



1992 Gordon Bell Prize Winners

The fifth annual Gordon Bell Prize competition saw a clash of cultures. The majority of the entries ran dedicated parallel processors, while the big winner used a heterogeneous collection of machines scattered around the US.

Alan H. Karp, Hewlett-Packard

Ken Miura, Fujitsu America

Horst Simon, NASA Ames Research Center

The Gordon Bell Prize recognizes significant achievements in the application of supercomputers to scientific and engineering problems. In 1992, prizes were offered in three categories:

- *Performance*, which recognizes those who solved a real problem in less elapsed time than anyone else.
- *Price/performance*, which encourages the development of cost-effective supercomputing.
- *Speedup*, which measures how effectively large numbers of processors are used.

No entries were received in the compiler-generated-speedup category.

Of the eight entries received, six used Intel parallel processors, one a heterogeneous set of machines scattered around the US, and one a machine built out of digital signal processors.

Gordon Bell, an independent consultant in Los Altos, California, is sponsoring \$2,000 in prizes each year for 10 years to promote practical parallel pro-

cessing research. This is the fifth year of the prize, which *Computer* magazine administrators. The winners were announced November 19 at the Supercomputing 92 conference in Minneapolis.

Results

Hisao Nakanishi and Vernon Rego of Purdue University and Vaidy Sunderam of Emory University received a check for \$1,000 in recognition of the outstanding price/performance they achieved on the simulation of polymer chains parallelized over a heterogeneous collection of machines from all over the US. This application does little floating point, so the judges had to rely on less traditional measures in selecting the winner. In their most cost-effective configuration, this team achieved 1 GIPS (billion instructions per second) per \$1 million — more than 150 times faster than sequential scalar code running on a Cray Y-MP.

Mark T. Jones and Paul E. Plassmann of Argonne National Laboratory submitted two applications that required

the solution of large, sparse linear systems of equations. They were awarded \$750 for running at more than 5 Gflops (billions of floating-point operations per second) on the Intel Touchstone Delta prototype at the California Institute of Technology. Their new approach to factoring such systems lets them solve for vortex configurations in superconductors and model the vibration of piezoelectric crystals.

Michael S. Warren of Los Alamos National Laboratory and John K. Salmon of Caltech received \$250 for running a simulation of almost 9 million gravitating stars at more than 5 Gflops on Caltech's Touchstone Delta machine. They parallelized a tree code, one of the most efficient sequential algorithms for large numbers of bodies.

In addition to these winners, two other teams were selected to present their work in a special session at Supercomputing 92.

Tom Cwik and Jean Patterson of the Jet Propulsion Laboratory at Caltech and David Scott of Intel solved an electromagnetic scattering problem that was much larger than the memory on the

Intel Touchstone Delta machine they used. Their out-of-core algorithm ran at more than 5 Gflops, almost as fast as in-core solvers ran last year.

A team from the Swiss Federal Institute of Technology—Anton Gunzinger, Urs Müller, Walter Scott, Bernhard Bäuml, Peter Kohler, Florian Müller-Plathe, Wilfred F. van Gunsteren, and Walter Gugenbühl—built a machine made up of 30 digital signal processors that outperformed a conventional supercomputer on a molecular *N*-body calculation. Their \$40,000 machine now holds the record for speed on this particular code.

The performance and speedup figures are somewhat lower than past years' spectacular numbers. We attribute this to the difficulty of the problems solved. Sparse matrices, out-of-core solvers, tree codes, and Monte Carlo simulations of the sort presented this year are notoriously difficult to parallelize, particularly on distributed-memory machines.

Price/performance winner

It is well known that while workstations provide very cost-effective computing, their actual utilization is low, often less than 1 percent. Thus, many people have thought about doing large-scale computations with the unused cycles of a multitude of these machines. The next step would be to supplement this power with time on conventional supercomputers and massively parallel processors. It's a good idea, but the complex logistics limit its applicability. While we still don't have a general-purpose system that makes it easy to use large numbers of workstations for parallel processing, the Purdue/Emory team built a system that successfully handles stochastic simulation problems.

Many phenomena in nature are driven by random processes. Crystal growth and Brownian motion are two simple examples. In each case, the macroscopic features result from many random events. Clearly, these events are not totally random. Rather, the statistics depend on the physics of the problem. For example, a small particle undergoing Brownian motion will drift down a density gradient because, on average, it receives more pushes from the high-density side.

While workstations provide very cost-effective computing, their utilization is low.

