Deficit law makes first round of budget cuts

Galen Gruman, Assistant Editor

The Balanced Budget and Emergency Deficit Control Act, popularly known as the Gramm-Rudman law, cut 4.3 percent of more than a third of the nation’s nondefense spending and 4.9 percent of more than a third of its defense spending as the law went into effect Mar. 1.

Programs exempt from the cuts include Social Security, most welfare programs, interest on the national debt, military payroll, and research into the Strategic Defense Initiative ‘Star Wars’ defense project. Health programs are not exempt from cuts, but are limited to one percent reductions in 1986 and two percent reductions thereafter.

The law requires $11.7 billion in cuts across the nonexempt programs during the 1986 fiscal year to help reduce the federal deficit, which will total $220.5 billion, the Office of Management Budget and the Congressional Budget Office jointly estimated in mid-January. Exempt programs together cost $646.3 billion. The total budget was $996.5 billion.

The law also sets maximum allowable deficits for the rest of the decade. Each subsequent allowable deficit is less than the previous year’s, until the federal budget is balanced in 1991. The maximum allowable deficit for 1987 is $144 billion. The maximum allowable deficits for 1988, 1989, and 1990 are, respectively, $108 billion, $72 billion, and $32 billion.

President Ronald Reagan’s proposed $994 billion 1987 budget has a projected deficit of $31.9 billion — just under the deficit ceiling. That budget increases defense spending by six percent and compensates for those increases with cuts in nondefense programs.

However, despite the proposal, an Office of Management and Budget spokesman predicted a 1987 budget deficit of $31.9 billion more than Gramm-Rudman allows. That translates to a further 8.8 percent cut in 1987 spending.

Defense cuts. Unlike nondefense programs, the Defense Department can spare some programs from cuts and transfer the burden to other areas — as long as the average is 4.9 percent. The Defense Department’s Advanced Research Projects Agency is perhaps hardest hit by this shifting of cuts.

ARPA’s original $5.9 million budget for university research lost $565,000 — after Congress reduced the budget in December. ARPA’s total $82.7 million budget was cut by 76.7 million.

Because Reagan exempted the Strategic Defense Initiative, ARPA’s share of the cuts rose to 9.4 percent to compensate ‘With the SDI portions of R&D so large, other programs will actually have higher cuts,’ explained James Turner, a Defense Department spokesman.

The SDI budget is $2.8 billion, $241,934 of which goes to the program’s software systems research office. Despite the exemption from Gramm-Rudman, the SDI research has not remained unscathed.

In December, Congress had cut the SDI funds that Reagan had requested by nearly 1 billion.

General defense research and development dropped from $1.9 billion to $0.9 billion, while Defense Department-sponsored general science and space research funds were cut from $400 million to $300 million. Total defense research, development, testing, and evaluation spending dropped $1.7 billion from its original $35.3 billion budget.

The Defense Department’s software initiatives lost $2.2 million, while advanced Ada development lost $442,000.

Nondefense cuts. Nondefense areas did not suffer such severe cuts because the Gramm-Rudman law does not let the reductions be transferred from one agency to another. All nonexempt, nondefense programs lost 4.3 percent of their budgets.

The National Bureau of Standards science and technology research $1.2 billion budget was cut by $50 million. The Department of Energy’s general science and research funds dropped by $30 million from $700 million.

The National Aeronautics and Space Administration — already financially hurt by the loss of the space shuttle Challenger in a January postlaunch explosion and by the subsequent investigative costs and loss in revenue from suspended commercial shuttle missions — saw its $5.2 billion in funds cut by $220 million.

The National Science Foundation’s original $1.3 billion budget dropped by $58 million. Because the increase from 1985 to 1986 had been about three percent, the result is a slight decrease in available funds. ‘It means that every budget will be looked at a little harder,’ said Thomas Keenan, director of the NSF’s software systems science office.

‘The 4.3 percent is a nominal change. We can live with that,’ he said: ‘The great concern is next year,’ Keenan added, when the across-the-board cuts are predicted to be 8.8 percent. ‘That becomes serious.’

Museum features intelligent machines

The Museum of Science in Boston has announced plans for the first national museum exhibit devoted to a comprehensive exploration of the effect of artificial intelligence and robotics on society and the workplace.

Known as ‘The Age of the Intelligent Machines,’ the exhibit will begin a three-year tour of the US early next year, starting from Boston. An overseas tour is being considered.

