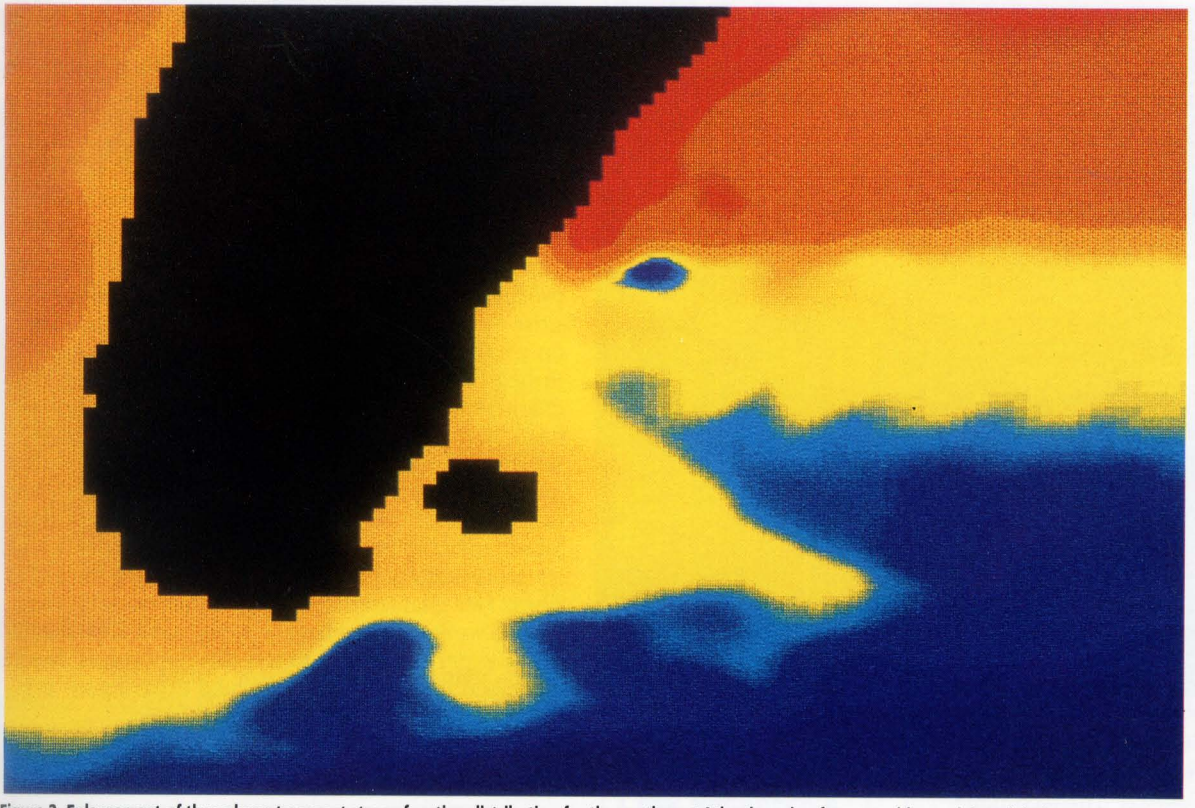


Figure 2. Enlargement of the volume transport streamfunction distribution for the southwest Atlantic region from an eddy-resolving global ocean model. Clockwise circulations are shown in orange-red; counterclockwise circulations are shown in blue.

Also evident in these figures is another conspicuous region of eddy activity slightly to the east of the Drake Passage. Ring pairs are produced that wrap clockwise around each other and act to transport heat poleward. Such behavior is supported by the limited observations available for this region.

Discussion

As computer technology changes, climate modelers must adapt techniques to take advantage of new computer capabilities to advance the field of climate science. The modeling community will be challenged to achieve higher model performance by exploiting the parallelism inherent in the climate system and in multiprocessor supercomputers such as Cray systems. However, meeting the challenge requires the consideration of a new set of "software engineering" issues that depend on machine architecture, compilers and operating systems, as well as on the prescription of physical processes and numerical techniques within the model.

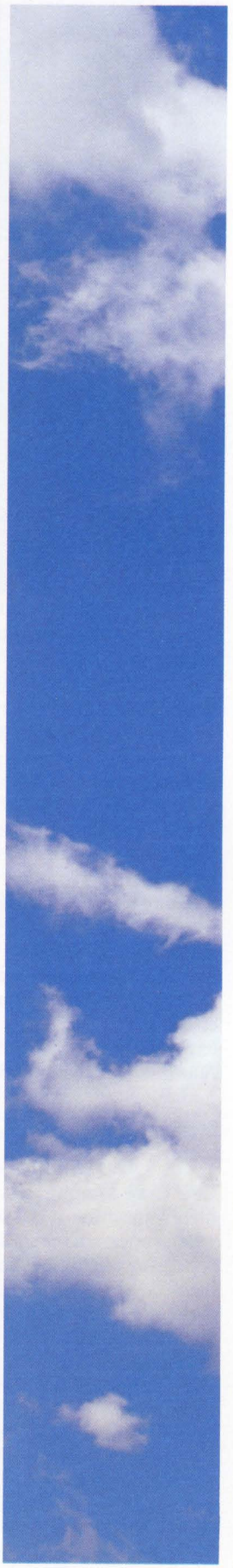
The decision to develop multitasked atmosphere and ocean models at NCAR was motivated by the desire to perform more research in less time. The new scientific results obtained with NCAR's CRAY X-MP computer system clearly demonstrate the value of this effort. ▬

Acknowledgments

The National Center for Atmospheric Research is sponsored by the National Science Foundation. This article was adapted from the author's article "Do Global: A Climate Modeling Imperative as well as a Microtasking Directive," from the proceedings of Cray Research's Fourth International Symposium, October 1988. The ocean model was developed in collaboration with Albert J. Semtner of the Naval Postgraduate School.

References

1. Chervin, R. M., "Interannual Variability and Seasonal Climate Predictability," *Journal of Atmospheric Science,* Vol. 43, 1986, pp. 233-251.
2. Pitcher, E. J., R. C. Malone, V. Ramanathan, M. L. Blackmon, K. Puri, and W. Bourke, "January and July simulations with a Spectral General Circulation Model," *Journal of Atmospheric Science,* Vol. 40, 1983, pp. 580-604.
3. Ramanathan, V., E. J. Pitcher, R. C. Malone, and M. L. Blackmon, "The Response of a Spectral General Circulation Model to Improvements in Radiative Processes," *Journal of Atmospheric Science,* Vol. 40, 1983, pp. 605-630.
4. Bourke, W., B. McAvaney, K. Puri, and R. Thurling, "Global Modeling of Atmospheric Flow by Spectral Methods," *Methods in Computational Physics, Vol. 17, General Circulation Models of the Atmosphere,* J. Chang, Ed., Academic Press, 1977, pp. 267- 324.
5. Chervin, R. M., "On the Relationship Between Computer Technology and Climate Modeling," *Physically-Based Modeling and Simulation of Climate and Climate Change,* M. E. Schlesinger, Ed., Reidel, Dordrecht, 1988, pp. 1053-1068.
6. Semtner, A. J., "An Oceanic General Circulation Model with Bottom Topography," Technical Report 9, UCLA Department of Meteorology, 1974.
7. Bryan, K., "A Numerical Model for the Study of the World Ocean," *Journal of Computational Physics,* Vol. 4, pp. 347-376.
8. Semtner, A. J., and R. M. Chervin, "A Simulation of the Global Ocean Circulation with Resolved Eddies," *Journal of Geophysical Research,* Vol. 93, 1988, pp. 15502-15522 and 15767-15775.



The impact of microtasked applications in a multiprogramming environment

Michael Bieterman, Boeing Computer Services
Seattle, Washington

Microtasking is a technique for distributing applications across two or more processors on multiprocessor Cray computer systems. The technique most often is used to exploit parallelism at the loop level and can generate significant improvements in performance. Most comparisons of microtasked and unitasked jobs have come from experiments conducted in dedicated single-job computing environments. This article examines microtasking performance in a multiprogramming (batch) environment, in which jobs in a batch stream compete for processors and memory. Results of three experiments are presented; these results show the positive effect of microtasked jobs on batch-stream throughput time. The experiments were conducted using Cray Research's COS operating system and microtasking. Even greater throughput improvements could be expected from similar experiments using Cray Research's UNICOS operating system and the Autotasking capability of the CFT77 Fortran compiler.

Motivation

When applications are run on multiple-CPU Cray computers in multiprogramming environments, machine cycles are wasted from time to time when processors are idle. One approach for reducing processor idle time is to include multitasked jobs in batch streams, which increases the number of tasks that can be scheduled for execution. On today's two-, four-, and eight-processor Cray systems, this approach offers a significant opportunity to capture otherwise lost clock cycles. As the number of processors increases, multitasking in batch has an even greater impact on system throughput because keeping all processors on a system busy becomes increasingly difficult.

Any of several situations can result in idle processors when a system runs in batch mode:

- ☐ System load is temporarily low
- ☐ Less than *N* jobs simultaneously fit in memory on an *N*-processor system; that is, memory is "oversubscribed"
- ☐ Resident jobs use input/output (I/O), but not CPU, for some time
- ☐ I/O occurs when rolling jobs in and out of memory
- ☐ Jobs wait for a critical I/O resource, such as SSD storage

This article examines the ability of one implementation of multitasking to use otherwise idle processors in batch when some of these situations exist. Specifically, the performance of microtasking[1] on a four-processor CRAY X-MP system is evaluated through the results of three experiments. An important characteristic of microtasking is its relatively low overhead, which is due to its direct use of the hardware semaphore registers on the CRAY X-MP system. Another important characteristic is that microtasked jobs *request*, rather than demand, available processors. Microtasked jobs in memory get more than one processor only when no processorless job in memory is waiting for a CPU. These two characteristics suggest that microtasking should work well in a batch environment. The purpose of the experiments was to test this hypothesis. Results described here and for one related experiment[2] evidently are the first of their kind reported.

Experiments

The three experiments were run at the Pittsburgh Supercomputing Center (PSC). At the time, the configuration at the PSC included a CRAY X-MP/48 computer system running a 9.5 nsec clock, a Cray Research I/O Subsystem with four I/O Processors, Cray Research DD-49 disks, and a 128-million-word SSD solid-state storage device. In all experiments, version 1.15 BF2 of the COS operating system and release 1.15 of the Cray Fortran compiler CFT were used.

Each experiment consisted of running two or three separate batch job streams. A baseline job mix in each experiment comprised all unitasked jobs, to be executed sequentially on one processor. The other job mixes in each experiment included some microtasked jobs that requested the maximum number of available processors.

Each job mix consisted of twelve related memory-intensive jobs. Each job fetched a compiled Fortran program from disk, performed a segmented load, and executed the resulting absolute binary. Microtasked jobs also accessed appropriate libraries and the Cray utility PERFMON.[3] The PERFMON,ON=1 option enables users to obtain wait-on-semaphore times for processors assigned to a microtasked job. These provide rough estimates of idle times due to synchronization overheads and time spent outside of the parallel parts of a code. The experimental batch streams were cold-started and static. That is, all other jobs on the system were suspended; a 7.5-million-word set-up job was run to avoid memory fragmentation and to ensure that relevant files were on disk; an experimental job mix was submitted; and no other jobs ran until all jobs in the experimental batch stream completed. The job extent was large enough to include all twelve jobs in a mix. All jobs had the same priority for scheduling into memory. Initially, this scheduling generally is performed on a first-in-first-out basis.

The key values listed below were used with the COS job scheduler in all runs; these values were taken "off the shelf" and were not tuned for the experiments:

- *In-memory thrash lock* (2 seconds): the minimum time a job entering memory stays there
- *On-disk thrash lock* (12 seconds): the minimum time a rolled job stays on disk
- *Time slice* (.44 seconds): the execution period for each unitasked job in memory and for each of the four logical tasks in a microtasked job in memory

Performance in the experiments was measured simply by comparing life lines of jobs in the unitasked and (partially) microtasked mixes. Two somewhat arbitrary times compared here are those needed for all and for half of the jobs in a batch mix to complete. Fractional batch stream throughput times are relevant because, in practice, additional jobs would enter a batch stream before all jobs initially in the stream were finished running.

Experiment 1

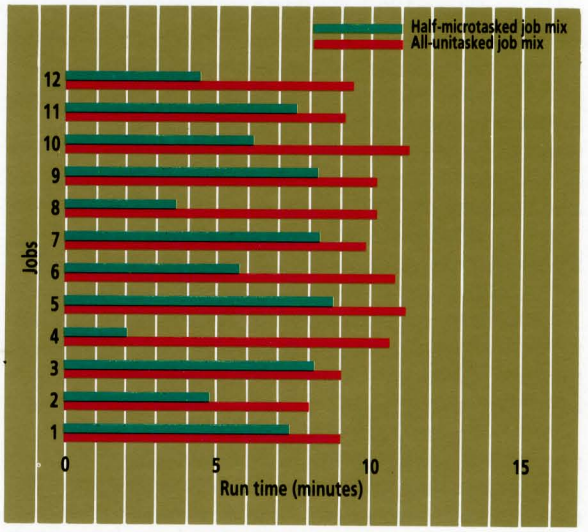
The baseline mix for this experiment consisted of 12 copies of one job. These were identical except for the memory requirements imposed. Four jobs required 1.7 million words of storage, four required 2.5 million words, and four required 3.5 million words. The job was computationally intensive and highly parallel. It executed matrix-matrix multiplications with *N*-by-*N* matrices, where *N* varied from 400 to 700. In a single-job (dedicated) environment, the job required 106 wall-clock seconds to complete. The second mix of jobs was identical to the baseline mix, except that six of the jobs were microtasked. In a single-job environment, the microtasked job completed in 28.5 wall-clock seconds (a speedup of over 3.7).

The results for these streams are superimposed in Figure 1, which shows the life lines of jobs in the batch streams. Jobs are numbered at the left of the figure in the order in which they were submitted in the two runs. This format also is used in the other figures. In each run, the memory requirements were 1.7 million words for jobs 1, 2, 11, and 12; 2.5 million words for jobs 3, 4, 7, and 8; and 3.5 million words for jobs 5, 6, 9, and 10. In the half-microtasked mix, the microtasked jobs were even-numbered.