Nakanishi, Rego, and Sunderam's winning entry in this category examines the statistical mechanics of polymer solutions. Important unanswered questions await more computer power. For example, what shape does a large molecule take on when it is confined in something like a porous sandstone? How do polymers move through a membrane? Exactly what controls the rate at which a polymer diffuses through a gel under the influence of an electric field? The first question is of interest to the oil industry; the second, to pharmaceutical companies; the third, to those analyzing DNA.

There are also important theoretical issues, since the statistical properties of long-chain polymers are closely related to critical phenomena such as the liquid-to-gas phase transition that occurs at the boiling point. The importance of this work was highlighted when the application of critical phenomena theory to polymers was cited in the recent award of a Nobel Prize to Pierre-Gilles de Gennes.

Critical phenomena can be represented by so-called universal constants. These quantities are said to be universal because the same behavior characterizes quite different systems. For example, universal constants do not depend on the type of atom making up the polymer; rather, they are different for linear and branched molecules.

The most interesting problems arise where there is more than one length scale involved. Polymer growth in a porous medium is a system that can have two independent length scales. The question studied by this team is whether or not the constants derived from such things as the straight-line distance between the ends of the polymer in a porous medium are affected by disorder in its environment. Previous studies have been inconclusive.

If some fraction of the cube's volume is filled, there is some probability that an opening spans the cube. If the fill-fraction is high, the probability of there

being such an opening is low; if the fill-fraction is low, the probability that the cube will have several, wide openings is high. A "critical fraction" exists for which there is often a single, connected region that contains all these spanning paths. Away from the critical fraction, the connected regions have a typical size that sets a length scale that is independent of the polymer length. The key question is, what happens as we approach the critical fraction while increasing the length of the polymer? Do the two diverging lengths compete and result in different universal constants, whether or not the constants themselves change at the critical fraction?

The modeling process has two steps. First, a lattice must be created that is as filled as possible while having openings that span the lattice in all three dimensions. Such a lattice corresponds to the critical point. For each point in the lattice, starting in one corner and proceeding in a regular manner, a point is filled on the basis of a uniformly distributed random number. Connectivity information is kept as the lattice is processed, a nontrivial exercise. Once the entire cube has been traversed, the connectivity information is used to determine whether there are paths connecting the faces. If not, the process is repeated. The probability of filling a point in the lattice needs to be set at just the critical value to simulate the competition of two diverging lengths. If this probability is set too high, there will be no spanning path through the lattice; if it is set too low, the "opening" will be too wide. In either case, the system will not be critical.

The next step models all ways the polymer can be made to grow on this lattice. The technique used is a "self-avoiding walk." Starting from an open point on the surface of the lattice, the program tries to extend the polymer to a neighboring point. If this point is not filled, the path length does not exceed the specified maximum length, and the point has not been previously occupied by the polymer, the process continues from that point. Otherwise, it backs up and tries another point. This depth-first search will eventually back up to the starting point once all possible paths of a specified length have been found.

The longer the paths taken, the better the estimates of the universal constants. Unfortunately, the computer time increases exponentially with the path

length. One way to speed things up is to sample the paths rather than follow them all. Unfortunately, there is no way to be sure the sampling is unbiased, and biased samples have led to errors in deciding on the universality question. Hence, complete searches are the only way to resolve the issue, and they are time-consuming. The runs reported here take more than 3 hours each on a Cray Y-MP, and thousands of runs are needed to gather enough statistics to answer the question of universality of the constants.

The parallel job is run under the Eclipse system, an execution environment for controlling tree-structured computations. The user input consists of a configuration file describing the machines to be used, a set of parameters that describe the physical system being modeled, and random-number seeds. A master process distributes subsets of the parameters to different subtrees in a way that ensures the runs are independent of each other. Each node uses one random-number seed to generate a partially filled lattice and another to start the self-avoiding walks. Every so often, as determined by a parameter set by the user, each child sends its accumulated statistics to its parent. The parent combines the results from its own computation with those of its children and passes the data up the line. The original process does no computation of its own but does produce the output from the run. The statistics are combined in parallel because a single processor could not keep up and would be a serial bottleneck.

When using a large number of machines, especially when others have access to the power switch, it is important to use a system that doesn't have to restart the job if a processor fails. The Eclipse system handles failures in a very straightforward way. Each parent is responsible for doing the computations of its children, with one exception. Leaf nodes — those with no children of their own — that fail are simply ignored. If a node with children fails, its parent will reallocate the work to other subtrees. In the worst case, one processor could end up doing all the work, but in practice the method works quite well.