Displays will include demonstrations of robotics, machine vision, speech technology, computer-aided instruction, expert systems, natural language technology, game playing, advanced graphics, and music composition.

Collaborating with the Boston museum in the exhibit are the science museums in Chicago; Los Angeles; Philadelphia; St. Paul, Minnesota; Columbus, Ohio; Charlotte, North Carolina; and Fort Worth, Texas.
‘Almost’ not enough for SDI software, Parnas maintains

Paul A. Myers

Software for the Strategic Defense Initiative has a fundamental problem — its capabilities cannot be completely tested against its operational requirements — David L. Parnas maintained in a debate at Stanford University.

“We don’t know what it is supposed to do, and we don’t have any way of verifying if it does what we think it should. If we don’t trust our system, then we have to continue with our other preparations,” Parnas, a professor of software engineering at the University of Victoria in British Columbia, Canada, told the 500 people attending the December 19 panel discussion. The debate on the antiballistic missile defense research program was sponsored by the Stanford University Department of Computer Science and the Palo Alto, California, chapter of Computer Professionals for Social Responsibility.

Speaking in support of the software aspects of the SDI research project was Richard Lipton, a computer science professor at Princeton University and a member of the Eastport Study Group, the Defense Department advisory board on SDI computing issues.

Lipton countered that there are ways to organize the programming during the design phase to reduce the number of critical problems to a level that starts to provide great confidence in the system's capabilities.

Parnas, a noted authority on defense software, expanded on the arguments he originally made last June in his resignation letter from the Eastport Group. At that time, he argued that SDI software could never be perfect and would be impossible to completely test in any case. (See the Soft News reports in the November 1985 and January 1986 issues of IEEE Software for details.)

Verification problem. At the Stanford session, however, Parnas conceded that the software does not have to be perfect and that no fundamental law keeping the SDI system from working. Rather, he stressed the improbability of verifying that the system will do what it is supposed to do.

Developing a large body of error-free code of the size required by the SDI project is virtually impossible, Parnas argued. It might be possible to have a 10-million-line system with 100,000 errors that would still “work” in some essential sense — “however, you would have to pick your errors very carefully, and that’s hard to do,” he asserted.

“There is something fundamentally different about software,” Parnas observed. Software always seems to be the unreliable part of the system where errors continue to be found after implementation. Even after thorough testing, it is hard to eliminate the chance of catastrophic failure from software.

He said that his experience with military real-time software showed that reliability comes from corrections made during repeated operational use. Although a high degree of fault tolerance may be achieved, that does not necessarily decrease the risk of catastrophic failure. In the case of SDI software, “almost right is not good enough,” Parnas emphasized.

“No, these are not limits on what computers can do, but rather limits on what human beings can do. We have lots of experience with building software, and we know what human beings can do — we know that they cannot build software that is correct before it is put in use,” Parnas concluded.

Flexible architecture. In contrast, Lipton said that it may be possible to organize a strategic defense so it isn’t a single integrated system, but rather a collection of systems of sufficient independence and diversity to achieve goals of resiliency and robustness that characterize other distributed computing systems successfully operating today.

Lipton stressed the advantages of an open architecture featuring relatively independent mission elements and computational structures that preserve locality, allowing autonomous action by local subunits organized into battle groups. Diversity in design and implementation would characterize the subunits.

Among the advantages of this type of architecture is a degree of independent testability allowing use of statistical inference techniques similar to those used to assess the reliability of offensive weapons systems, Lipton added.

Lipton's comments parallel arguments made by the Eastport Group. Its recent report on SDI computing issues stated, “The unique characteristic of a strategic defense system, the one that could make it difficult to test to the point of full confidence, is the extent of presumed coordination among the parts of the system.”

The report adds that several approaches to this testing problem are promising. These include more use of advice-only information transfers and extensive use of simulation so the operational code and algorithms could be tested under very large numbers of battle variations.

Another approach favored by the Eastport Group is simulation for in-line testing of a deployed system with simulated data going to and from the sensor and weapons platforms.

Because the aim of the current program is to research solutions posed by a defense system's complex requirements, Lipton warned that people must be very careful at this stage about claiming that something cannot be done. Proofs of impossibility are hard to come by in mathematics and computer science, he observed. On the other hand, it is very difficult to predict the nature of future technological advances.