Figure 1 shows that all twelve jobs in the half-microtasked mix finished before their counterparts in the all-unitasked baseline mix. The improvements are largest for the six microtasked jobs. By including six microtasked jobs in the batch stream, the total batch stream throughput time was reduced by 22 percent, from 11.3 to 8.8 minutes. The time needed for half of the jobs in the mix to complete was reduced by 38 percent from 9.9 to 6.1 minutes. This percentage improvement is larger than the 22 percent improvement because no microtasked jobs remained to use idle processors during the last several minutes of the half-microtasked mix run.

Close examination of the job life lines in the figure reveals that, of the six microtasked jobs, the larger-memory jobs finished last. The same is true of the unitasked jobs in each mix. This was to be expected, because memory requirement and absence or presence of microtasking were the only distinguishing features of the jobs.

Figure 1. Results of Experiment 1.



Even greater throughput improvements could be expected from similar experiments using Cray Research's UNICOS operating system and the Autotasking capability of the CFT77 Fortran compiler.

The fact that large-memory jobs finished last in the static batch streams had two consequences: fewer jobs fit in memory simultaneously and a higher percentage of the wall-clock time was spent in rolling jobs near the end of each batch run than was spent near the beginning. These facts were observed with large granularity snapshots of system activity provided by the Cray station software at the Pittsburgh Supercomputer Center. Similar observations were made in all experiments described here.

Generally, two or three jobs were executing simultaneously near the beginning of a run, while only one or two executed at a time in the middle and toward the end. The relatively low value of two seconds set for the COS job scheduler *in memory thrash lock* contributed to the large amount of roll time. Rolling jobs to or from the DD-49 disks proceeded at a sustained rate of about 1.2 million words/second, and so several seconds were required to roll the large-memory jobs in the experiments.

A final note on Experiment 1: none of the processor idle times for the microtasked batch jobs was greater than .5 percent. This is similar to what was observed for the matrix multiplication job in a single-job environment, indicating that microtasking worked as efficiently for individual jobs in batch as it did in the other environment.

Experiment 2

The baseline mix for Experiment 2 consisted of 12 copies of one job with, again, four jobs in each of the memory categories: 1.7 million words, 2.5 million words, and 3.5 million words. The job was the same as that used in Experiment 1, except that synchronous disk I/O occurred after each matrix multiplication when the resultant matrix was written to disk. I/O occurred with varying granularity and frequency. I/O granularity varied from .28 to .90 seconds (about one-half to two time slices). I/O occurred every 1.2 to 5.6 seconds (about three to thirteen time slices). In a single-job environment, I/O accounted for approximately 15 to 20 percent of the total 130 wall-clock seconds used by the job. The microtasked

version of the job finished 2.33 times faster. In a single-job environment, synchronous disk I/O bottlenecks greatly reduce the degree of parallelism in the job. A second mix of batch jobs for Experiment 2 was identical to the baseline mix, except that six of the twelve jobs were microtasked. A third mix, consisting of microtasked versions of all twelve baseline jobs, also was used in the experiment.

The results for these three batch streams are superimposed in Figure 2. The memory requirements of the jobs in each mix were as in Experiment 1: 1.7 million words for jobs 1, 2, 11, and 12; 2.5 million words for jobs 3, 4, 7, and 8; and 3.5 million words for jobs 5, 6, 9, and 10. The microtasked jobs in the half-microtasked mix were the even-numbered jobs.

Figure 2 shows that the inclusion of six microtasked jobs in the batch stream reduced total throughput time by 12 percent, from 14.8 minutes for the all-unitasked mix to 13.1 minutes for the half-microtasked mix. Reduction when all jobs were microtasked was 28 percent, from 14.8 to 10.7 minutes. Time required for half of the jobs in a mix to complete was reduced by 20 percent, from 13.4 to 10.8 minutes, for the half-microtasked mix. Reduction when all jobs were microtasked was 26 percent, from 13.4 to 9.9 minutes.

As in Experiment 1, the first jobs to finish in the half-microtasked mix were the microtasked jobs (six out of the first seven). Also, as in Experiment 1, all jobs in both batch mixes containing microtasked jobs finished before their counterparts in the baseline unitasked mix. However, three microtasked jobs (numbers 2, 4, and 8) finished about 10 to 15 percent later when the number of microtasked jobs in the mix was increased from six to twelve.

When the microtasked version of the job involving matrix multiplications and disk I/O was run in a single-job environment, the Cray utility PERFMON reported that additional "slave" processors were idle about 9 percent of the time they were assigned to the job. In the batch mixes here, none of the processors assigned to microtasked jobs spent over 3.5 percent of the time idle. This indicates that in the present case microtasking is more efficient in each batch job than it is in the same job running in a single-job environment.

Experiment 3

The baseline mix in this experiment consisted of two copies of six different jobs. The six jobs were applications of the general-purpose partial differential equation package PLTMG by R. Bank et al.[4,5] PLTMG can be used to model many physical phenomena in two spatial dimensions, including aerodynamic flows and structural vibrations. The applications used here are a subset of those used in a related experiment described in Reference 2. Run times for the six applications in a single-job environment varied from .5 to 10.5 minutes. Memory requirements for the twelve baseline jobs were imposed as follows: 2.5 million words for four jobs, 3.5 million words for four jobs, and 4.1 million words for four jobs. As often occurs in practice, the longest-running jobs required the most memory. One other batch mix was used in the experiment; the twelve jobs in it were identical to those in the baseline mix, except that six (one for each application) were microtasked.

Figure 2. Results of Experiment 2.

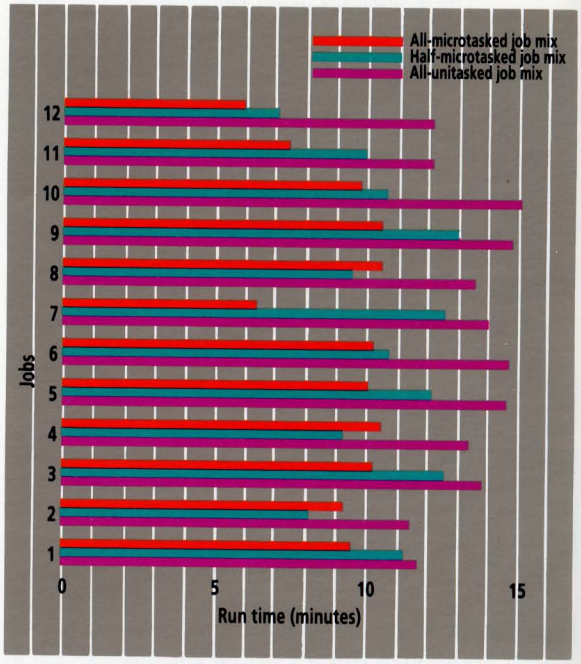
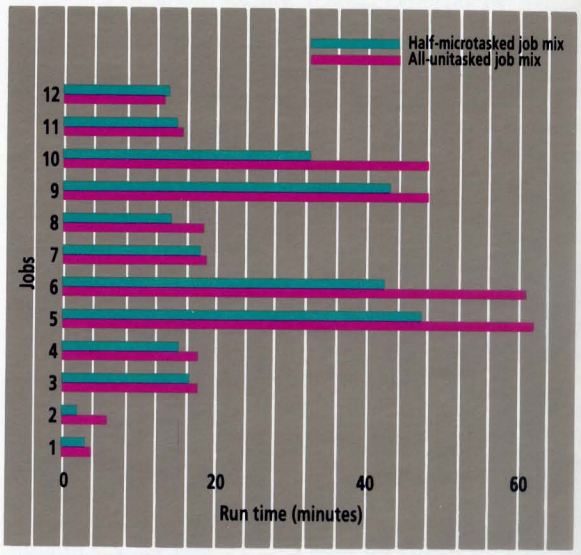


Figure 3. Results of Experiment 3.



Like much general-purpose mathematical software, PLTMG is CPU-intensive, highly modular, and has a low-to-moderate degree of parallelism (about 60 to 65 percent for most of the applications here). With four processors in a single-job environment, speedups observed in microtasking the six applications ranged from 1.22 to 2.34, with a time-weighted average of 1.77. Packages with only moderate parallelism often are not considered good candidates for microtasking. We soon will see, however, that microtasking such software makes a good deal of sense when the software is to be run in batch.

Results of running the two batch streams are superimposed in Figure 3. Memory requirements for the jobs in each mix were 2.5 million words for jobs 1, 2, 11, and 12; 3.5 million words for jobs 3, 4, 7, and 8; and 4.1 million words for jobs 5, 6, 9, and 10. Each pair of consecutively numbered jobs ran the same application. The even-numbered jobs in the half-microtasked mix were the microtasked jobs. In

unitasked single-job runs, the only applications that required more than 2.5 minutes to complete were the one corresponding to batch job pair 5 and 6 (10.5 minutes) and the one for batch job pair 9 and 10 (6.4 minutes). Speedups of microtasked versions of these runs were 2.34 and 1.54, respectively.

Close examination of Figure 3 reveals that all jobs except number 12 in the half-microtasked mix finished sooner than their counterparts in the all-unitasked mix. The largest absolute improvements were for the large-memory, long-running jobs (numbers 5, 6, 9, and 10). In each batch stream, only one job at a time could fit in memory during the period when only these four jobs remained in the batch stream. The total throughput time was reduced by 24 percent, from 62 to 47.2 minutes, when six microtasked jobs were introduced. The relative improvement due to microtasking the smaller-memory, shorter jobs is apparent when one compares times needed for half of the jobs in the two mixes to complete. This time was reduced by 17 percent, from 17.8 to 14.8 minutes, when microtasked jobs were introduced.

A final note on Experiment 3 concerns data on idle processors returned by PERFMON: for each microtasked batch job, the "master" processor spent under 5 percent of its time idle, while the three "slave" processors were idle about 55 to 65 percent of the time. This is similar to what was observed with the same applications in a single-job environment.

Summary and discussion

Results of three experiments on a four-processor CRAY X-MP system showed that including microtasked jobs in batch streams can improve system resource utilization and throughput. In the experiments, the times needed for both half and all of the jobs in a batch stream to complete were reduced by about 25 percent when half of the jobs were microtasked. Results of one experiment also showed the marginal benefits of including more microtasked jobs in a batch stream. Improvement in throughput was achieved when microtasked jobs made use of processors that otherwise would have been idle. Most of the idle processor time recovered by microtasking probably was that due to oversubscription of memory, that is, by the relatively large-memory jobs. This occurred in all experimental batch runs, as it often does in practice. Processors also were idle, however, in the unitasked runs in all experiments when rolling of jobs occurred and in one experiment when jobs carried out some synchronous disk I/O.

Results presented here should not be considered best-case results. A need exists for experiments to assess more thoroughly the benefits of microtasking in batch. Such experiments should, for example,

- ☐ Vary COS job scheduler parameters
- ☐ Allow jobs to enter a batch stream from time to time
- ☐ Use heterogeneous job mixes (perhaps typical of those at a particular Cray installation)
- ☐ Include the use of SSD and asynchronous I/O (BUFFER IN/BUFFER OUT)
- ☐ Include interactive sessions (with Cray Research's UNICOS operating system)

More of the power of multiple-CPU Cray systems will be exploited by encouraging microtasked applications in batch environments.

Nevertheless, evidence indicates that much more of the power of today's and tomorrow's multiple-CPU Cray systems will be exploited by encouraging microtasked applications in batch environments.

The results reported here also suggest that conventional charging algorithms should be re-examined. Conventional cost allocations should be changed in at least two cases. In one case, machine cycles spent by idle processors waiting on semaphore should not be charged to microtasked batch jobs because microtasked jobs surrender their slave CPUs to processorless jobs on request. These cycles can be tracked on Cray systems with the hardware performance monitor feature. A second instance where change is needed is in the case of large-memory batch jobs, that is, jobs that require more than $1/N$th of the total central memory on an N-processor system. Those that use multiple CPUs with any reasonable efficiency should cost less than their unitasked counterparts because the former deprive other jobs of system resources for less wall-clock time than do the latter. A general approach for cost allocation in multiprocessor environments that motivated much of the present work is described in Reference 6. ▬

Acknowledgments

This article is based on a paper of the same name that appeared in the proceedings of the fall 1987 Cray User Group meeting in Bologna, Italy, September 22-26, 1987. The author gratefully acknowledges partial research support and use of the Pittsburgh Supercomputing Center's facilities provided through NSF grant no. ASC-8519354. Appreciation also is expressed to the many people who contributed to this work. They include Jerry Kennedy of the Westinghouse Electric Corporation, which provides facilities management for the Pittsburgh Supercomputing Center (PSC); Robert Stock, Jim Ellis, Doug Fox, and others at the PSC; Jeff Nicholson and Lauren Radner of Cray Research; and Ben Dembart and Ken Neves of Boeing Computer Services.

