1990 GORDON BELL PRIZE WINNERS

This year, a multicomputer system broke the Gflop barrier and several entrants showed the utility of using several RISC workstations.

JACK DONGARRA
ALAN H. KARP
KEN MIURA
HORST SIMON
Bell Prize Judges

The Gordon Bell Prize recognizes significant achievements in the application of supercomputers to scientific and engineering problems. In 1990, two 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, and
- compiler parallelization, which encourages the development of smart parallelizing compilers.

We received 12 entries — seven were considered for the performance prize, two for price/performance, and two for compiler parallelization. One enterprising entrant, knowing how flexible the rules are, entered in a nonexistent category: speedup. We awarded prizes in the price/performance and compiler-parallelization categories. Honorable mentions were named in the performance and compiler-parallelization categories.

Gordon Bell, chief scientist at Stardent Computer and a consultant for Me Unlimited, both in Sunnyvale, Calif., is sponsoring two $1,000 prizes each year for 10 years to promote practical parallel-processing research. This is the fourth year of the prize, which *IEEE Software* has administered. The winners were announced Feb. 27 at the Computer Society's Compcon conference in San Francisco.

RESULTS

A check for $1,000 went to G. Al Geist and G. Malcolm Stocks of Oak Ridge National Laboratory. Beniamino Ginatempo of the University of Messina, Italy, and William A. Shelton of the US Naval Research Laboratory for winning the price/performance award. They computed the electronic structure of a high-temperature superconductor on a 128-node Intel iPSC/860 at a price/performance of more than 0.8 Gflops per million dollars. In addition, they solved the same problem on a network of 11 IBM RS/6000 workstations at 0.7 Gflops per million dollars. (Four of the least expensive of these machines performed at 1.9 Gflops per million dollars.) The 2.5-Gflop rate for the run on the Intel machine is the first report of the Gflop barrier being broken by a multicomputer running a real application.

Gary Sabot, Lisa Tennies, Alex Vasilevsky of Thinking Machines and Richard Shapiro of the United Technologies received $500 for their work in compiler parallelization. They used a Fortran-77-to-Fortran-90 conversion package to parallelize a grid-generation program used to solve partial differential equations. They achieved a speedup of 1,900 and ran at more than 1.5 Gflops on a Connection Machine with 2,048 floating-point processors.

Two honorable mention prizes of $250 each also were awarded.

The winners of last year's performance award — Mark Bromley, Steven Heller, Cliff Lasser, Bob Lordi, Tim McNerney, Jacek Myczkowski, Irshad Mufti, Guy L. Steele, Jr., and Alex Vasilevsky of Thinking Machines and Doug McCowan of Mobil Research — submitted the same

0740-7459/91/0500/0092/$01.00 © IEEE

application code that won last year. This year, they increased the size of the problem and used an improved compiler and library. These changes alone let them improve their job's performance by more than 2.5 times, to 14 Gflops.

The other honorable mention went to Eran Gabber, Amir Averbuch, and Amiram Yehudai of Tel Aviv University for a parallelizing Pascal compiler. They successfully parallelized 14 programs, including one of the programs that won the first Gordon Bell Prize, achieving speedups of as much as 25 on 25 processors of a Sequent Symmetry computer.

The judges felt that, although they did not win prizes, two entries were worthy of note. A seismic-migration problem submitted by George Almasi of the IBM T.J. Watson Research Center and Tim McLuckie, Jean Bell, Aaron Gordon, and David Hale of the Colorado School of Mines ran at more than 1.5 Gflops per million dollars on a network of 4 IBM RS/6000s. David Strip of Sandia National Laboratories and Michael S. Karasick of IBM Research figured out how to do solid modeling on a Connection Machine more than 35 times faster than they could on an IBM 3090 vector processor.