Data-sharing risks. In rebuttal, Parnas said the complex informational demands of responding to a missile attack will put tremendous pressures on system designers to move away from independent and autonomous computing systems back to a more centrally coordinated defensive system that relies on widely shared data — data that must be communicated. Globally based data systems accordingly risk partial or global failure, Parnas argued.

Compared to other large software systems, such as telephone switching systems and the Apollo missions to the Moon, that are often offered as examples of the feasibility of successful implementations by SDI proponents, Parnas observed that the SDI system must contend with an intelligent adversary intent on defeating the system.

Other systems are contending against a statistically predictable natural environment, but any SDI system will be involved in a battle of strategy and tactics between intelligent opponents.

This contest is not analogous to the reliability measurements of the physical world that describe essentially continuous functions as being within a certain percentage over a period of time or number of trials. “We don’t rate chess players as being 97-percent effective,” he said, emphasizing the win/lose nature of a nuclear defense system.

Correction

A line of text was inadvertently dropped in the January issue. The last lines on p. 62 should have read "Systems programs that are designed independently of one another often interact according to some standard pattern of usage. For example, after an editor modifies a file, the file is often passed to a text formatter." We regret the error.
Software makes first comet flyby possible

David Dunham,
Computer Sciences Corp.

An international caravan of spacecraft — from the Soviet Union, Europe, and Japan — will fly by Halley's Comet between March 6 and 14, returning detailed scientific data about the famous every-76-years visitor. Several satellites, such as the US's Pioneer Venus orbiter, will also take breaks from their primary missions to examine the comet as it gets within their ranges.

But on September 11, 1985 — six months before the Halley flybys — the International Cometary Explorer spacecraft flew through the tail of Comet Giacobini-Zinner, becoming the first manmade object to encounter a comet. On March 27, the same US spacecraft will measure the solar wind upstream of Halley's Comet.

That September flyby would not have been possible were it not for National Aeronautics and Space Administration's Goddard Mission Analysis System, a software system used to compute spacecraft trajectories, that was running on IBM equipment NASA was phasing out at the time.

Change in mission. ICE, originally called the Third International Sun-Earth Explorer (ISEE-3), was initially placed in a very different trajectory than the one required for the comet flyby. The original path was a halo orbit around the L1 libration point, one of five such points in space where the gravity of the Earth and the Sun balance each other.

From this vantage point about 1.5 million km from Earth, ISEE-3 measured the solar wind outside the influence of the Earth's magnetosphere for four years after its 1978 launch.

Although the ISEE-3 mission's original plan did not call for a comet flyby, Robert Farquhar, the spacecraft's flight director at NASA's Goddard Space Flight Center in Maryland, had envisioned a comet flyby and had successfully lobbied for a relatively large fuel supply on ISEE-3 so it could maneuver in case of such a flyby.

When plans for a US mission to Halley's Comet evaporated in 1982, Farquhar convinced project scientists and NASA officials that they should extend the ISEE-3 mission to include an exploration of the distant tail of Earth's magnetosphere and a visit to Comet Giacobini-Zinner.

This required an extremely complex series of propulsive maneuvers and five lunar swingbys. Each close pass to the Moon drastically altered the spacecraft's trajectory. ISEE-3 scientists used the Goddard Mission Analysis System to make the extensive targeting calculations to define the new trajectory.

Difficult calculations. In early 1982, Farquhar's trajectory calculation team — Dan Muhonen and Leonard Church of the Goddard center and David Dunham, Sylvia Davis, and David Folta of Computer Sciences Corp. — were confronted with a difficult two-point...

Programming space probes in flight

Redirecting the ISEE-3 satellite from its solar studies and reprogramming the rechristened ICE to rendezvous with Comet Giacobini-Zinner was a difficult task for scientists at the National Aeronautics and Space Administration. But such programming is becoming almost routine for NASA's space probes.

The TDRS communications and tracking satellite launched last year by a space shuttle went into the wrong orbit, but ground crews were able to program the satellite to fire its small orbit-adjustment rockets to slowly nudge it where it belonged. The IRAS infrared astronomy satellite was reprogrammed several times to let it continue mapping the heavens for several months after it exceeded its life expectancy and began heating up. The heat buildup, caused when its coolant ran out, interfered with IRAS's infrared sensors, just as light does with an optical telescope.