About the author

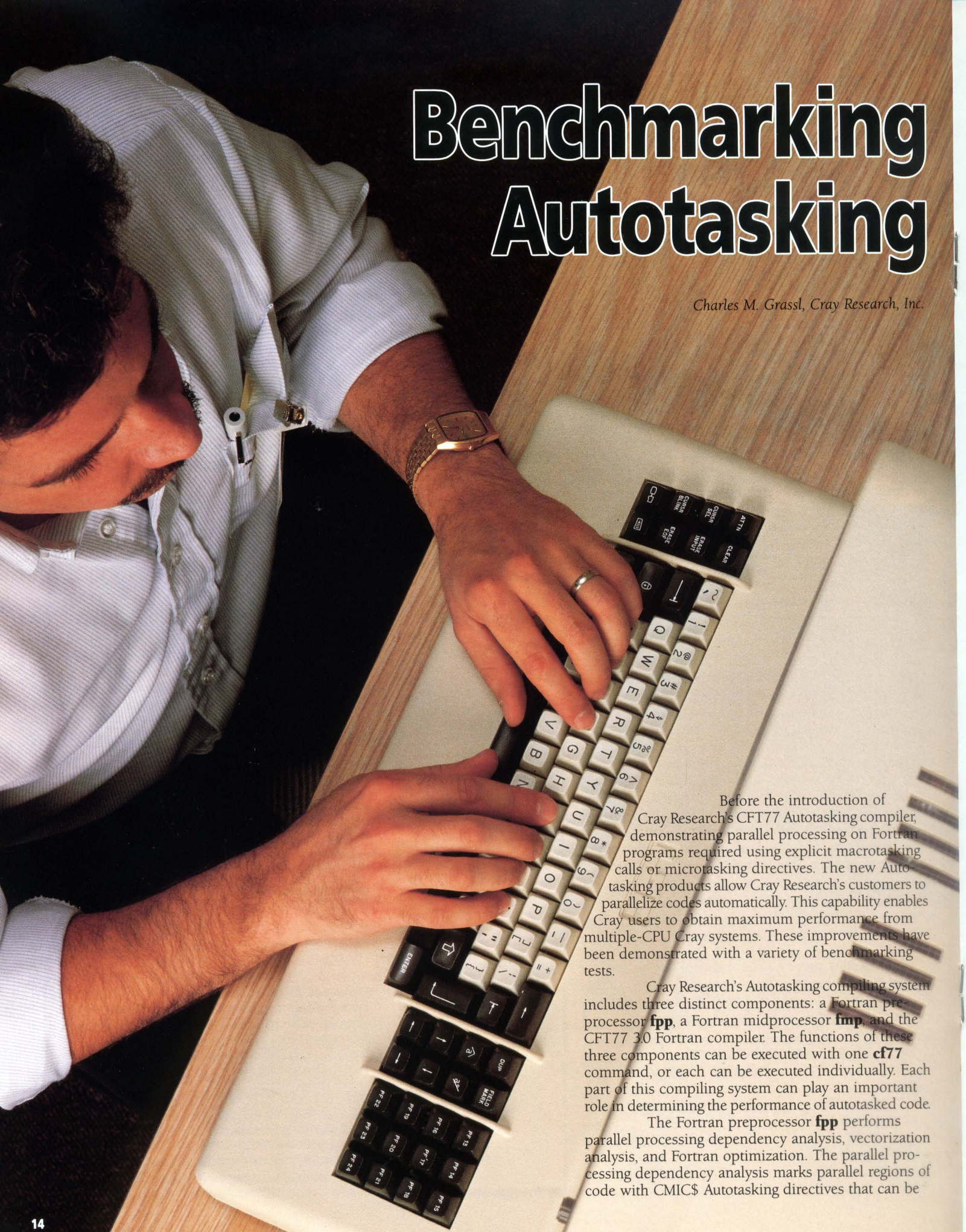
Michael Bieterman is a numerical analyst in Scientific Analysis and Computing at Boeing Computer Services. Before joining Boeing in 1984, he conducted research and supported biomedical scientists in the Computer Division of the National Institutes of Health. He received a Ph.D. degree in mathematics from the University of Maryland in 1982.

References

1. Cray Research, Inc., *CRAY Y-MP, CRAY X-MP EA, and CRAY X-MP Multitasking Programmer's Manual*, Pub. No. SR-0222, May 1989.
2. Bieterman, M., "Microtasking General Purpose Partial Differential Equation Software on the CRAY X-MP," *The Journal of Supercomputing*, Vol. 2, pp. 381-414, 1988.
3. Cray Research, Inc., Benchmarking Utilities Manual, September 1986.
4. Bank, R. E., PLTMG Users' Guide, Edition 4.0, Technical Report, Department of Mathematics, University of California at San Diego, March 1985.
5. Bank, R. E. and T. F. Chan, "PLTMGC: A Multi-Grid Continuation Program for Parameterized Elliptic Systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, No. 2, pp. 540-559, April 1986.
6. Dembart, B., "Special Report: An Approach to Allocating Costs in a Multiprocessor Environment," Report ETA-TR 24, Boeing Computer Services, January 1985.

Benchmarking Autotasking

Charles M. Grassl, Cray Research, Inc.



Before the introduction of Cray Research's CFT77 Autotasking compiler, demonstrating parallel processing on Fortran programs required using explicit macrotasking calls or microtasking directives. The new Autotasking products allow Cray Research's customers to parallelize codes automatically. This capability enables Cray users to obtain maximum performance from multiple-CPU Cray systems. These improvements have been demonstrated with a variety of benchmarking tests.

Cray Research's Autotasking compiling system includes three distinct components: a Fortran preprocessor **fpp**, a Fortran midprocessor **fmp**, and the CFT77 3.0 Fortran compiler. The functions of these three components can be executed with one **cf77** command, or each can be executed individually. Each part of this compiling system can play an important role in determining the performance of autotasked code.

The Fortran preprocessor **fpp** performs parallel processing dependency analysis, vectorization analysis, and Fortran optimization. The parallel processing dependency analysis marks parallel regions of code with CMIC$ Autotasking directives that can be

```
                DIMENSION FZ(NX, NY, NZ) CZ(0: NZ1)
                  ...
P----             DO 220 I = 1, NX
P V---            DO 220 J = 1, NY
P V D-            DO 220 K = 2, NZ
P----- 220        FZ(I, J, K) = FZ(I, J, K) - CZ(K) * FZ(I, J, K - 1)
```

Figure 1 (top). A recent benchmark executes at 5 MFLOPS on a CRAY Y-MP/832 computer system without the Autotasking compiler.

```
CMIC$ DO ALL SHARED(NX, NY, NZ, FZ, NZ1, CZ) PRIVATE(I, J, K)
      DO 77001 I = 1, NX
       DO 77002 K = 1, NZ - 1
CDIR$    IVDEP
          DO 77003 J = 1, NY
           FZ(I, J, 1 + K) = FZ(I, J, 1 + K) - CZ(1 + K) * FZ(I,J,K)
77003     CONTINUE
77002   CONTINUE
77001 CONTINUE
```

Figure 2 (bottom). When the same code is autotasked, it performs at 1020 MFLOPS on a CRAY Y-MP/832 computer system.

translated by the midprocessor **fmp**. The vectorization analysis marks vectorizable DO loops with CDIR$ IVDEP compiler directives. This vectorization analysis complements and enhances the vectorization performed by the CFT77 compiler. The **fpp** Fortran optimization includes library call substitution for equivalent source code, DO-loop transformations, and subroutine inlining.

Fortran optimization is particularly useful on CRAY-2 systems, in which the performance of scalar- and vector-code substituted library routines is dramatically better than that of scalar- and vector-code Fortran DO loops. An example of library substitution can be found in the Livermore Fortran Kernels (LFK) benchmark. The 24th kernel, or DO loop, performs a "search for first minimum." The **fpp** preprocessor recognizes this DO loop as being equivalent to the SCILIB routine ISMIN. With this substitution, the performance of the loop increases from 4 MFLOPS to over 50 MFLOPS on a CRAY-2 system. The CRAY Y-MP system performance increases from 6 MFLOPS to over 30 MFLOPS.

Other examples of vectorization and Fortran optimization are found in the NAS kernel benchmark. One of the kernels, or subroutines, in this benchmark program is the equivalent of a matrix multiply. The **fpp** preprocessor recognizes this kernel and replaces the Fortran source with a call to the SCILIB routine SGEMM. On the CRAY-2 system the original code runs at approximately 120 MFLOPS. The one-CPU version of the SGEMM routine runs at nearly 450 MFLOPS. The parallel version of SGEMM runs at approximately 1500 MFLOPS. On CRAY Y-MP systems, the original Fortran code runs at 280 MFLOPS and the parallel version of SGEMM runs at over 2 GFLOPS. This method of Fortran optimization is also a method of parallelizing a program. In this case, the Fortran optimized code calls an Autotasking library routine. In the future, additional library routines will include parallel versions.

The Fortran preprocessor **fpp** enhances vectorization most by performing loop transformations. These loop transformations include loop inversion and loop splitting. Loop inversion in this case refers to changing the order of DO loops. If, for instance, a Fortran code consists of double nested DO loops in which the inner DO loop is not vectorizable but the outer DO loop is vectorizable, then **fpp** will, if possible, rewrite the two DO loops with the original outer DO loop as the inner DO loop. Consider the nested DO loops in Figure 1, which appeared in a recent benchmark.

The code in Figure 1 is a partial listing as generated by **fpp**. *P* refers to parallel, *V* refers to vector, and *D* refers to dependency. The preprocessor transforms the original code in Figure 1 to the code in Figure 2, in which the vectorizable loop is on the inside, the data-dependent loop is in the middle, and the parallel loop is on the outside. The resultant code can be executed in vector and parallel modes.

For NX = NY = NZ = 125, the original code executes at 5 MFLOPS on a CRAY Y-MP/832 system without the Autotasking compiler. The optimized code runs at 1020 MFLOPS on a CRAY Y-MP/832 system.

The Fortran midprocessor usually is used to translate CMIC$ directives inserted by the preprocessor **fpp** into Fortran statements recognizable by the CFT77 3.0 compiler. The midprocessor **fmp** also can be used to analyze data scoping. If, for instance, a programmer manually has inserted a CMIC$ DO. . . directive without regard to the scope of the data directive, then the programmer can use the **fmp** midprocessor to scope variables within the range of the DO loop. Scope of data refers to data referencing, which can be local or global. That is, the data under question can be private to a task or shared by parallel tasks. The scoping feature of the midprocessor **fmp** greatly simplifies a programmer's job for parallel programming. Often, it is merely enough to identify parallel regions and then use the **fmp** midprocessor to scope variables within a parallel region. Before the advent of Autotasking, a large amount of effort usually was required to scope variables. Variables were not scoped explicitly in microtasking; rather, a variable's scope was determined by its declaration. Changing the scope of a variable in microtasking usually requires changing the dimension declarations, common statements, and call sequences. With the new Autotasking feature, and in particular the **fmp** midprocessor, one is able to facilitate parallelism with considerably fewer changes to the original code.

CPUs	CRAY Y-MP/832 system CFT77 2.0	CFT77 3.0	CRAY-2S/4-128 system CFT77 2.0	CFT77 3.0
1	49	84	24	41
2	—	129	—	56
4	—	185	—	82
8	—	200	—	—

Table 1. Results of LINPACK benchmark performance in MFLOPS.

Prior to Autotasking, parallel processing software used external Fortran calls to implement concurrency. The new CFT77 Fortran compiler generates inline instructions to manipulate shared registers and semaphores. This feature allows the compiler to schedule instructions more efficiently. The inline instructions reduce the overhead associated with implementing concurrency.

The Fortran preprocessor has been extremely useful when running many so-called standard benchmarks. Perhaps the most often noted benchmark is the LINPACK benchmark from Argonne National Laboratory. The benchmark program performs lower upper (LU) matrix decomposition on a 100-by-100 matrix of real numbers. This algorithm is from the area of dense linear algebra and is not presumed to be a definitive test of computer systems; rather, it is only a test of a specific algorithm. The preprocessor **fpp** is able to optimize and parallelize the original Fortran source code for this benchmark. The **fpp** optimized code has the kernel subroutines pulled inline. This optimization reduces subroutine calling overhead and allows concurrency analysis across the inlined subroutines.

Parallel processing speedup is not linear; three factors are responsible:

- ☐ The speedup of the entire program is limited by the proportion of parallelism (Amdahl's law)
- ☐ Multiple CPUs using shared memory incur inter-CPU memory contention (memory contention refers to more than one CPU attempting to access the same memory bank concurrently)
- ☐ Small granularity (short parallel tasks) can incur shared-register contention (shared-register contention is the condition in which more than one CPU attempts to access the same shared register concurrently)

The first factor, Amdahl's law, is usually the primary one limiting parallel speedup of entire programs. The second factor, inter-CPU memory contention, is more of a concern with CRAY X-MP and CRAY-2 memories, but has relatively little effect on CRAY Y-MP systems. The memory can allow only one reference per clock period per memory bank or section. This causes some CPUs to wait on their memory references. The memory contention can be alleviated by building more or faster banks, adding more structure to the memory organization, or by designing algorithms with fewer memory references. The final factor, shared register contention, limits the smallest or minimum granularity for parallel tasking. The first and third factors limit the performance of the 100-by-100 all-Fortran LINPACK test on the CRAY Y-MP/832 system.

With **fpp** optimizations, approximately 80 percent of the program running time is performed in parallel. Approximately 93 percent of the computations are performed concurrently, and most of the nonconcurrent computer results in this program are computed in scalar mode. These scalar calculations require more time to compute than do vectorized calculations. As the vectorized and parallel portions of this program speed up, the scalar portion tends to dominate the total run time.

The complete LU decomposition requires approximately 0.008 seconds of CPU time when running at 84 MFLOPS. The complete algorithm requires 667,000 FLOPs. The parallelism in this benchmark program is at a very low level, or of small granularity. The parallel processing of the entire program uses approximately 50,000 different concurrent tasks. Each task has an average size of 133 FLOPs; this amount of computation, when performed at the average 84 MFLOPS, requires approximately 250 clock periods. For each task, each CPU must execute approximately 30 clock periods of "guarded" code. Additional CPUs on this problem eventually are hindered by shared register contention.