The problem run uses a cubic lattice with 35 points on a side and periodic boundary conditions. This lattice is small enough to be manageable while large

The most cost-effective configuration, 16 IBM RS/6000s, ran at 3 GIPS per \$1 million.

enough that its finite size doesn't significantly affect the results. Walks of 35 steps were chosen because methods based on sampling indicate a change in behavior for paths of around 30 steps. The actual runs require 2,000 or more different lattice fillings to get sufficiently accurate results.

Timings were reported for 11 different configurations ranging from 16 Sun Sparcstations at Purdue to 192 nodes (48 IBM RS/6000s, 80 Sun Sparcs, and 64 nodes of an Intel iPSC/860) scattered around the US. The smallest configuration ran 495 different batches of walks on a single lattice in about 2 hours compared to more than 3 hours on a Cray Y-MP. The largest configuration ran 384 batches in less than 10 minutes compared to more than 3 hours on the Cray. The most cost-effective configuration, 16 IBM RS/6000s, ran at 3 GIPS per \$1 million, more than 150 times the price/performance of the Cray. To be fair, we should note that the application does not vectorize and most of the computation is on integers. On the other hand, the actual cost of the parallel runs, except for the iPSC time, is zero, since the only cycles used would have gone to waste.

What about the scientific results? Are the constants universal? We will have to wait to find out; only preliminary results are available. Because the problem is controversial, the investigators do not want to report their conclusions until they are certain. Runs even larger than those reported here are under way.

Speedup winner

High-temperature, or type II, superconductors differ from type I superconductors in several ways. One important difference is their response to an external magnetic field. A type I superconductor excludes the magnetic field lines for any field strength up to a critical

value above which the superconductivity vanishes. Type II superconductors have a mixed state that exists when the magnetic field is in a particular range. In this mixed state, the material is superconducting, but the magnetic field penetrates the material.

What happens is that vortex currents form around the penetrating field lines. These currents generate a magnetic field that cancels the external field, shielding the bulk of the material. Understanding this behavior is important if high-temperature superconductors are to be used in high-field applications like magnetic-levitation trains and magnetic-resonance-imaging devices.

Piezoelectric crystals vibrate when an electric current is applied to them. They can be found in computers, cellular phones, and many other devices. It is important that the crystal be designed to vibrate at a single frequency as operating conditions vary over a relatively wide range. For example, circuit designers would like to build a device as small as possible that vibrates in a single mode over a wide range of temperatures. As it happens, the desired mode is near the middle of the spectrum of possible crystal vibration modes, which makes it hard to isolate it from other modes at nearby frequencies.

What could these two applications possibly have in common? It turns out that they share a property that appears in many problems: The most computationally efficient algorithm for solving them involves repeated solutions of a large, sparse system of linear equations.

The type II superconductors are modeled as alternating layers of superconducting and insulating material. The vortices move around until they are in a configuration that minimizes their free (total minus internal) energy. This process is simulated by discretizing the model in three dimensions and applying a standard optimization procedure to find the solution that minimizes the computed free energy. Of the optimization methods used on serial computers, the inexact Newton method converges fastest — about 20 iterations versus several thousand for competing methods. The problem is that the inexact Newton method requires the solution of a large, sparse linear system at each iteration.

The piezoelectric crystal problem is solved in two steps using a finite-element discretization. First, a nonlinear, static, thermal stress problem is solved

to determine the equilibrium shape of the crystal at some temperature. Next, the amplitudes of the vibration modes near the desired frequency are found by solving a large eigenvalue problem. The most time-consuming part of the calculation is the solution of a large, sparse linear system of equations. For many problems, more than 90 percent of the time is spent solving the linear system. Most of the remaining time is spent evaluating the matrix elements.

While general-purpose sparse solvers have been available on scalar and vector processors for quite some time, coding an efficient routine for a distributed-memory parallel processor has been a challenge. The Argonne National Laboratory team of Jones and Plassmann has produced such a package for symmetric matrices. Their package, soon to be released into the public domain under the name BlockSolve, is even portable; it runs on several different parallel machines, as well as on workstations connected over a network.

Typical linear systems arising from the solution of problems such as those submitted with this entry have 100,000 to 2 million unknowns. The saving factor is that only a few of the coefficients are nonzero, usually fewer than 1 percent. For example, in the problems submitted there are fewer than 2 billion nonzeros, compared to some 4 trillion total matrix entries.