But the Voyager 2 mission to the outer planets is perhaps the most ambitious example of in-flight programming. Launched in 1977, the probe flew by Jupiter in 1979, Saturn in 1981, and Uranus earlier this year. Its next stop is Neptune in 1989. For each flyby, mission scientists at the Jet Propulsion Laboratory in Pasadena, California, have had to adjust the probe from afar.

In its encounters with the three giant gas planets, Voyager 2 has discovered several new rings and many new moons. In the Uranus encounter, the probe found about a dozen rings and 10 new moons. Scientists knew of only nine rings and five moons before the encounter — and had no real details of their composition or structure.

Two billion miles from Earth, with a communications time lag of nearly three hours, Voyager 2 travelled about 40,000 mph as it passed Uranus and the planet's many rings and moons. The great distance meant the spacecraft could send back only 21.6K of data per second, compared to 115.2K when it was at Jupiter and 44.8K when at Saturn.

Before the Uranus encounter, the spacecraft used to send about 500 million bits of data for each picture (800 pixels wide by 800 pixels deep by eight brightness levels). But mission scientists reprogrammed Voyager 2 to send back only the difference between the pixel brightness values, cutting down the number of bits needed so much that the probe could send twice as many pictures as before — despite its slower send rate.

To further complicate transmission, the probe's primary receiver failed six years ago, forcing mission scientists to use a backup receiver that had a bandwidth (±96 Hz) a 1000 times narrower than the primary's. Backup programs have loaded into the probe in case the backup receiver fails so Voyager 2 can continue its mission. (The craft has had no problems sending data to Earth.)

The amount of sunlight at Uranus for the spacecraft's cameras is only a fourth of that available at Saturn — a four-hundredth that at Earth — forcing Voyager 2 to compensate for motion so its cameras could have longer exposure time for the photos in the darker environment and so it could adjust the camera's focus during very fast flybys of the some of the moons.

To allow the extra exposure time, most of the craft's equipment was turned off so their slight vibrations wouldn't produce blurry images. The biggest challenge was to make the probe aim its cameras properly as it sped by the small inner moon Miranda from a distance of only 18,000 miles. Because Voyager 2's arc-shaped flyby meant that its relative position to Miranda was constantly changing, the probe's computers had to continually move the probe itself so the camera wouldn't "see" the motion. The turning speeds required surpassed the on-board computer's programmed limits, so mission scientists reprogrammed them to override the original overrides.

For the 1989 flyby of Neptune, mission scientists will have to conduct similar long-distance reprogramming — this time from three billion miles away.
boundary-value problem: ISEE-3 was then in a known orbit around 1.1 and needed to reach the one point on the ecliptic plane (on which most planets orbit) that Comet Giacobini-Zinner's orbit would intersect and that could be reached by a spacecraft leaving Earth via a low-energy transfer orbit.

In most points of its trajectory, the spacecraft — rechristened ICE — was subject to strong gravitational perturbations by the Sun and Moon. No simplifying approximations could be made in the trajectory calculations, and all targeting calculations had to be performed by numerically integrating the trajectory many times using a realistic full-force model.

The Goddard Mission Analysis System performed these trajectory calculations first on IBM S/360-95 and S/360-75 computers and later on IBM 4341 and NAS 8040 machines, using both MVT and MVS operating systems.

It took about five months in mid-1982 to come up with the basic trajectory. At the time, the project was using older, oversubscribed machines. Downtimes for unscheduled maintenance occasionally disrupted the work. To compensate, the mission scientists were given high-priority job classes and weekend programmer support so they could perform the extensive targeting calculations and scans needed to compute trajectories with many different combinations of lunar swings and propulsive maneuvers.

In late 1982, they optimized the trajectory to get a practical low fuel use during the maneuvers. In 1983, they updated the targeting and planning for future maneuvers after each successive lunar swingby and propulsive maneuver. Each swingby and propulsive maneuver introduced trajectory errors that had to be corrected with small trim maneuvers. Fortunately, the transition of the original software to Goddard's newer computers in 1983 and early 1984 went smoothly.

**Analysis system.** The GMAS mission analysis system used has four major entities: an executive module to manage data, a system library with its utility load modules (including trajectory propagators, targeters, and other mission analysis tools), dynamic arrays as the primary communications links between load modules, and the GMAS automatic sequence — created by a structured language developed for the system — that lets users control the other components.

An automatic sequence contains the names of executive modules, utility load modules, data for the modules, and logical operation directives similar to Fortran logical statements.

GMAS allows flexibility and software extensibility. Users can create personal load modules to compute special data or perform other functions and can add or replace load modules anywhere in a sequence of executable load modules.