Program	CRAY Y-MP/832 system CFT77 2.0	CFT77 3.0
NAS Kernels		
MXM	303	2250
CFFT2D	132	483
CHOLSKY	106	378
BTRIX	160	238
GMTRY	146	1277
EMIT	142	1190
VPENTA	175	583
Average	157	549

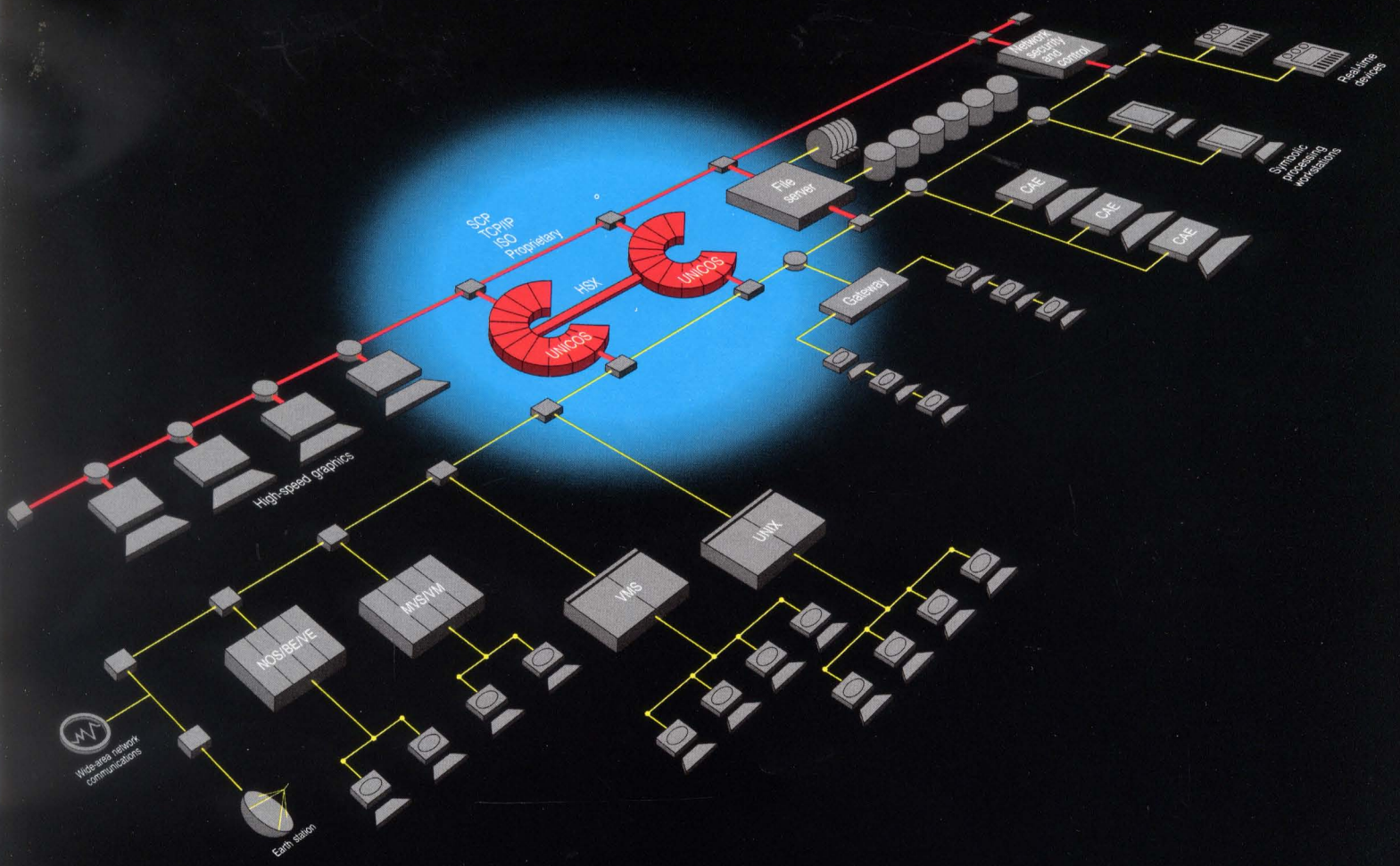
Table 2. Performance of the CRAY Y-MP/832 system in MFLOPS using two versions of the CFT77 compiler. CFT77 3.0 includes the Autotasking function. Parallel speedup varies greatly for these vector kernels.

The LINPACK benchmark demonstrates that the Autotasking software is able to optimize and parallelize the algorithm. The benchmark further demonstrates that the CRAY Y-MP system is able to parallel process at a relatively fine-grained level of parallelism. The CRAY Y-MP system's ability to parallel process on such a fine scale is impressive. The NAS kernel benchmark is considered a vector benchmark because all of the kernels potentially are vectorizable. This characteristic generally favors multiple-pipe machines for good performance. For example, the four-pipe NEC SX-2 system runs at over 300 MFLOPS on this benchmark. Achieving this speed on a Cray system requires use of multiple CPUs. Using the CFT77 2.0 compiler system, which utilizes only one CPU, the NAS kernel benchmark runs at 150 MFLOPS on a CRAY Y-MP system. Using the CFT77 Autotasking compiler, the CRAY Y-MP/832 system ran at over 500 MFLOPS. ▬

Each issue of CRAY CHANNELS includes a technical article that offers insights into the Cray environment. The editors thank Chris Hsiung and Jim Schwarzmeier for their regular technical advice.

About the author

Charles Grassl is manager of Cray Research's Benchmarking Department. Before joining Cray Research in 1984, he was employed by Sperry Semiconductor, where he was a process engineer and modeled integrated circuit fabrication. Grassl earned a Ph.D. degree from the University of Wisconsin at Madison.



Network supercomputing

integrating Cray power into varied environments

David Thompson, Cray Research, Inc.

If all Cray system users diagrammed their computing environments, the diagrams probably would be as distinct as the users themselves. Because a single vendor rarely provides a total computing solution, most Cray systems operate in heterogeneous environments that incorporate hardware and software products from a variety of vendors. To accommodate the needs of users in heterogeneous network environments, Cray Research and its customers have developed the concept of network supercomputing. Cray Research offers the widest range of network supercomputing capabilities and aims at developing transparent supercomputing environments in which users will not need to expend extra effort or endure unnecessary delays because of various formats or protocols among different makes of hardware or software. As a result, Cray users can be confident that their computer systems will fit into existing and future computing environments.

An open systems standard-based approach

Scientific computing environments increasingly are designed with an open systems, standards-based approach. Such an approach involves the use of standard interfaces that allow heterogeneous systems to operate interactively. This approach is popular because it does not lock customers into single-vendor solutions. The diagram above illustrates Cray Research's direction in network supercomputing.

During the past decade, most operating systems and networking capabilities have been proprietary products. Today the UNIX operating system and

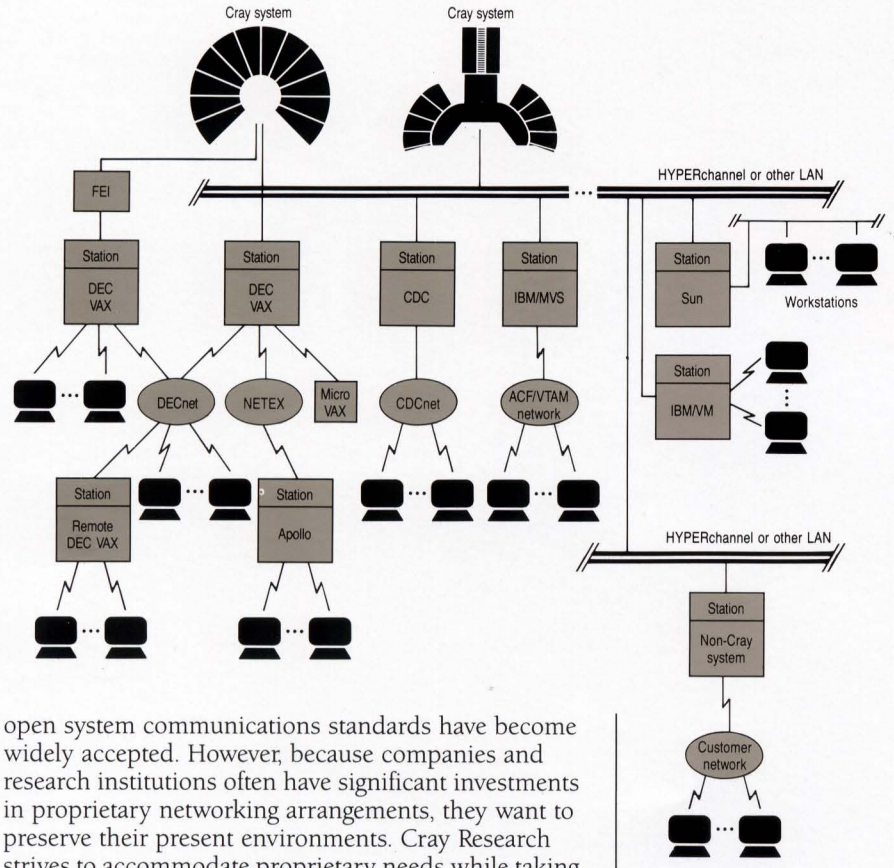


Figure 1. Cray station configuration.

open system communications standards have become widely accepted. However, because companies and research institutions often have significant investments in proprietary networking arrangements, they want to preserve their present environments. Cray Research strives to accommodate proprietary needs while taking advantage of the inherent economy, interconnectivity, and flexibility of standards-oriented solutions.

Vendor proprietary system support

Cray Research provides connectivity to the IBM (MVS and VM), Digital Equipment (VMS), and Control Data (NOS/BE, NOS, and NOS/VE) environments through their respective proprietary networking offerings — IBM's System Network Architecture (SNA), Digital Equipment's DECnet, and CDC's CDCNET. The connectivity of Cray systems is provided by hardware and software capabilities that interface with these proprietary solutions via front-end gateway systems. Figure 1 illustrates how Cray Research's station software products can be used to provide user-familiar transparent access to vendor-proprietary environments.

Through the implementation of standards, primarily the TCP/IP networking protocol, Cray Research provides connectivity to virtually all UNIX-based mainframe systems and workstations in an open systems environment. This capability allows seamless access from workstations to a Cray system, independent of the user location on the network. Figure 2 illustrates the Cray TCP/IP network at Cray Research's computer center in Mendota Heights, Minnesota.

As part of its network-supercomputing support, Cray Research plans to support the emerging ISO standards. The ISO protocols have been under definition and development for the past few years, and we expect them to be customer requirements in the near future. The next challenge is to provide for the coexistence of ISO and TCP protocols because TCP will remain in use and ISO will increase in importance.

Networking tools

Network supercomputing can be described best as efficient supercomputer use in a multivendor, heterogeneous, standards-oriented computing environment. The most significant components of this environment are: general and special-purpose workstations; a file server for permanent and archival data storage; Cray systems; general and special-purpose systems not manufactured by Cray Research; and networking capabilities that connect all system components and accommodate proprietary and standard implementations.

Most users access Cray systems through networks and network applications. Figure 3 illustrates the variety of network services that can bring the power of Cray supercomputers to an end user located anywhere in a network:

- ☐ The batch processing service allows users to submit a Cray job from within the network, view and control its execution status, and route the output to a chosen destination.
- ☐ File transfer allows users to copy files to nodes on the network. It maps internal block structure, character representation, and other attributes.
- ☐ The interactive service provides desktop terminal access to Cray systems.
- ☐ Data access is the ability to mount network-resident file systems on the Cray system and vice versa. This service allows data to be shared networkwide on the record or block level.
- ☐ Distributed services provide tools to distribute applications across the network. They provide the platform for support of network-based resources such as file servers, graphics servers, and windowing packages.
- ☐ Graphics services support visualization packages, which play an important role by enhancing user productivity in scientific computing environments.

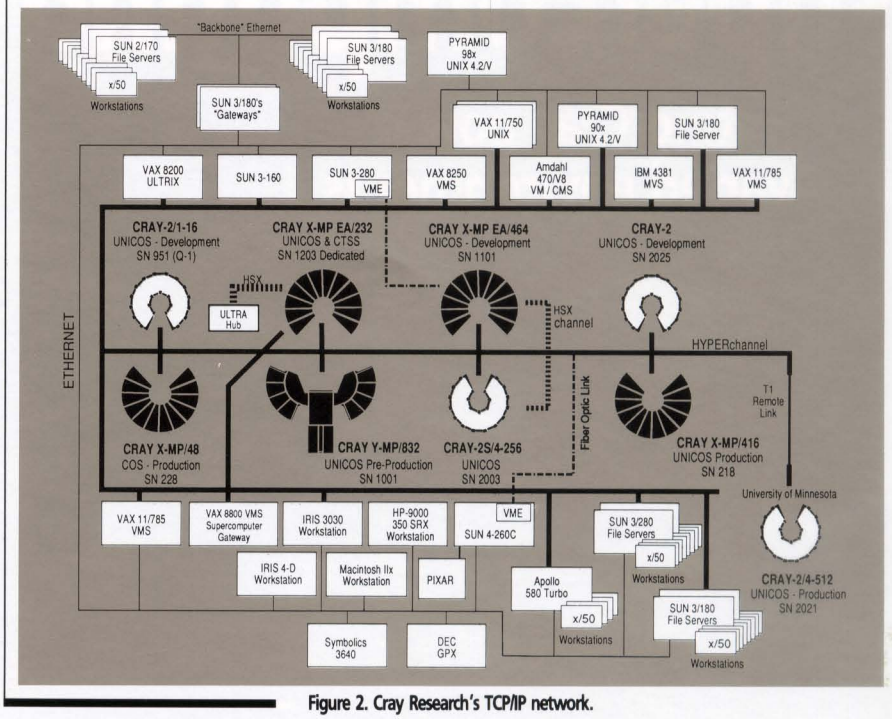


Figure 2. Cray Research's TCP/IP network.

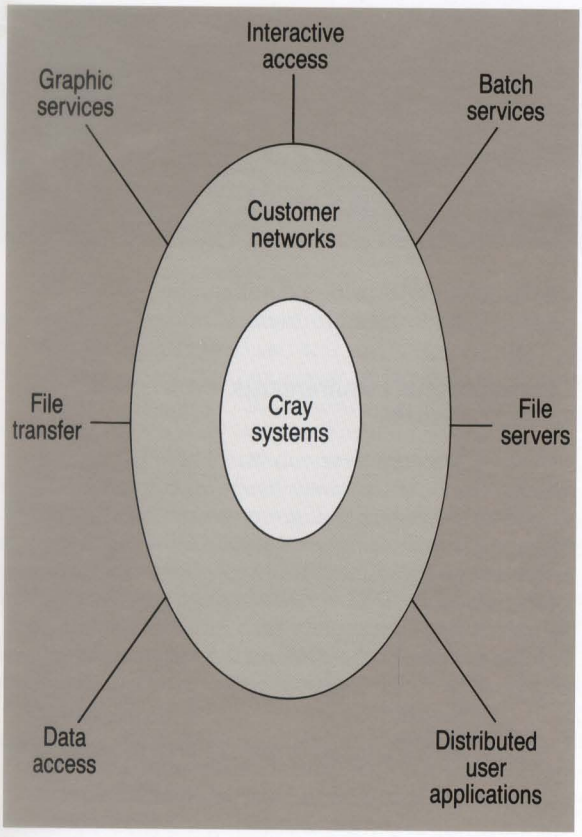


Figure 3. Cray network services.

Tomorrow's supercomputing environment

Figure 4 illustrates Cray Research's view of the supercomputing environment of the future. We view the network surrounding the supercomputers from a Local Area Network (LAN) and a Wide Area Network (WAN) perspective. The supercomputer environment should not be constrained by physical boundaries.

The LAN capability is tiered. We look to a very high-speed LAN (at least 1 Gbit/sec) to connect Cray systems, file servers, special purpose servers, and high-speed graphics devices. The file servers must support Cray systems as well as other systems connected via the network, providing network data-access and hierarchical data storage capabilities. The special-purpose servers provide various functions such as data base management and symbolic processing. The graphics devices can be used for real-time animation requiring large volumes of data at very high speeds.