The price/performance category presented a new problem this year. In the past, all entries in this category ran on machines costing millions of dollars. Adding the cost of software, a display, and keyboard to the cost of one of these machines does not have a major effect on the price. But this year, three entrants used collections of workstations, and each made different assumptions about the equipment needed to run the jobs. For example, the Colorado School of Mines team used a price for the RS/6000 Model 320 nearly twice that used by the Oak Ridge team. We adjusted both team's price/performance figures — Colorado's up substantially and Oak Ridge's down slightly — to reflect the current published prices.

PRICE/PERFORMANCE WINNER

Theoretical materials scientists have traditionally studied either highly ordered materials like crystals or highly disordered materials like glasses. Recently, however, there has been a great deal of experimental work done on what are called substitutionally disordered materials. The one that has received the most press attention is the family of high-temperature superconductors, but metallic alloys and materials with metal-insulator or magnetic-phase transitions also fall into this category.

These materials are messy, and interpreting the experimental data is difficult. Five years after the discovery of high-temperature superconductors, researchers still do not understand why they lose electrical resistance. Thus, theoreticians have been called on to help. While semiempirical studies can be useful, it is widely felt that scientists will understand these materials only if they can compute their observed properties from first principles. In this case, "first principles" means relativistic quantum electrodynamics.

It is not yet possible to solve the Schrödinger equation for a solid; there are just too many degrees of freedom. Instead, scientists use methods that let them average over the bulk properties, leaving a manageable number of states.

Three entrants used workstation groups, and each made different assumptions about the equipment needed to run the jobs.

One approach is based on an observation arising from density-functional theory. It states that a fundamental property of a solid, its ground state energy, depends uniquely on the electron density. Furthermore, the electron density can be computed by solving for the motion of a single electron moving through an electric field that is an average of the motion of all the nuclei and other electrons. This solution must include the effect that the moving electron has on all the other electrons — the solution must be self-consistent. In this approach, you approximate the complex, many-electron effects by treating the background electrons as a continuous gas and using the density of that gas at any point as an approximation to the true electron density — by making the local density approximation.

The local density approximation does not work for substitutionally disordered materials. Instead, you must use a method like the coherent potential approximation. You approximate the effects the disordered crystal has on the electronic structure by the effects of some "effective" scatterer. An ordered, periodic array of effective scatterers gives, in some sense, the best approximation to the disordered structure of the material. The coherent potential approximation computes the best effective scatterer that you can get using only average properties of the nuclei.

Finding the effective scatterer involves the self-consistent solution of a set of integral equations. The procedure starts with a guess about the electron distribution. Next, you perform many numerical quadratures. You use any deviations from self-consistency to update the electron distribution. You continue the procedure until it converges.

You calculate the integrals representing the solution over the fundamental modes of the crystal; these modes depend on the locations of all the nuclei. If the algorithm were parallelized at this level, each processor would be responsi-

ble for a subset of the crystal. Because these integrals involve data from the entire crystal, this scheme would require a lot of communication among processors. Fortunately, the function being integrated over the fundamental modes involves an integral over the energy, which can be done separately at each atomic site.

The integration over energy requires between 200 and 1,000 energy evaluations at each site to determine the charge distribution for the next iteration. Each energy evaluation involves the iterative solution of the coherent potential approximation equations. Because there is a great deal of computation and little communication, this approach is well-suited for parallel processing, even on a loosely connected set of independent processors.

In its winning entry, the Oak Ridge/Messina/Naval Lab team implemented the parallelization with a master/slave approach. One processor is responsible for reading the problem description, the location of the input files, and managing the overall iteration. It balances load by assigning tasks to processors in order of decreasing difficulty.

> In its entry, the winning team implemented the parallelization with a master/slave approach.

An interesting feature of the code is its portability; it has versions for serial, shared-memory, and distributed-memory computers. The multicomputers can be moderately closely coupled systems like hypercubes or a loosely connected collection of workstations. The input file determines whether multitasking or message passing will be used. The total program takes 16,000 lines of Fortran and contains 127 subroutines. Of these, only about 20 are explicitly involved in the parallelism.