Except for cases where the matrix has some special structure, direct solution methods such as Gaussian elimination are impractical; the solution process changes many of the zeros to nonzeros, a process called "fill-in." Hence, iterative methods must be used. The most popular method for symmetric matrices can be viewed as minimizing the square of the residual. More specifically, if we want the solution to $Ax = b$, where A is the matrix, b the known right-hand side, and x the desired solution vector, we minimize the size of the square of the residual, r^2 , where $r = Ax - b$.

Several minimization algorithms can be used for this quadratic function. Steepest descent moves from the current estimate of the solution in the same direction a marble would roll in a valley the shape of the function being minimized. Steepest descent does not work well when the valley is long and narrow, since the estimate of the solution tends to bounce back and forth across the valley floor. The conjugate gradient

For many problems, more than 90 percent of the time is spent solving the linear system.

method works better. It too moves downhill in the general direction of steepest descent, but the direction is made perpendicular to the direction of all previous moves.

The conjugate gradient method is very easy to parallelize. Each iteration requires a multiplication of the sparse matrix A times a dense vector r , the current residual. If the matrix A is partitioned among the processors by blocks of rows, this operation is perfectly parallel. In addition, each iteration requires several vector dot products. These require a global summation that can also be done efficiently on parallel processors.

But there is a catch; (there always is). The convergence rate of the conjugate gradient method depends on the shape of the valley. Convergence is fast if the valley is circular, slow if the valley is long and thin. The shape of the valley is related to the ratio of the largest to the smallest eigenvalue of the matrix A , the "condition number" of the matrix. The larger the condition number, the slower the convergence. Most matrices of interest have very large condition numbers, making the unmodified conjugate gradient method impractical most of the time.

There is a sparse matrix with the smallest possible condition number, the identity matrix. If we could find a matrix M such that $MA = I$, convergence would be very fast. Of course, M is the inverse of A , and finding M is equivalent to solving the original problem. What if we could easily construct a matrix M that approximates the inverse of A ? Then MA would be close to the identity matrix, the condition number would be small, and the conjugate gradient method applied to $MAx = Mb$ would converge quickly. We call this process "preconditioning." The better M approximates the inverse of A , the faster the convergence.

The preconditioned conjugate gradient method is a practical method for

solving large, sparse, symmetric systems of equations. Unfortunately, the preconditioning step is often hard to parallelize. We can see why by looking at the most popular preconditioner, incomplete factorization. Here we construct M by performing Gaussian elimination on A , ignoring all fill-in. (If A_{ij} is zero, then we set the ij element of the decomposition to zero.) MA and Mb are then computed from the back substitution of the decomposition. The problem for parallel processors is the recursive nature of the back substitution.

A technique for vectorizing iterative methods, called red-black ordering, has been used for many years for matrices with regular structure. Here, we update the value of every second point in a grid — the red points — using the current values of their neighbors — the black points. Next, we update the black points using the latest values for the red points. This split iteration is perfectly parallel, although it increases the number of iterations somewhat. On parallel processors we can use several colors.

The novelty in the Argonne team's work is that they use graph-coloring theory to implement a multicolor scheme for irregular grids such as those arising from the finite-element method. Contrary to their worst fears, the use of many colors does not increase the number of iterations dramatically. Since the communications cost on a parallel processor is proportional to the number of colors needed to achieve parallelism, they have developed a heuristic for coloring the unknowns into a small number of independent sets. They also use the fact that several rows of the matrix often have their zeros in the same locations to improve the individual nodes' performance of the matrix vector product.

The performance obtained with this code is impressive. Several models of type II superconductors that ran in tens of minutes would have taken tens of hours using other methods on a Cray 2. Their computation ran at over 4 Gflops on an Intel Touchstone Delta prototype and achieved a speedup of between 350 and 500, depending on the particular problem being solved.

Performance winner

One of the simplest problems in physics is finding the motion of two bodies

under their mutual gravitational attraction. However, to attack such interesting problems as how galaxies formed in the early universe, how a galaxy evolves once it coalesces, and what happens when two galaxies collide or pass close to each other, we must be able to follow millions of objects. The difficulty in these calculations is not modeling the underlying physics, which is quite simple. Rather, the difficulty lies in the large number of computations.