The next LAN tier, through which most users access the Cray systems, is provided by medium-speed commercially available technology. Today this function is provided by 10-Mbit/sec Ethernet to 50-Mbit/sec HYPERchannel technology. We see this capability moving to the 100-Mbit/sec Fiber Distributed Data Interface (FDDI) standard.

Wide Area Network (WAN) capabilities are provided by a variety of commercially available solutions. Many environments currently have remote connections capable of T1 (1.54-Mbits/sec) speeds. In some situations this speed is adequate; however, we anticipate commercially available T3 (44.5-Mbits/sec) capabilities in the near future.

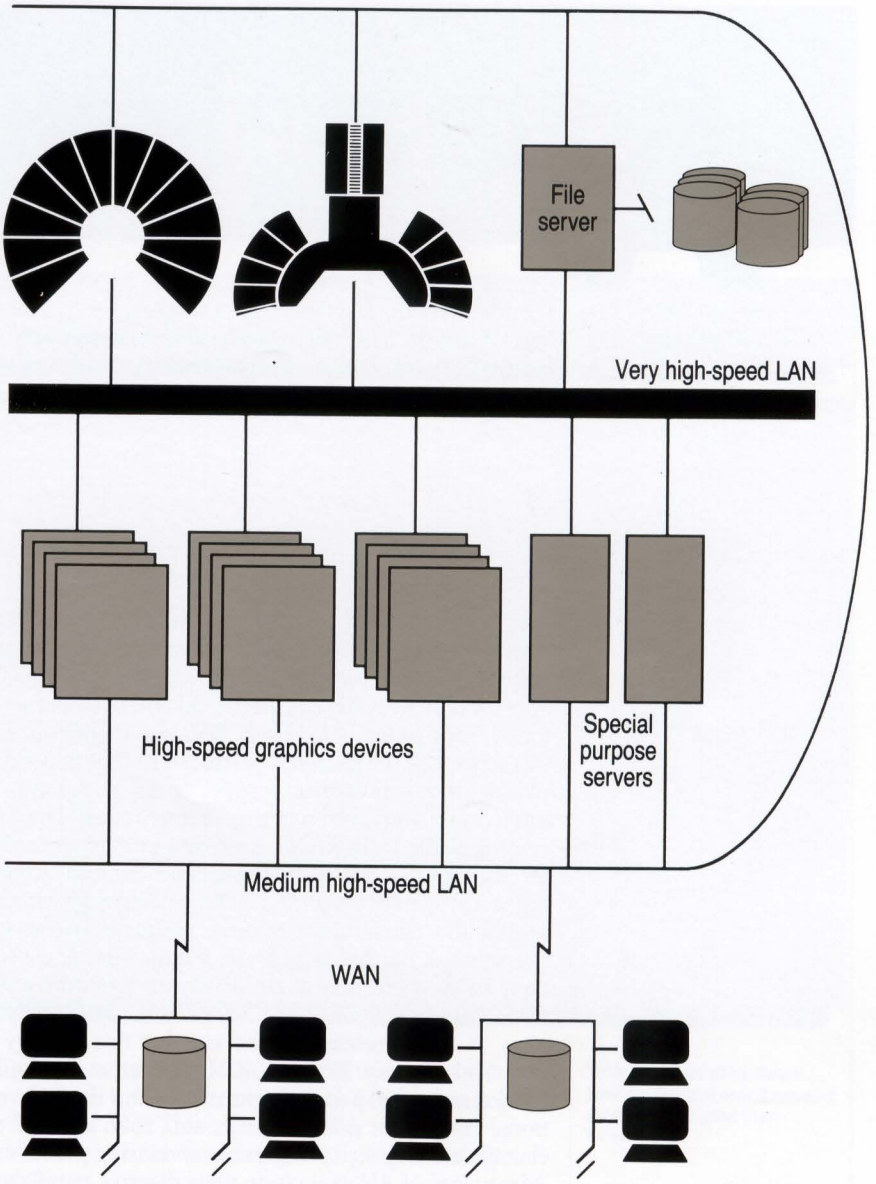


Figure 4. The supercomputer environment of the future.

Supercomputer networking allows users to preserve networking investments and to adapt future technology into a unified computing environment. Cray Research's networking software enables users to integrate Cray supercomputers transparently into environments that include personal computers, workstations minicomputers, mainframes, storage devices, printers, plotters, and image recorders, forming a comprehensive computational environment. The result: Cray power is available on the desks of scientists, engineers, and researchers, enhancing productivity and making possible more creative solutions.

About the author

David Thompson is group leader of Cray Research's Network Communication group. Before joining Cray Research in 1985, he was with Control Data Corpororation for 16 years. Thompson earned an M.S. degree in mathematics from the University of Oregon in 1965.

Artificial neural network

Denton A. Olson, Cray Research, Inc.
Stanley C. Ahalt, The Ohio State University
Roger S. Barga, Battelle Pacific Northwest Laboratories
George L. Wilcox, University of Minnesota

Researchers working in the field of artificial intelligence (AI) hope to design computer hardware and software systems that can mimic human thinking. One approach to AI is based directly on the structure and functioning of the brain itself. Researchers pursuing this approach are designing artificial neural networks (ANNs) that simulate nerve cells, or neurons, their interconnections, and patterns of interaction. The processing elements in ANNs, sometimes called neurodes, receive signals from neighboring neurodes and, according to a system of weighting factors, determine whether to pass the signals along to other neurodes farther up the network. In this way, ANNs handle signals essentially in the same way as do networks of neurons in the brain.

ANN researchers believe that this strategy may lead to a new generation of computers that will excel at tasks that people perform well and that conventional computers don't perform well, such as pattern classification, speech recognition, and vision processing. Advantages of ANNs include their massive parallelism and adaptivity. Massive parallelism provides the potential for high-speed decision making and fault tolerance. Neural networks can be "trained" rather than programmed in the classical sense, and this adaptivity allows the performance of neural networks to improve with experience. Their responses become more accurate or complete as they are fed more data; that is, they "learn."

ANNs and their simulations sometimes are called connectionist models, parallel distributed processing models, neuromorphic systems, and neurocomputers. Their characteristics typically include[1]

- ☐ Many simple neurodes with a set of interconnections with variable weights
- ☐ Memories stored in patterns of weights
- ☐ Data processing by "spreading activation" from input to output
- ☐ Behavior determined by teaching or training, rather than by programming
- ☐ Control by transfer function and learning law(s), not by a separate control program
- ☐ Natural functioning as an associative memory
- ☐ High fault tolerance
- ☐ Potential for self-organization

Computational requirements for artificial neural networks

Although research into ANNs is proceeding rapidly, little computer hardware has been built to evaluate the various configurations of networks that researchers have designed. Instead, most of the designs function as software simulations that run on existing computers. Because ANN simulations are computationally intensive, supercomputers such as Cray systems are important tools for the continued design and refinement of advanced ANN models.

Researchers are using Cray computer systems to simulate, study, and develop numerous artificial neural network algorithms. Cray systems lend themselves naturally to ANN simulations because most of the calculations performed during the simulations are vector oriented, and the simulations are characterized by fine-grained parallelism, which is well-supported by pipelined vector processors. Cray system resources that support efficient ANN simulations also include

- ☐ Floating point processing; the mode of arithmetic in neural network models usually is floating point
- ☐ Fast multiply-accumulate operations; the intensive processing in neural network models is predominately in multiply-accumulate operations
- ☐ Large, high-speed main memory; the speed and size of an ANN model is dependent on the speed and size of the main memory of the computer on which it is run
- ☐ Network supercomputing; remote Cray systems networked with local graphics workstations provide a powerful environment for ANN simulation

The U.S. Government's Defense Advanced Research Projects Agency (DARPA) has established a 28-month, $33-million program in artificial neural networks to help determine their potential advantages, further develop neural network theory, and develop advanced hardware technology. The limitations of conventional computers in neural network research have been addressed by the *DARPA Neural Network Study*[2] and by independent researchers in government, education, and industry. These limitations include inadequacies in storage, speed, and user flexibility. The Simulation/Emulation Tools and Techniques Panel of the DARPA study group recommended that "...access to high-end computational facilities be available for large [artificial neural] network (e.g., vision) research and development."[2] Supercomputers provide the fastest operating implementation for artificial neural network simulation (Table 1).

simulation on Cray systems

The computational units of artificial neural networks are best understood in terms of the number of interconnections (storage) and interconnections-per-second (speed) within a layer or between layers of neurodes. As ANN configurations are scaled up, by increasing the training iterations or the number of neurodes in the network, the computation and memory requirements grow proportionately. The computational speed of an artificial neural network is determined largely by the rate at which signals can be transmitted across interconnections. The 50 million interconnections-per-second rating shown in Table 1 for a single-CPU CRAY X-MP system is a theoretical upper limit based on the number of multiply-and-add operations that can be performed per second.

Hardware class	Processor	Interconnections per second (millions)
Supercomputer	CRAY X-MP system (1 CPU)	50
Massively parallel computers	Thinking Machines Corporation Connection Machine	13
	BBN Advanced Computers, Inc. Butterfly	8
Bus-oriented computers	TRW MK V	10
	TRW MK III	0.5
Attached processors	SAIC SIGMA-1	5-8
	Texas Instruments Odyssey	5
Micro/mini computers and workstations	DEC VAX	2
	SUN 3	0.250
	IBM PC AT	0.160
	Apple Macintosh	0.005

Table 1. Classes of hardware used for artificial neural network simulation (adapted from Reference 2).

Distributed artificial neural network simulation

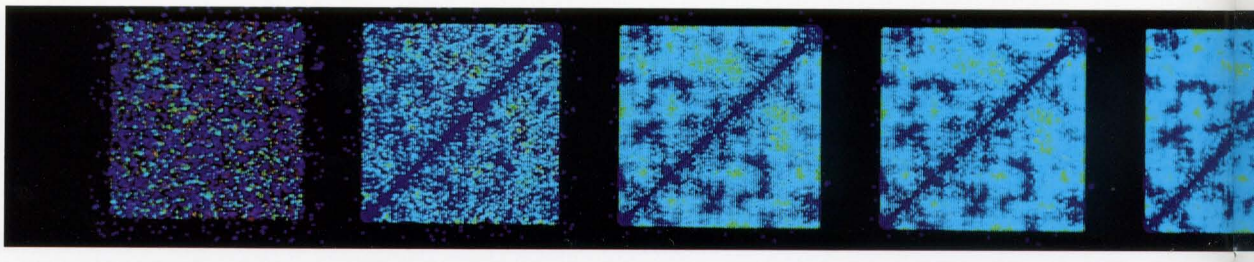
At Ohio State University, researcher Stanley Ahalt is applying ANN models to the vector quantization of speech and vision data. Vector quantization is a technique used to encode data statistically to minimize the bandwidth required for transmission or storage. The technique exploits the natural clustering, or redundancy, of information that occurs in speech or image data. Consequently, one can find and transmit a cluster's center instead of the original data without losing information. Using the CRAY X-MP/28 system at the Ohio Supercomputer Center (OSC), Ahalt and his associates have implemented a type of ANN called a Frequency Sensitive Competitive Learning neural network. This is a "winner-take-all," unsupervised ANN specifically designed to perform vector quantization. The model produces a vector quantization code book for compressing speech data by large factors while maintaining its audio quality.

Ahalt's research also has resulted in the development of the Neural Shell, an interactive Sun Microsystems Suntools window-based distributed environment for experimenting with ANN algorithms. The ANNs developed with the system can be executed on Sun Microsystems workstations in scalar mode and on CRAY X-MP systems in vector mode. The TCP/IP facilities ftp and telnet provide the communication capabilities. A graphics display facility allows the user to view the interconnection weight matrices or the results generated by the Neural Shell. The Neural Shell supports several ANN algorithms: Hopfield, Hamming, back-propagation, Kohonen self-organizing feature maps, and frequency-sensitive competitive learning.

ANN models can be prototyped and tested quickly on a workstation, then scaled for more intensive computation on a Cray system. The performance advantages of a supercomputer are demonstrated, for example, by simulations conducted by Ahalt using the learning phase of the back-propagation ANN algorithm. One processor of the CRAY X-MP/28 system at OSC outperformed a Sun Microsystems 3/60 workstation by a factor of 375. This algorithm, like most used in ANN research, is written in the C programming language.

The multilayer perceptron (MLP), a back-propagation network, probably is the most widely used artificial neural network and was the first to use the error back-propagation algorithm for learning network connection strengths. Roger Barga and associates at Battelle's Pacific Northwest Laboratories are using their distributed Battelle Connectionist Simulation Environment (BCSE) to evaluate the application of the MLP and other ANNs to acoustic emission classifications and seismic waveform discrimination problems.

BCSE allows local graphics workstations to control and display interactively the output of ANN simulation engines implemented on CRAY X-MP systems and other computers on Ethernet, HYPERchannel, and NSFnet networks. Each machine does what it does best: interactive graphics and user interface activities run on the graphics workstation, and intensive simulation computations run on the Cray system. The enormous quantities of data that ANN simulations can produce make visualization a major concern for artificial neural network researchers. By translating output data into images, computer graphics provide an effective way to reveal information that otherwise would



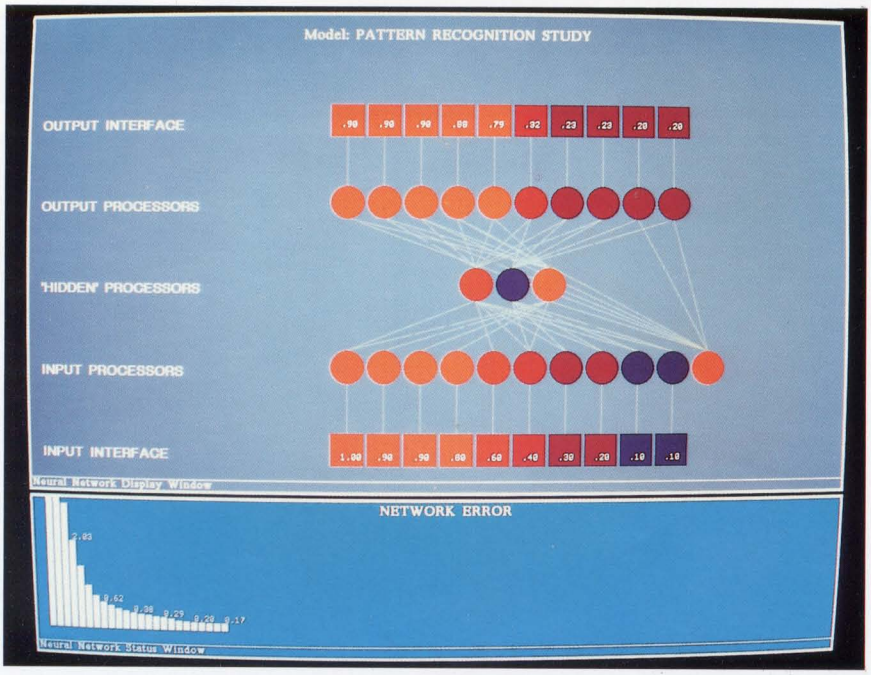


Figure 1. BCSE color graphics display of an ANN learning simulation running on a CRAY X-MP system. The ANN connection weight and processing element display is updated by the color graphics workstation while the simulation runs on the Cray system.