The team presented results for the simplest of the Perovskite superconductors, a family that includes several of the other high-temperature superconductors. This barium-bismuth system is of interest because it has only five simple cubic sublattices, it has a cubic symmetry that reduces the amount of computation, and it is closely related to the more complicated systems involving lanthanum and yttrium that have received a lot of attention.

Most of the time in this calculation goes into constructing a 160×160, dense, complex matrix (12 percent) and then inverting it (78 percent). These matrices are constructed and inverted independently on each node hundreds of times per iteration. The matrix construction runs at more than 24 Mflops per i860 node, 3 Gflops aggregate, while the inversion is done at 21 Mflops, 2.6 Gflops aggregate. Each processor of a Cray Y-MP does these tasks at 300 Mflops, which translates into a 2.4-Gflop aggregate rate.

A complete calculation involving almost 4×10^{13} floating-point operations took 4.5 hours on a 128-node Intel iPSC/860, a computational rate of 2.5 Gflops. Because the machine has a list price of $3 million, this performance translates into a price/performance of more than 0.8 Gflop per 1 million dollars. This rate includes all the start-up time and load imbalance.

This same program was ported to a network of IBM RS/6000s. A single RS/6000 Model 320 with a list price of $8,500 ran at 16.7 Mflops, which corresponds to 2.1 Gflops per million dollars. Although this price/performance is impressive, it would have taken about a month to complete one model. However, four of these bottom-of-the-line machines could finish the job in about a week at a price performance of more than 1.9 Gflops per million dollars.

Because neither configuration could complete the full problem in a reasonable amount of time, the team made a run on all the RS/6000s available at Oak Ridge. This set of seven Model 530s and four Model 320s ran at a sustained rate of 226 Mflops and a price/performance of more than 0.7 Gflops per million dollars. The speedup on this application is nearly linear in the number of processors. Therefore, if there had been 11 Model 320s at Oak Ridge, these figures would have been 170 Mflops and a price/performance of about 1.8 Gflops per million dollars.

COMPILER-PARALLELIZATION WINNER

As engineers have improved their models, they have found it increasingly necessary to work with complicated shapes. Those who want to use finite-difference or finite-element methods must construct grids that match the complicated boundaries associated with realistic models. Unfortunately, carelessly generated grids result in solutions with lots of error. A great deal of effort has gone into generating grids that do a good job of following the objects being modeled and having desirable numerical properties.

Generating a good grid is not trivial. Consider the simple problem of modeling the flow over a ramp. A grid consisting of uniformly spaced lines in the vertical and horizontal directions is unlikely to be satisfactory because the grid intersections will not necessarily fall on the ramp, which will make it difficult to handle the boundary conditions. Furthermore, the ends of the ramp are likely to be places needing

higher resolution than the regions on either side of the ramp. A better approach might be to produce a grid that is locally perpendicular to the surface. While you can represent the boundary accurately this way, the grid can become highly nonuniform. The sudden changes in grid spacing and the odd shapes of the grid cells can seriously degrade the numerical accuracy of the partial-differential-equation solver.

Numerical methods are often rated on how quickly the error in the solution decreases as the grid is refined. A good method with a good grid will reduce the error to one fourth when the spacing is halved. But even a good method will reduce the error only linearly as the spacing is refined if the grid is poor. If the grid is very bad, the convergence can be sublinear.

While it is difficult for a person to design a good grid even for something as simple as a ramp, it is virtually impossible to build a good grid for the kinds of surfaces engineers are interested in, things like the space shuttle or the Rocky Mountains. Thus, some sort of automatic tool is needed.

A grid-generation program is such a tool. It produces a grid that conforms to the shape being modeled and retains the good numerical properties of the discretization. Such a grid has a spacing that changes smoothly and has grid lines that cross at nearly right angles. The result is a grid that conforms to the shape being studied and with grid points lying on smooth curves.

Conceptually, the grid-generation algorithm warps the surface into the unit cube and places a uniform grid either around this cube if you are modeling the exterior or inside it if you are interested in the interior. The differential-equation solver is then applied to this simple domain. To visualize the results, you then transform back into the original coordinates.