The problem studied by the winners in this category is related to the ultimate fate of the universe. We know from measuring the velocity of distant galaxies that the universe is expanding. The rate of expansion must be decreasing because of the mutual attraction of the material. If this rate of decrease is large enough, the universe will eventually start to contract; if not, the universe will expand forever. Since we would have to wait millions of years to measure directly any change in the expansion rate, we rely on indirect measurements. One test is the amount of mass in the universe, which determines the overall force of attraction.

The amount of luminous matter is only some 10 percent of the amount needed to make the universe reverse its expansion. However, we know there is a substantial amount of nonluminous material. Some of it is in planets and some is in black holes; the rest, if there is any more, is in another form. One possibility comes from modeling the motion of stars in galaxies, which suggests that there is also a lot of dark matter in a nearly spherical halo around most galaxies. Models of these halos typically contain more than 100,000 particles.

Solving a problem this large is hard because the gravitational force extends to infinity. Hence, every particle affects the motion of every other particle. Using a naive algorithm to follow the motion of 1 million particles would require almost 1 trillion force evaluations per time step — 1 million seconds of computer time even if we could compute the force between two particles in 1 microsecond! Clearly, algorithmic improvements are needed.

A simple observation breaks the bottleneck. When we compute the gravitational force of the Sun on the Earth, we do not need to compute the force of every atom in the Sun separately. Instead, we treat all the particles as if they

To attack problems such as how galaxies formed, we must be able to follow millions of objects.

were part of an oblate spheroid and use only a few moments of the mass distribution. We can use the same trick in gravitational N -body calculations; a clump of particles far from the one we are computing the forces on can be treated as a single particle with some mass distribution. Using this approach means that the computer time increases only linearly with the number of particles N , or as $N \log N$, depending on how we do the clumping and how many moments we include in the force computation. In either case, we can follow a million particles with a reasonable amount of computer time per time step.

Clearly, a program to compute the forces in such a manner will be more complicated than one that implements the simple N^2 method, and it will be harder to parallelize. The most common approach is based on a tree. We start with a cube containing one of the particles in its initial position. We add a second particle and divide our cube into eight subcubes of equal size. If the two particles are in the same subcube, we continue the process, recursively dividing cubes until each particle is in a separate cube. Then we add another particle and subdivide until each of the particles is in a separate cube. When we have finished adding our 1 million particles, we will have a domain divided into a large number of cubes of varying size, each containing at most one particle.

Now we can compute the force on each particle completely in parallel. For each particle, we look at each of the cubes in the first level — those that are one-eighth the volume of the region being studied. If any one of these cubes is far enough away from the particle that the gravitational force of the particles it contains can be approximated accurately with a few moments of its mass distribution, these moments are used to compute the force on our particle. Since this force approximates the effect of all the particles in the cube, we

do not need to progress further down this branch of the tree. If the cube is not far enough away, the eight cubes it contains are examined. This depth-first search continues until the effect of all the particles has been included.

Several important problems must be solved to get good parallelism. First, we can't afford to store all the particle data on each node; the limited amount of node memory would restrict us to rather small problems. This means we can't keep a copy of the entire tree on each node, either. We must also worry about load balancing, making sure that one processor doesn't have to work much more than any of the others. Finally, if we speed up the force calculation a lot, the tree-building process, which must be repeated for every time step, will become a bottleneck; so we will have to parallelize this part too.

Warren and Salmon used domain decomposition in which each processor is responsible for all the particles in some volume. Clearly, some processors will end up doing most of the work if each is given an equal volume of space. In fact, since the work to compute the

Now Available !

Real-Time Systems Design and Analysis

by Phillip A. Laplante

This monograph provides a comprehensive guide to the practical design and analysis of real-time systems. The book is self-contained and covers all aspects of real-time software design including computer architecture, operating systems, programming languages, software engineering, and systems integration. It contains many examples, illustrations, and exercises as well as practical tools which the software engineer or student can apply today in the field or laboratory.

360 pages. September 1992. Hardcover.
ISBN 0-7803-0402-0. Catalog # 3107-04
— \$49.95 Members \$40.00 —

Call 1-800-CS-BOOKS
for your copy today

 IEEE COMPUTER SOCIETY

IEEE COMPUTER SOCIETY PRESS

Sixth annual Gordon Bell Prize

The entry deadline for the sixth annual Gordon Bell Prize is May 31, 1993. Finalists will be asked to present their results at a special session during the Supercomputing 93 conference. The rules for the prize will not change in 1993. The judges will award prizes from four categories, plus any honorable mentions merited. Complete rules will be available later this year.

force on a particle depends on the particle's neighbors, we can't even give each processor an equal number of particles. Instead, we can use the amount of time it took to compute the force on each particle during the last time step to help us assign an equal amount of work to each processor.