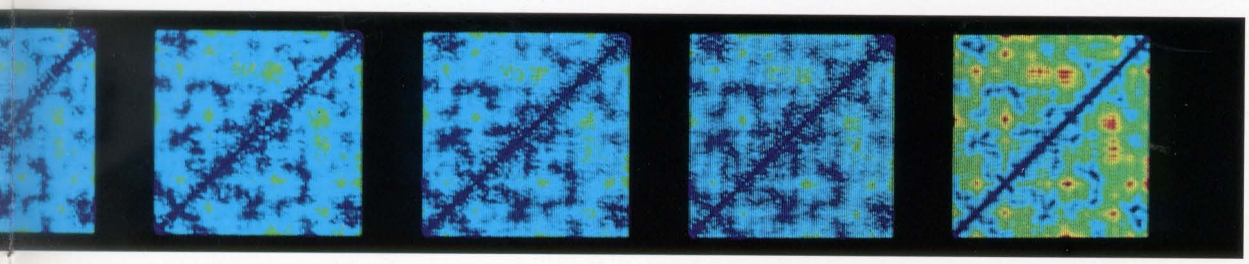
be lost in a flurry of numbers. The TCP/IP protocol suite and Berkeley sockets provide remote users with transparent, uniform access to the various computing resources on the network. The simulation modules for learning on all systems were implemented to perform operations on a standardized vector-data representation of the ANN connection weights and neurode values. ANN connection weights are data compressed prior to being transmitted with a 12:1-to-15:1 compression, using what amounts to a sequential quantization entropy reduction technique. The graphics interface module on the local workstation renders images from the vector representation of the ANN and displays the network in near-real time. This allows the researcher to view the ANN information as the computations are taking place, independent of the process running the ANN simulation on the Cray system. Icons on the workstation screen, which represent each neurode and connection, are updated to reflect the changing values (Figure 1). By remotely invoking the ANN simulation engine on the Cray system, researchers are able to evaluate (train and test) in minutes ANNs that contain thousands of neurodes. In contrast, only small ANNs can be evaluated on minicomputer or workstation simulators in a reasonable period of time.

Large-memory artificial neural network simulations

Large-memory computer systems are critical to the development of some ANN models. A project using the 512-Mword CRAY-2 system at the Minnesota Supercomputer Institute is underway to develop an ANN to analyze protein sequence data. DNA- and protein-sequencing technology enables researchers to determine rapidly the primary structure of protein molecules — the one-dimensional sequence of amino acids of which protein molecules are composed. However, information concerning the tertiary, or three-dimensional, structures of proteins has grown much more slowly, and this information is critical to understanding protein function. Although the primary structure of a protein completely determines its secondary (chain folding and twisting) and tertiary (domain folding) structures, no procedure yet can predict the complete folded structure of a protein molecule from its amino acid sequence alone.

A project headed by George Wilcox at the Minnesota Supercomputer Institute applies ANN simulations to the protein folding problem. Wilcox has implemented a back-propagation ANN architecture called BigNet and optimized it for the CRAY-2/4-512 system at the Minnesota Supercomputer Center (MSC), an affiliate of the University of Minnesota. BigNet has been configured to form an association between sequence information and three-dimensional structure information for some of the smaller proteins with known structure. The first tests of BigNet are using three-layer networks with 130 to 260 input units that receive the sequence information, 130 to 260 hidden units, 17,000 output units that deliver the three-dimensional structure, and 2 to 5 million connections among the three layers. Learning progresses as BigNet adjusts its weights in an attempt to minimize the difference between the desired output with which it is presented and the output that it actually produces by forward propagation. BigNet has "learned" a great deal about the initial training set of 26 proteins, after 2000 presentations. Its first test examines whether it has formed a sufficient associative memory to recall all the structures in the training set given only the input sequences. Ultimately, after this first hurdle has been overcome, the network will be presented with a large training set (about 1,000 to 10,000 presentations). To test it for its retention of protein-folding rules, BigNet will be presented with sequences that are new to it but have known structures. Successful performance has been set at 95 percent correct predictions for these known structures.

BigNet was developed to model arbitrarily large ANNs with almost unlimited dimensionality. The program includes a Network Description Language (NDL) that allows an experimenter to configure the network for input-output data sets of arbitrary structure and dimensionality. Using the interactive UNICOS operating system on the MSC CRAY-2 system and the University of Minnesota Ethernet network, NDL programs can be created and executed interactively. The

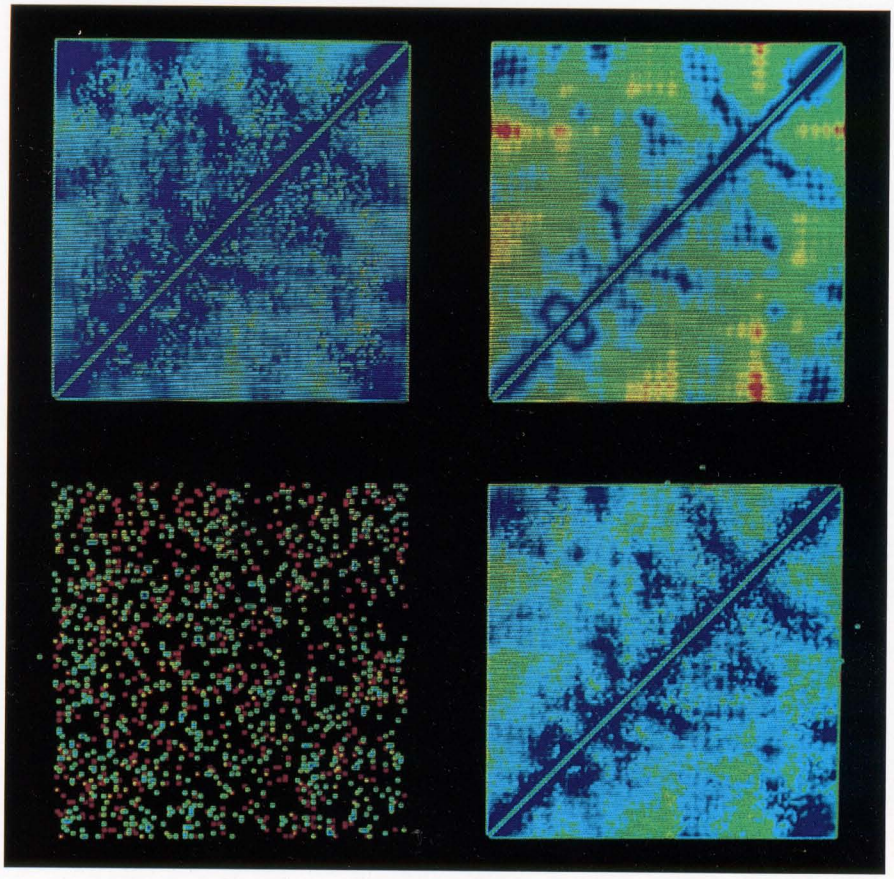


results of a simulation can be reviewed graphically at a Sun, NeXT, or Macintosh II workstation (Figure 2).

Conclusion

Although no one can predict the direction and success of future ANN technologies, in the short term two developments seem very likely. The first is the software "hybridization" of ANNs with conventional software systems and with expert systems. The second likely development is the hardware implementation of ANNs in analog VLSI circuitry.

Although the human brain's structure may hold the key to a future generation of computers, much work remains to be done before scientists will be able to approximate the computational power of the human brain in a computer system. Researcher Terrence Sejnowski of the Salk Institute and the University of California at San Diego, who has described the human brain as a "dedicated analog processor," suggests the following formula to determine the computational power of the human brain: The brain comprises about 10^{11} neurons and about 10^3 connections per neuron, with about 10 operations proceeding per connection per second.[3] These factors multiply out to about 10^{15} operations per second. Based on the current rate of increases in computer power, Sejnowski projects that by about the year 2015 computer power will catch up with human brain power!



Distance matrices for the proteins cytochrome c550 (above) and ferredoxin (across top of page) as output from BigNet at various stages during its training. The output matrix for cytochrome c550 is shown after 250 iterations (lower left), 500 iterations (upper left), and 1000 iterations (lower right). The target matrix is shown at upper right. The output matrix for ferredoxin is shown at 125 iterations (far left) and in increments of 125 iterations. The target matrix is at far right. Cytochrome c550 and ferredoxin are 2 of the 26 proteins in a training set that researchers are using to "teach" BigNet to identify three-dimensional protein structures from one-dimensional amino acid sequences.

About the authors

Denton Olson is an artificial intelligence specialist in the Industry, Science & Technology Department of Cray Research. He monitors artificial intelligence (AI) activities within Cray Research and outside of the company. He is involved in the optimization of Portable Standard LISP for Cray systems, benchmarking the LISP and REDUCE codes on Cray systems, prototyping coupled symbolic-numeric systems, and monitoring artificial neural network technologies and implementations on Cray systems. He received a B.S. degree in medical technology from Mankato State University in Mankato, Minnesota and an M.S. degree in computer science from North Dakota State University in Fargo, North Dakota.

Stanley Ahalt is an assistant professor in the Department of Electrical Engineering at The Ohio State University. His research interests include artificial neural networks and parallel computer architectures. He is the recipient of research grants from Cray Research, under which he directed development of the Neural Shell. He received a Ph.D. degree from Clemson University in South Carolina in 1986 and previously worked with Bell Telephone Laboratories where he developed local area network hardware and software.

Roger Barga is a research scientist in the Compuational Sciences Department of the Pacific Northwest Laboratories, which is operated for the U.S. Department of Energy by the Battelle Memorial Institute. His current research interests include computing with artificial neural systems, machine learning, and the development of artificial intelligence systems. He also teaches courses in artificial neural systems for Washington State University's Tri-Cities Extension. He received a B.S. degree in mathematics from Boise State University in 1985 and an M.S. degree in computer science from the University of Idaho in 1987.

George L. Wilcox is an associate professor of pharmacology at the University of Minnesota Medical School and a fellow of the Minnesota Supercomputer Institute. He received a B.A. degree in chemical physics from Columbia University in 1970 and M.S. and Ph.D. degrees in aerospace engineering from the University of Colorado, Boulder, in 1972 and 1975 respectively.

References

1. Caudill, M., "Basic Concepts of Neural Networks," The 4th Twin Cities IEEE-SIGART Symposium on Neural Networks & Parallel Computing, February 18, 1989.
2. DARPA Neural Network Study, AFCEA International Press, November, 1988.
3. Sejnowski, T., "What is the Computational Power of the Brain?" invited talk at AAAI-88, August 23, 1988.

CORPORATE REGISTER

Cray Research gains three chemical industry customers

Eli Lilly and Company has ordered a CRAY-2 computer system to be installed in the second quarter of 1990 at Lilly's Corporate Center in Indianapolis, Indiana. Lilly is the first pharmaceutical company to order a Cray supercomputer, as well as the first commercial company in the United States to purchase a CRAY-2 supercomputer. Cray Research also announced that it and Lilly have formed a unique relationship to advance the use of supercomputers in pharmaceutical research. Experts in computational chemistry from both organizations will explore new methodologies in the application of supercomputers to drug discovery. Lilly will use the CRAY-2 system to pursue applications in pharmaceuticals, agricultural products, and medical devices. The system will run Cray Research's UNICOS operating system, which is based on AT&T UNIX System V.

Monsanto, a company specializing in agricultural chemicals, pharmaceuticals, and biotechnology, has installed a CRAY X-MP EA/116se system. Monsanto, a new customer for Cray Research, installed the leased system in the second quarter of 1989 at its Life Sciences Research Facility in St. Louis, Missouri. The company is using the Cray system mainly for molecular simulations in the areas of biotechnology, agricultural chemicals, plastics, and artificial sweeteners. In addition, Searle, a subsidiary of Monsanto, is applying the Cray system to pharmaceutical research. Monsanto will use Cray Research's UNICOS operating system.

Sumitomo Chemical Engineering Company, a leading Japanese manufacturer of fine chemicals, has ordered a CRAY X-MP/EA 116se supercomputer. The company is a new customer for Cray Research, and its computer system will be the first Cray system in Japan devoted to industrial chemical applications. The purchased system will be installed in the third quarter of 1989 at the Sumitomo Computer Center in Osaka, Japan. Sumitomo will use the Cray system to model a diversity of chemical products, including agrichemicals and polymers. Sumitomo will use Cray Research's UNICOS operating system.

The Massachusetts Institute of Technology (MIT) and Cray Research have announced a five-year joint effort in supercomputer research that will include the installation of a CRAY-2/4-256 system on the MIT campus. The system will be used for scientific and engineering research. MIT will install the leased system in the third quarter of 1989. Cray Research will provide MIT with grants over five years to further research on the supercomputer.

Electricité de France (EDF), the French national utilities company, has ordered a CRAY Y-MP8/432 system and a CRAY Y-MP4/232 system. The systems will be installed in the fourth quarter of 1989 at EDF's research facility in Clamart, France. The systems will be applied to nuclear plant design and operation simulation, as well as electricity production and distribution. EDF will use Cray Research's UNICOS operating system. The systems will replace a CRAY X-MP/216 system, which was installed in May 1986, and a CRAY X-MP/28 system, which was installed in October 1987.

The **Royal Armament Research & Development Establishment** (RARDE), a British government defense research laboratory, has installed and accepted a CRAY X-MP EA/416 supercomputer, which will be upgraded to a CRAY Y-MP8/432 system in the second quarter of 1990. RARDE installed the purchased system at its main computer facility in Sevenoaks, Kent, England. The new system replaces a CRAY-1/S system installed since 1983, and will be applied to armor and anti-armor research as well as a range of defense system assessments.