For the ICE mission, the maneuver team created special modules to compute such things as special lunar targeting parameters. In some of the GMAS sequences, a flag was set when the spacecraft achieved escape velocity too close to the Moon during an intermediate lunar swingby. The automatic sequence corrected the targeting whenever the flag was set.

Another module produced printer plots of the trajectory at the end of a targeting calculation. An Earth-centered ecliptic rotating system, with the x-axis pointing away from the Sun, was used to portray the orbits because this best estimated the success of a calculation for a given set of lunar swingbys.

---

**Panel considers simulation in the desktop age**

**Galen Gruman, Assistant Editor**

The wide availability of relatively powerful computers on people's desktops will force computer simulationists to rethink how they deal with simulation users, members of a panel concluded at the Society for Computer Simulation's annual conference in San Diego January 23-25.

In 1978, the available microcomputers — Apple IIs with 48K RAM running Basic — kept serious applications such as simulation in the province of researchers and companies using mainframes like VAX 11/780s. Managers would ask for a simulation and would several weeks later be given, oracle-like, a mound of green-and-white perforated paper to divine.

Now, the same manager probably has an IBM PC-XT with 512K RAM and 10M-byte hard disk, and is running complex spreadsheets (which the panelists agreed are simply financial simulation models). Indeed, today's microcomputers are comparable in power to yesterday's shared mainframes, argued Claude Barnett of Walla Walla College in Washington state. Microcomputer CPUs are slower than VAX CPUs, but a micro gives the user faster turnaround because its resources aren't shared, he said.

"Big, fast computers aren't going away — they seem to be used for bigger problems," offered Glen Johnson of CACI in La Jolla, California. As the power of desktop computers increases, so does their use for increasingly complex tasks. "The word 'microcomputer' is obsolete — it's just a desktop computer," asserted George Marr of Sorrento Valley Associates in San Diego, to the agreement of his fellow panelists and the audience.

The shift in power to the user means many of the traditional, moderately sized jobs can be done by a company in-house. The simulationist becomes a consultant rather than a vendor. Simulationists will have to adapt to this growing I'll-do-it-myself attitude by letting users create and implement the models, said Ray Swartz of Berkeley Decision/Systems in Santa Cruz, California.

"Users would not program the models, but would control them through an interactive interface. "Let users make their own models within the framework that we provide,"' he suggested. "Make a packaged simulation available to PC markets," Johnson echoed.

The tools a simulationist needs — lots of text processing, basic calculations, and a database manager — are supported by desktop computers, Johnson pointed out. But software for non-programmers who want to run simulations on their micros isn't. The challenge for simulationists, he said, is to write generic simulation models with adjustable parameters that are controlled by a simple, friendly user interface.

The problem of the simulation's user interface dominated the exchanges between the panelists and the audience. "People don't know what to do when A>B appears on the screen. That's not part of our cultural experience," exclaimed moderator Lance Leventhal of Emulative Systems when asked if the stress on desktop computer interface software was "just a transition from holding your hand on a mainframe to holding your hand on a micro."

Most agreed that graphical menus, input-checking menus, and natural language interfaces were needed before users could run simulation models that they didn't program. However, when one member of the audience said that user-friendly interfaces don't exist for the small simulation programs available for desktop computers, another argued...
that “most people who do simulations are not ignorant.”

Still, as another participant pointed out, the success of spreadsheet software proves that the user can be ignorant about programming and still effectively use a computer if he knows the subject he is simulating. It also proves how quickly the user can supplant the model programmer for what become routine simulations, he said.

However, making simulation software available to the engineering and scientific communities is more difficult than acknowledging its absence or designing good user interfaces. “Where are you going to get an entrepreneur to invest $10 million, where are you going to sell it?” Marr asked. The question to be addressed is “Who are you designing for? The simulationist? The business user? Who?” challenged a member of the audience. No one had an answer.

Supercomputers look beyond Fortran

Ware Myers, Contributing Editor

At the San Diego Supercomputing Center, Fortran will no longer be the only programming language. “We intend to have Pascal, C, and Lisp compilers available for our users,” said Wayne Pfeiffer, the center’s manager of user services. At present, however, compilers for those languages have drawbacks.