Let's look at some time step other than the first. (Special procedures are needed to get started.) Each processor contains the data for some particles, including the time it took to compute the forces on the last time step. We start with all the processors in the same group. First, we pick some axis, say the x -axis, and find the value of x that divides the particles into two parts that took equal amounts of time on the last step. The processors holding the particles on the left side are put into one group; those holding particles on the right go into the other group. Any processors holding particles that would be on the wrong side send the corresponding data to a processor on the other side. This splitting is continued recursively until each group contains only one processor.

Next, each processor constructs the part of the tree that the particles it contains will need. Since the particles owned by each processor all lie in a limited volume, the processor will need only limited information from processors holding distant particles. The problem is complicated because the processors hold particles in rectangular volumes, which are often long and thin, while the tree is based on cubes. The fact that the part of the tree needed locally can be computed efficiently is a key part of the parallelization. Once the needed part of

the tree has been built, the force on each particle can be computed.

It would seem that all these manipulations would affect overall performance. They do, but not as much as we might think. For example, assembly-coded force calculations run at 22 Mflops per processor on the Intel Touchstone Delta prototype. The average performance per processor, including all overhead, communications delay, and load imbalance, is 5.4 Gflops divided by 512, the number of processors used—more than 10 Mflops per processor. The calculation submitted ran for almost 17 hours and produced more than 4 Gbytes of data on disk to be used for further analysis.

Other finalists

Two additional entries were recognized as finalists. Tom Cwik and Jean Patterson of the Jet Propulsion Laboratory and David Scott of Intel submitted an electromagnetic scattering calculation run on the Intel Touchstone Delta prototype. These problems require the solution of very large, dense systems of linear equations. Until now, the only dense solvers available on parallel machines required that the entire matrix fit in memory. This group produced a code that solved larger problems by storing parts of the matrix in the concurrent file system. The problem they submitted computed the scattering from a conducting sphere and required solving a system of almost 50,000 equations, taking up some 38 Gbytes for storing the matrix. The problem took almost 20 hours at a sustained computation rate of 5 Gflops. A sphere was chosen because the analytic solution can be used to check the accuracy of the calculation. Since the matrices are dense, the computing requirements depend only on the number of unknowns, not on the details of the model.

The remaining finalists, Gunzinger et al. from the Swiss Federal Institute of Technology in Zurich, built a distributed-memory parallel processor using 30 digital signal processing chips. They submitted applications in neural nets and molecular dynamics. In particular, with just \$40,000 in parts, they solved a molecular dynamics calculation of 1,000 particles faster than the previous record holder, an NEC SX-3 supercomputer. They also trained neural networks us-

ing back propagation at a rate of 870 Mflops, compared to 780 Mflops on an NEC SX-3.

Other entries

Three other entries were received. All represented excellent work, and the selection of finalists was, as always, difficult.

A group from ONERA in France submitted the solution of an electromagnetic scattering problem for axially symmetric bodies that ran at more than 3 Gflops on a 128-node Intel iPSC/860 and more than 7.5 Gflops on an Intel Touchstone Delta prototype. Axisymmetry allows them to use fast Fourier transforms to solve the linear system. An important achievement is that they can solve real problems in a very short time, only 27 seconds for a complex matrix of order 50,000.

Enrique Castro-Leon of Intel and Elizabeth Yip of Boeing also submitted the solution of an electromagnetic scattering problem. They assume that the scatterer has one plane of symmetry. This assumption leads to a special structure in the coefficient matrix. Their out-of-core solver runs at up to 4.9 Gflops on an Intel iPSC/860; a full problem, including setup and computing the scattered radiation, runs at more than 1 Gflops.

Majdi Baddourah and Olaf Storaasli of the NASA Langley Research Center submitted a method for assembling finite-element matrices that runs significantly faster than conventional methods on parallel processors. In one case, a computation that took 70 seconds on one processor of a Cray Y-MP using the best serial algorithm took only 3 seconds on 512 nodes of the Intel Touchstone Delta prototype. ■

The judges

Alan H. Karp, who chaired the judging committee, is a member of the senior technical staff at Hewlett-Packard Laboratories in Palo Alto, California.

Ken Miura is vice president of the Computational Research Department at the Fujitsu America facility in San Jose, California.

Horst Simon is a staff member at Computer Sciences Corp. working under contract at the NASA Ames Research Center in Moffett Field, California.