Cray Research releases UNICOS 5.0

Cray Research announces the availability of version 5.0 of the UNICOS operating system, which is based on AT&T UNIX System V. UNICOS 5.0 provides significantly improved performance and functionality over previous releases, with new capabilities in areas such as security, connectivity, and resource management.

Bob Ewald, executive vice president of Software at Cray Research, said, "Cray Research developed the UNICOS operating system to combine the functionality of UNIX with the advantages of using a Cray computer system. UNICOS 5.0 reflects our commitment to providing users with a stable, powerful, and portable software environment for interactive and batch processing."

When Cray Research introduced the UNICOS operating system in April 1986, it was the first UNIX-based operating system for general-purpose supercomputing. Today UNICOS is the primary operating system on 79 Cray systems at 68 customer sites around the world, and is the operating system of choice for most new Cray system installations. UNICOS includes Berkeley extensions to the UNIX System V standard, as well as more than 200 programmer-years of Cray enhancements important to the high-performance computer environment.

UNICOS 5.0 enhances supercomputer performance and functionality with features such as,

- ☐ Multi-level security, which provides increased protection for sensitive information
- ☐ Increased connectivity, including support for the SUPERLINK connection to IBM systems, the extension of Sun Microsystems Network File System (NFS) capability across all Cray systems, and the introduction of a Remote Queuing System (RQS) facility that further enhances batch job submission facilities from remote UNIX systems
- ☐ Support for Autotasking, the automatic parallel processing function of Cray Research's CFT77 Fortran compiler, on all multiprocessor Cray systems
- ☐ Improved production environment, including online tape support, data migration facility, new schedulers, improved process and job accounting, and an expanded user-information database
- ☐ Extensions to real-time features, including pre-emptive resource allocation and a scheduler that prioritizes competing real-time processes based on the amount of CPU time required to meet the time constraints specified

For more information about the UNICOS operating system, contact the nearest Cray Research sales office.

Cray Research announces CDC NOS/VE Link Software 2.0

Release 2.0 of the NOS/VE Link Software logically links a Control Data Corporation (CDC) CYBER 180 series computer system running the NOS/VE operating system to a CRAY-2, CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, or CRAY-1 computer system running the Cray operating systems UNICOS or COS.

Major new features of the NOS/VE Link Software 2.0 release include

- ☐ File transfer gateway; Cray jobs now can use the Cray FETCH and DISPOSE data transfer statements to transfer data sets located on remote systems linked through the Permanent File Transfer Facility (PTF) or the Transmission Control Protocol/Internet Protocol (TCP/IP) File Transfer Protocol (FTP)
- ☐ Queue file transfer enhancements; Cray job submission and automatic output rerouting now is supported for remote systems linked through the Network Transfer Facility (NTF); additional procedures for remote NOS/VE and NOS systems are provided to facilitate Cray job submission
- ☐ NOS/VE 1.4.1 support
- ☐ USCP 5.0 support; a display similar to the UNIX ls command display now is supported through the NOS/VE Link Software
- ☐ Access to NOS or NOS/BE Station tapes; tapes written by the NOS or NOS/BE Station in transparent mode now can be accessed
- ☐ New installation and maintenance procedures
- ☐ Installation verification test suite
- ☐ On-line internal reference manual
- ☐ Resource control for spun-off jobs
- ☐ Corrective code

The following items are required to operate the NOS/VE Link Software 2.0: UNICOS 4.0 or later, or COS 1.16 BF3 or later; NOS/VE 1.4.1, level 716 AC; and a Cray Front-end Interface (FEI) or NSC HYPERchannel connection.

Features

NOS/VE Link Software features include

- ☐ Batch job submission from NOS/VE to the Cray input queue; the Cray job output is returned to the NOS/VE output queue or to a NOS/VE permanent file (this feature also is available on remote systems linked through the Queue File Transfer Facility (QTF) or NTF; the Cray job output automatically is delivered to the originating system)
- ☐ Data set transfer to and from the Cray computer system in response to the execution of the Cray ACQUIRE, DISPOSE, and FETCH data transfer statements within Cray jobs; this feature also is available on remote systems linked through PTF or FTP
- ☐ Full-screen refreshing status display of Cray jobs and the Cray system; Cray jobs are controlled by using the drop, kill, rerun, or switch function
- ☐ Cray interactive job execution from a NOS/VE terminal
- ☐ Processing of tapes written by the NOS or NOS/BE Station software in transparent mode
- ☐ Cray master operator functions
- ☐ System fine-tuning or problem resolution, using the performance tuning options, and the trace and debugging facilities
- ☐ Accounting for resources used by the station or station users

For more information about CDC NOS/VE Link Software, contact the nearest Cray Research sales office.

Version 3.01 of the MVS Station now available

MVS Station, version 3.01 adds several enhancements, including support for the UNICOS data format and the MVS Station communications software product. The MVS Station runs on an IBM or compatible computer running IBM's MVS operating system in either the MVS/SP1 or MVS/SP2 (XA) environment. The IBM system serves as a front-end gateway to a Cray computer system. The station software provides users with a wide array of communication facilities, including the submission, status updating, and control of batch jobs, file transfer (dataset staging) between systems, and interactive access to the Cray system.

Release 3.01 of the MVS Station supports CRAY X-MP EA, CRAY Y-MP, CRAY-2, CRAY X-MP, and CRAY-1 computer systems, as well as the Cray operating systems UNICOS (release 3.0 or later) and COS (release 1.15 or later).

New features

Version 3.01 of the MVS Station adds new features, including

- ☐ Support for the UNICOS data format, which allows a significant reduction of the station's processing and of system overhead used on the Cray system
- ☐ Enhanced support for Network Queueing System (NQS) statements
- ☐ Enhancements to front-end servicing for Cray tapes
- ☐ Increased station slot size, which allows greater flexibility in the passing of installation-specific information between MVS and Cray systems
- ☐ A new installation exit routine for text field defaulting to help supply default information when a user unfamiliar with IBM JCL commands has omitted necessary parameters
- ☐ Support for the job entry subsystem JES3 SP 2.2.1
- ☐ Improved error recovery, which allows the station to synchronize again with the Cray system when a message has been delayed or lost
- ☐ Support for the COS commands sweep, restore, and release

For more information about the MVS Station, contact the nearest Cray Research sales office.

Cray France S.A. announces 1988 research competition winners

Cray Research France S.A. has announced the 1988 winners of the Seymour Cray Research Proposal Competition, a contest to acknowledge excellence in research by French scientists. The 1988 competition included five disciplines: microelectronics, computer architecture, numerical simulation, parallel algorithms, and microrobotics.

Exceptional prize was awarded to Roland Glowinski, a professor at the University of Houston, and formerly an engineer at Institut National Recherche Informatique et Automatique, for his work, "A Numerical Approach to the Exact Boundary Controllability of the Wave Equation." First prize was awarded to Philippe Verchère de Reffye, director of the modelization laboratories at the Centre de Coopération Internationale en Recherche Agronomique pour le Développement, for his work, "Modelization and Simulation of Growth and Architecture of Plants." Second prize was awarded to Jean-Marie Normand, director of the PERCOLA project at Commissariat à l'Energie Atomique, Saclay, for his work, "PERCOLA: a 64-bit Computer." Third prize was awarded to Marie Farge, a researcher at Meteorological Laboratories at Ecole Normale Supérieure Paris, for her work, "Coherent Structures of Compressible Two-dimensional Turbulence." A special prize was awarded to Claude Brezinski, director of numerical and analysis laboratories and a professor at the University of Lille for his work, "Numerical Analysis." A special team prize was awarded to Germain Pot, Jean-Paul Chabard, and Philippe Hemmerich, engineers at Electricité de France, Chatou and Clamart, for their work, "N3S, a Numerical Simulation Method for Turbulent Industrial Flows."

Cray Research France S.A. also announced the opening of the 1989 competition, which will include the same categories and will follow the same rules as the 1988 competition. An independent jury develops the competition rules, reviews submissions, and determines the award participants. Cray Research France limits its participation in the contest to financing and logistical support. Winners of the 1989 contest will be announced in the first quarter of 1990.

APPLICATIONS UPDATE

MPGS offers distributed graphics

Cray Research's new Multipurpose Graphics System (MPGS) is an interactive, menu-driven visualization tool that allows Cray system users to access powerful graphics capabilities quickly and easily with the click of a mouse. Using the TCP/IP network protocol, MPGS is distributed between a UNIX-based workstation and any Cray system running Cray Research's UNICOS operating system. The Cray system handles memory and CPU-intensive tasks, while the workstation handles local graphics manipulations. This workload distribution ensures the efficient use of both computer systems and minimizes network data transfers. The required TCP/IP protocol ensures that information can be transmitted between a Cray system and a user workstation over any distance. For example, MPGS has been distributed between a workstation in Minnesota and a Cray system in Great Britain.

MPGS features and capabilities include

- ☐ Dynamic transformations
- ☐ Comprehensive data file support
- ☐ Flexible parts structure
- ☐ Line drawings
- ☐ Hidden line drawings
- ☐ Complete display of all nodes in the computational mesh
- ☐ Contours (isolines) of equal scalar magnitude
- ☐ Shaded images
- ☐ False color map with shaded images
- ☐ Vector arrows
- ☐ Particle traces
- ☐ Arbitrary two-dimensional clipping planes
- ☐ Iso-surface extraction
- ☐ Keyframe animation
- ☐ Transient data animation

MPGS handles geometry, scalar, and vector data files. The required MPGS geometry data file describes all geometric components, such as nodes, lines, elements, and solids. By assuming unstructured problems, the geometry file can be used for diverse applications, such as finite difference and finite element analyses. The optional scalar data file contains a single scalar for each node in the geometry file. It can be used to depict scalar quantities, such as temperature, stress, and pressure. The optional vector data file contains a single three-dimensional vector for each node in the geometry file. It can be used to depict vector quantities, such as displacements and velocities. MPGS has been used to display data from a diverse set of codes, including MSC/NASTRAN, PAM-CRASH, FLUENT/BFC, and FIDAP.

Cray Research currently licenses MPGS for distributed processing between Cray systems and Silicon Graphics workstations in the IRIS-4D series. Feature enhancements and support for other workstations is planned. For more information on MPGS, contact Kent Misegades, Cray Research, Inc., Industry, Science & Technology Department, 1333 Northland Drive, Mendota Heights, MN, 55120; telephone: (612) 681-3660.

Neural Shell available on Cray systems

The Neural Shell is an interactive environment for experimentation with artificial neural networks (ANNs). The ANNs developed with the system can be executed on Sun Workstations and on CRAY X-MP computer systems running Cray Research's UNICOS operating system. This flexibility enables users to prototype and test ANNs rapidly on a workstation, then scale them up for more intense computation on Cray systems. Currently the environment supports several ANN algorithms: Hopfield, Hamming, Back Propagation (BP), Kohonen Self Organizing Feature Maps (KSFM), and Frequency Sensitive Competitive Learning (FSCL), which was developed at the Department of Electrical Engineering at Ohio State University. Each algorithm is available in two versions: one for the workstation environment and a vectorized version for execution on Cray systems.

The Neural Shell allows users to specify and execute neural networks easily through the use of a mouse, menus, pop-up windows, and simple keyboard sequences. The program comprises three main parts: a simulation window program called the Neural Shell Window, a graphics-oriented window program called the Graphics Display Window, and nine independent, executable, nonwindow-oriented programs, collectively called the Neural Shell Simulation Programs (NSSP). The NSSP programs constitute the core of the Neural Shell and perform all of the compuational tasks.

The Neural Shell Window and the Graphics Display Window act as interfaces between the user and the NSSP. The Neural Shell Window provides an interactive window for the simulation of neural network algorithms. When the Neural Shell Window receives a commmand from a user, it first checks for any inconsistencies in the command and then delegates the task of executing the command to the appropriate NSSP neural net program. The Graphics Display program provides an interactive graphics window that can be called from the Neural Shell. It allows users to view graphically the interconnection weight matrices generated while using the Neural Shell. Users can examine portions of a matrix, adjusting the portion being viewed by zooming in or out. Automatic shading and determination of minimum and maximum matrix values also are supported. This graphics capability makes it easier to interpret the results obtained from the Neural Shell when running specific networks.

For more information on the Neural Shell, contact Stanley C. Ahalt, Department of Electrical Engineering, Ohio State University, Columbus, OH, 43210; telephone: (614) 292-0068, or contact Denton Olson, Cray Research, Inc., Industry, Science & Technology Department, 1333 Northland Drive, Mendota Heights, MN, 55120; telephone: (612) 681-3638.

USER NEWS

Cray system helps clean up atmosphere

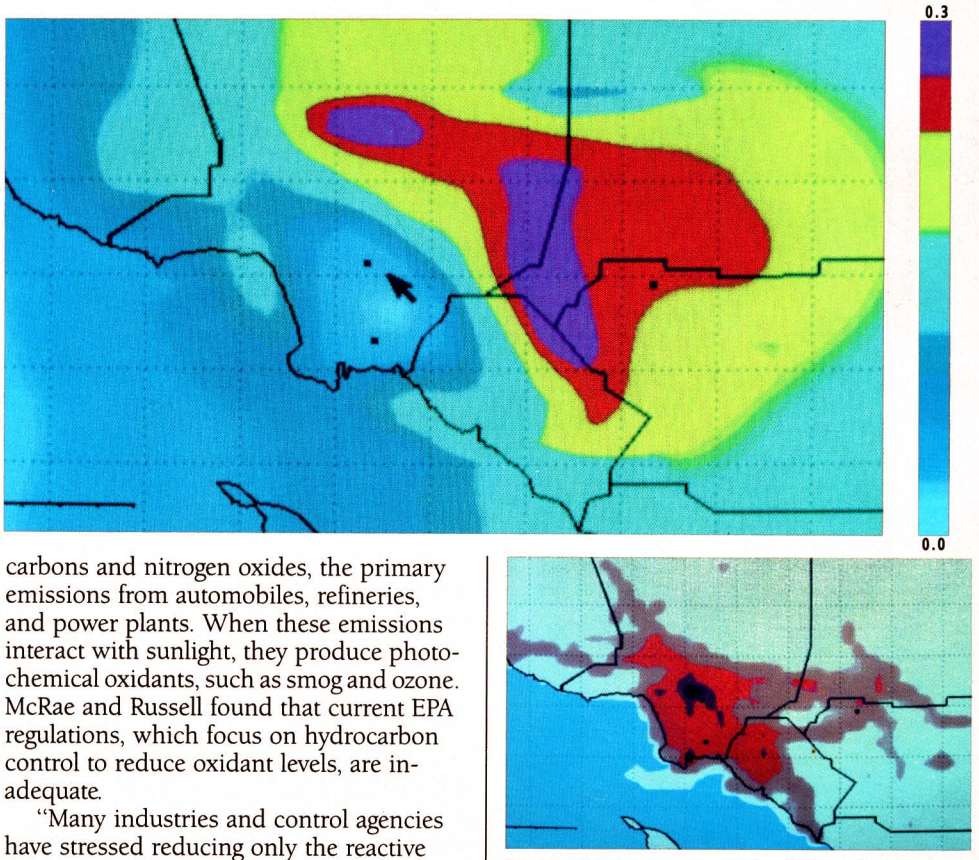
Environmental issues are receiving top billing on the world's list of concerns. Many fear that the quality of life on Earth may be endangered by acid rain, the growing ozone hole, the greenhouse effect, and other environmental hazards.