“A Pascal compiler is available for our Cray X-MP/48, but it is just now being made capable of vectorizing Pascal source code,” Pfeiffer explained in an interview. However, neither C nor Pascal compilers produce code as efficient for scientific computation as Fortran — and Lisp packages do not produce vectorized code.

The Cray X-MP series operates in two modes: scalar and vector. Scalar refers to the processing of one number at a time, as traditional von Neumann computers do. Vector refers to processing an ordered set of numbers, or a vector, through a series of computing stages, or a pipeline. The Cray X-MP/48 operates very fast in scalar mode, since its clock cycle is 9.5 ns, but in vector mode it is capable of 210 MFLOPS in each of its four parallel processors.

To get this vector performance, however, the source code must be vectorized. That means there must be ordered sets of numbers in the problem being run, a compiler must be able to identify these sets and apply them to the pipeline, or, alternatively, a programmer must manually identify them.

Fortran dominant. In spite of all the languages available for the Cray, almost all the time is spent computing in Fortran, Pfeiffer said. One reason is that Fortran is the traditional language for scientific and engineering computations and has the largest library of programs. Moreover, Fortran has been the only language with a compiler that automatically vectorized its source code.

“We will be using Fortran for the foreseeable future,” he added.

Still, one of the problems the supercomputer center faces is that Fortran is no longer the primary teaching language in the universities. “They have moved away from Fortran to Pascal, C, and Lisp,” said Bob Leary, a member of the user services group. “For scientific computation they probably teach Pascal. We may face the problem of trying to reach an audience that may not be familiar with Fortran.”

C niches. C can fill many niches that Fortran can’t, Pfeiffer said. One of these niches is artificial intelligence, Leary added. “Many expert-system shells are written in C. Also, some of the expert-system delivery programs are in C, which, at least on scalar machines, is much more efficient than Lisp. Some expert systems, after they have been developed in Lisp, are reprogrammed in languages such as C or even Fortran,” he explained.

“We would very much like to have C on our machine,” Pfeiffer said. “One of the immediate applications would be to run SMP [the Symbolic Mathematics Package], which is written in C and is intended for a Unix environment.”

Cray Research is supplying Unix as the operating system with the Cray 2, the first of which was delivered to Lawrence Livermore National Laboratory last year. “However, we have a strong commitment to CTSS, the Cray Time Sharing System, another interactive system which we believe to be more friendly to the user. Just as on the DEC VAX, where there are VMS and Unix, on Cray there is CTSS and Unix. We would like to implement Unix as a shell under CTSS. In this environment most of the features of Unix, but not all, would be available.

“Some features having to do with character passing would be extremely inefficient on a Cray under Unix, and one would not want to implement them,” Pfeiffer noted. “Doing so would seriously degrade performance. They won’t be implemented under the Unix Cray system either.”

Lisp performance. Some Lisp packages run on the Cray X-MP/48. “But they cannot be vectorized — they run only in scalar mode,” Leary said. “Even in this mode, however, the Cray X-MP/48 is undoubtedly the fastest Lisp engine available today.” Of course, its vector processing capability isn’t used when Lisp programs are run, thus wasting much of the power of the machine.

Leary cited three examples of Cray scalar performance using Portable Standard Lisp reported by Lucid Corp’s Richard Gabriel (then of Stanford University) and examples of performance differences between Crays and VAXs by J. Wayne Anderson of the Los Alamos National Laboratory. On a theorem-proving program, performance on a VAX 11/780 was 12.3 times slower than the scalar Cray. On a Symbolics 3600, it ran 5.5 times slower. On an expert system, the figures were 8.0 and 5.0. On the symbolic mathematics program Reduce, VAX performance was 15.0 times slower.

The enormous memory of the Cray X-MP/48 — eight million 64-bit words — is also an advantage in running artificial-intelligence programs, many of which make heavy use of memory.

“The downside of the supercomputer is that it is not necessarily a good development environment for expert systems,” Leary concluded. “That is true of any time-shared system. The preferred development environment now is the dedicated Lisp machine.”

But, for running very large artificial intelligence programs, the Cray’s memory and speed produce results that would take too long on other computers.

NSF funds fifth center

The National Science Foundation selected a Pennsylvania consortium to be the fifth National Advanced Scientific Computing Center. The mid-January decision added the already established supercomputer center to the four NSF-sponsored centers. The Pittsburgh-based center will use a Cray X-MP supercomputer.

The other four centers are in San Diego; Princeton, New Jersey; Urbana, Illinois; and Ithaca, New York.