Don't panic — just take a deep breath. On second thought, make that a shallow breath; for more than four out of ten Americans, a breath of fresh air may not be very fresh. In fact, it may include unhealthy levels of ozone and other pollutants.

There is hope, however. Two researchers at Carnegie Mellon University (CMU) are using Cray supercomputers to develop strategies for cleaning up the atmosphere. By modeling the transport, transformation, and fate of pollutants in the Los Angeles area, Greg McRae and Ted Russell hope to find the most cost-effective ways to improve air quality. Their findings are being incorporated into air quality plans for the Los Angeles area, as well as alternative fuel policies. Their supercomputer research, which was performed at the Pittsburgh Supercomputing Center (PSC), has been supported by the California Air Resources Board, the South Coast Air Quality Management District, the Department of Energy, the U.S. Environmental Protection Agency, and Cray Research.

In the United States, about $30 billion is spent each year on air pollution controls, according to McRae, an associate professor of chemical engineering at CMU. "If we could shave off even a very small fraction of that cost by designing more efficient control strategies, that could mean enormous savings to the nation," he says. "In fact, the cost of a Cray supercomputer is minuscule compared to the expected savings — a single calculation could pay for a Cray system."

To support the Los Angeles Air Quality Management Plan, Russell and McRae examined the tradeoffs between reducing specific components of pollution, such as hydrocarbons and nitrogen oxides, the primary emissions from automobiles, refineries, and power plants. When these emissions interact with sunlight, they produce photochemical oxidants, such as smog and ozone. McRae and Russell found that current EPA regulations, which focus on hydrocarbon control to reduce oxidant levels, are inadequate.



(Top) Predicted ozone concentration field over Los Angeles, September 1982. The arrow points to the downtown freeway interchange, where emissions of nitrogen oxide are high. (Bottom) Pattern of nitrogen oxide emissions for the same area.

"Many industries and control agencies have stressed reducing only the reactive organic gases, but we found that the amount of both components (nitrogen oxides and hydrocarbon emissions) must be reduced to obtain real air quality benefits," explains Russell, an assistant professor of mechanical engineering at CMU. He and McRae elucidated a complicated relationship between emission levels and the amount of pollution in the atmosphere.

Modeling the formation and transport of air pollution over an urban region requires modeling hundreds of processes that happen on extremely varied temporal and spatial scales. For example, 50 to 100 chemical species, such as ozone, nitrogen oxides, and other pollutants, are tracked from the time of emission. To accomplish this, Russell and McRae evaluate the sources of emissions and how they vary with time, which requires modeling emissions from automobiles, refineries, factories, and even lawn mowers and barbeque grills. They model the chemical processes that form pollution and the dependence of these processes on atmospheric conditions such as temperature, atmospheric pressure, and the intensity of sunlight. Additionally, their models account for meteorological factors that transport the pollutants.

"In the computer, all these factors are interrelated by a mathematical model, a huge set of extremely complicated, but fairly well understood physical and chemical formulae," explains McRae. By plugging in data

that describe various atmospheric conditions, the researchers can predict levels of pollution and test the effects of hundreds of pollution control strategies.

For example, one of the more noted strategies involves switching from conventionally fueled vehicles to methanol-fueled ones. Russell explains, "By performing a variety of simulations for the years 2000 and 2010 in Los Angeles, we have shown that methanol can be useful in reducing ozone concentrations. Our studies have shown that switching from gasoline to methanol-fueled vehicles would reduce the ozone attributed to automobiles by about a factor of two." He adds, "Methanol is considered a clean fuel because when you burn it you don't get the myriad of reactive organic gasses that form when you burn gasoline, which is a multicomponent mixture."

The primary pollutant emitted by methanol-fueled vehicles is methanol, which is less reactive than gasoline in the atmosphere, producing less ozone in the same amount of time. Using methanol fuel has additional benefits: it reduces the amount of nitric acid that is formed in the atmosphere and reduces the number of pollutant particles created by diesel-fueled vehicles. Russell also is exploring combustion modeling to design equipment that will decrease pollutant emissions from installations that produce large amounts of sulfur and nitrogen oxide.

The methanol research, funded by the California Air Resources Board, is being used to develop guidelines and strategies for switching to alternative fuels. The results of their research have been incorporated into the Air Quality Management Plan for Los Angeles, which according to McRae is the most complex air quality plan in the United States, and will shape the direction of air pollution controls throughout the world. The plan calls for the reduction in pollutant emissions to improve air quality in the Los Angeles area by the year 2000, to meet the National Ambient Air Quality Standards. On a national level, their research even has impacted the Clean Air Act.

McRae and Russell are beginning to look at regional and global chemical modeling to study the ozone hole problem, regional acid deposition, and the greenhouse effect. "Our long-term goal is to build a global chemically reactive climate model," says McRae. "The only way we are going to achieve these solutions is to take advantage of multiple processors — so we are excited about using PSC's new eight-processor CRAY Y-MP system."

McRae adds, "We would not have been able to perform these calculations without the availability of a Cray system — we are talking about experiments that would take years to do on a VAX computer. With a Cray system we were able to perform experiments that have largely changed the way people think about air pollution controls."

CRAY-1 systems go public

When an invention lands in a museum collection, the honor carries some ambiguity. Museum placement typically comes as recognition of an invention's historical significance, but it also humbles the technology with a suggestion of obsolescence. Supercomputers have established their historical significance by advancing computer modeling and revolutionizing scientific research and engineering. Despite their newness, however, supercomputers have evolved so rapidly that museums already are acquiring the early systems. Retired CRAY-1 systems, state-of-the-art supercomputers in the late 1970s, have become prized catches for museums, though any suggestion of obsolescence should be tempered by the approximately 50 CRAY-1 systems that continue to operate around the world.

Among the more visible CRAY-1 systems on public display is serial number 14 at the Smithsonian Institution's National Air and Space Museum in Washington, DC. The system is part of an exhibit, "Beyond the Limits: Flight Enters the Computer Age," that celebrates the contribution of computers to all phases of the aerospace industry. Other retired CRAY-1 systems are on public display elsewhere in the United States and in Canada, France, Germany, and Sweden.

Some Cray Research customers have taken an interest in displaying their early supercomputers. The National Center for Atmospheric Research (NCAR) in Boulder, Colorado, became Cray Research's second customer when it acquired a CRAY-1 system, serial number 3, in 1977. The system no longer is in operation, but NCAR researchers, along with the general public, can see it in NCAR's main laboratory building where it has been on display since February.

"We wanted to retain the system for several reasons," explains Paul Halpern of NCAR's Office of Information Services. "We offer extensive public tour programs and the computer room is a favorite stopping point, but people can see the machines only through windows. We wanted them to have a chance to walk around a supercomputer and get a good look at it. At the same time, this system was our first supercomputer, and it played a critical role in helping us develop our computer-based climate models and our networking capabilities, so it has real historical significance for us. It was clear, too, that some of the scientists had developed a sentimental feeling for 'Old Serial 3.'" More than 7500 people tour the NCAR facilities each year.

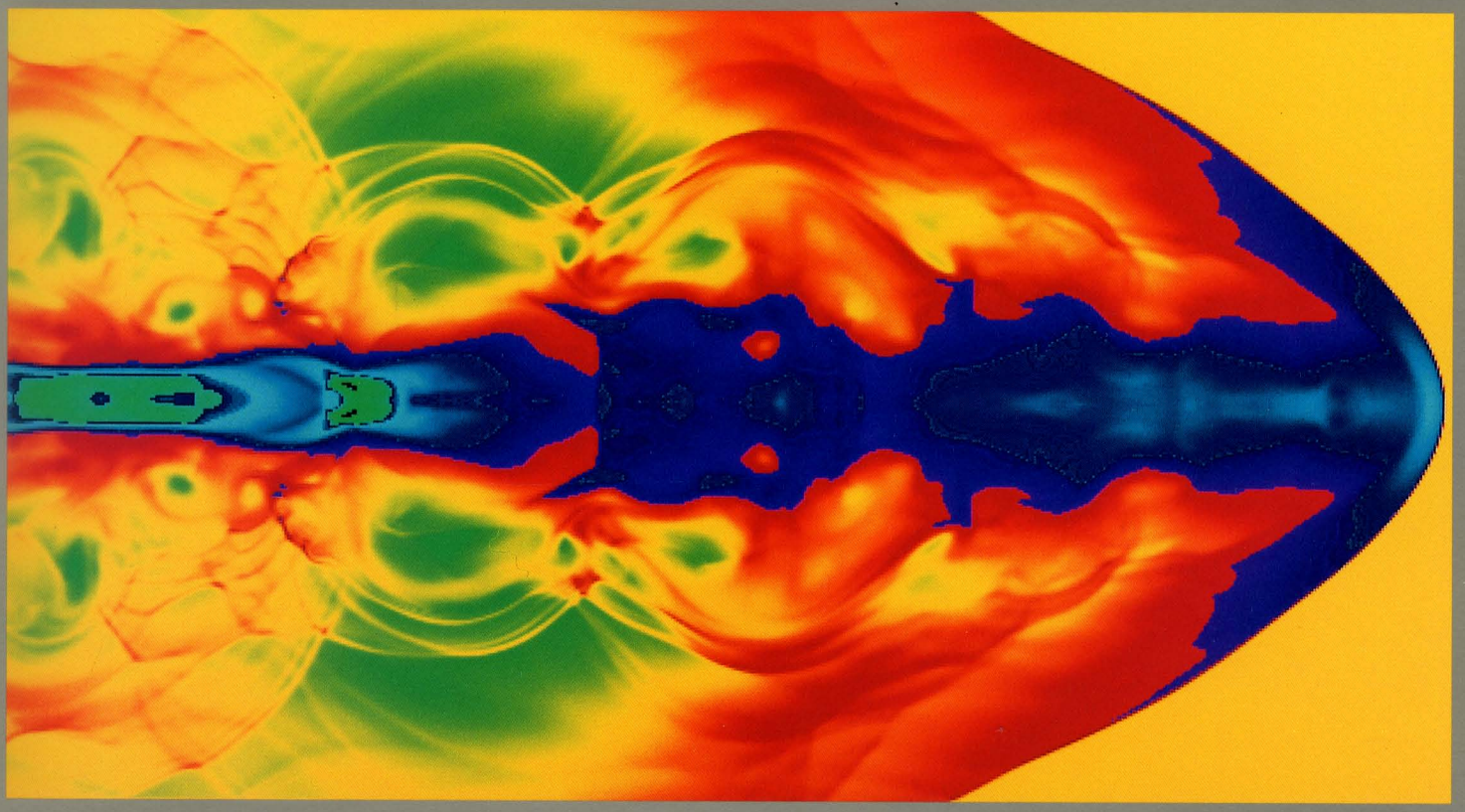
However, the CRAY-1 system at the Air and Space Museum probably will see more visitor traffic than any other system on public display. The Air and Space Museum is the most heavily visited museum in the United States, hosting more than nine million vistors each year. The CRAY-1 system is at the heart of "Beyond the Limits," and is located, naturally, in the aerodynamics section.

The exhibit underscores the critical role that computers, particularly supercomputers, have come to play in aerospace design and engineering. Supercomputers enable researchers to simulate complex airflows so that they can test many alternative design configurations quickly and inexpensively. Supercomputers have gone a long way toward relegating wind tunnels to tests of only the most computationally intractable flow geometries. "Computer-aided design has become so widely used in the aerospace world that engineers no longer even consider drawing up blueprints for a new airplane or spacecraft," said museum director Martin Harwit.

"In the aerodynamics section, we wanted to illustrate changes in aerospace design methods. The CRAY-1 system broke through large-scale CFD applications in aerospace design and engineering," added Paul Ceruzzi, exhibit coordinator at the Air and Space Museum. The aerodynamics section of the exhibit includes interactive displays with personal computers at which visitors can manipulate supercomputer graphics that were produced during aerospace design studies conducted at National Aeronautics and Space Administration (NASA) facilities. "The displays allow visitors to rotate and zoom in on the images in real time," explained Ceruzzi. The CRAY-1 system is part of the Air and Space Museums' permanent collection and will be on display for at least 10 years. Other sections of the exhibit include a scale model of a Boeing 737 with nacelles that were redesigned on a Cray system to accommodate larger engines, and a full-scale model of an X-29 fighter, an aircraft whose forward-swept wings also were designed with a supercomputer.

The CRAY-1 computer system inaugurated a new era of computational science and engineering, along with what is now a billion-dollar global supercomputer industry. Nonetheless, few nonscientists have had the opportunity to see a supercomputer close up. Now, thanks to exhibits like those at NCAR and the National Air and Space Museum, many people will be able to inspect first hand the standard bearers of supercomputing's early years.

GALLERY



This color plot of a cosmic jet is one in a series modeled on the CRAY X-MP/48 system at the San Diego Supercomputer Center (SDSC) from the calculations of David Payne and David Meier of NASA's Jet Propulsion Laboratory, Kevin Lind of the National Radio Astronomy Observatory, and Roger Blandford of the California Institute of Technology. Cosmic jets are powerful astrophysical phenomena that are collimated, supersonic flows, transporting mass, energy, and momentum from a central source. About 200 cosmic jets have been observed in a variety of astrophysical objects, ranging from protostars to quasars. The image was generated by Todd Elvins of SDSC on a DICOMED film recorder.

CRAY CHANNELS welcomes Gallery submissions. Please send submissions to the address inside the front cover.