This warping is a coordinate transformation determined by solving a system of partial differential equations. The solution of these equations gives the coordinates of the intersections of the new grid lines. There are many ways to affect the transformation because the properties of a good grid, like skew and relative size of neighboring grid cells, are specified so inexactly.

The various grid-generation approaches differ primarily in their choice of the equations used to generate the grid. The method used in the Thinking Machines/United Technologies team's winning entry starts with an arbitrary grid that follows the boundary. A system of second-order, nonlinear differential equations that represents the coordinates of the new grid as functions of the coordinates of the old grid is used to compute the transformation. There is one differential equation for each coordinate direction (two for the 2D problem submitted), each involving all combinations of second derivatives. The coefficients of the second derivatives involve only first derivatives of the new coordinates with respect to the original ones. It is these coefficients that make the problem nonlinear.

The solution of these equations is necessarily an iterative process. First, the differential equations are discretized using standard central-difference approximations for the first and second derivatives. You can solve the resulting system of nonlinear, algebraic equations with a variety of methods, many of which do not parallelize well.

The model submitted ran on a data-parallel Connection Machine, a CM-2G. This single-instruction, multiple-data computer works well on algorithms that involve lots of independent data elements. The team selected one such algorithm, a relaxation method using a Jacobi-like iteration, for the computations. This method works by guessing a solution, evaluating the discretized differential equation to compute the residual, and setting the new value of each grid coordinate to the old value plus the computed residual multiplied by a relaxation parameter. The team used a relaxation parameter because the approximations oscillate around the solution as the iteration proceeds.

While there are methods that converge in fewer iterations, they are more complicated, less parallel, or involve more floating-point operations per iteration. Key to the judges accepting this entry is the fact that Jacobi-like iterations are commonly used, even on sequential, scalar computers.

> As engineers as improved their models, they find it increasingly necessary to work with complicated shapes.

The entrants took a grid-generation program written in Fortran-77, containing no compiler directives. This program was passed through the KAP/F77-F90 preprocessor from Kuck & Associates. The preprocessor converted many of the nested loops in the original program into Fortran-90 array constructs. For this program, the sophisticated dependence analysis done by the KAP preprocessor was not needed; the important loops were very simple. The resulting Fortran-90 program was compiled with the CM Fortran compiler, which applies conventional vectorization optimizations to parallelize the code. The two-stage approach avoided the difficult question of whether the user or the compiler has determined the parallelism of a program written with Fortran-90 array constructs.

One difficulty the judges had with this entry was determining the meaning of speedup on a SIMD machine. Should we count each of the 65,536 one-bit processors as a separate machine or only the 2,048 floating-point chips?

How can you make a run on one processor of a SIMD machine? Do we let the problem increase in size as the number of processors increases, or do we consider only fixed-size speedup?

The original submission based its speed-up measurement on the time of the original Fortran-77 program running on the CM-2 front-end machine, a Sun-4 with a Weitek floating-point chip. Although this floating-point chip is similar to the one used on the CM, we questioned the validity of the reported speedup of 4,900, especially because the version of the CM Fortran compiler used treated the machine as 2,048 nodes and not 65,536 processors. Clearly, the floating-point performance of the Weitek chip on the Sun is not simply related to that on the CM-2.

The judges asked the entrants if they could run the job on one of the nodes. They did this by changing one environment variable to fool the system into thinking that the machine consisted of just one node containing a single, 64-bit floating-point unit and four megabytes of memory. (They were surprised when this approach worked because the compiler-testing strategy had not covered this serial configuration of the CM.) Then they ran the largest problem that could fit in the memory connected to this node.

The fundamental unit of computation was the number of grid points processed per second. Because this quantity is relatively insensitive to the problem size, we felt it was reasonable to compare the 128 × 128 grid on one floating-point unit to the 8,192 × 4,096 grid computed on the full machine. This method resulted in a speedup of 1,900. Equally impressive was the 2.3-Gflop computation rate.

THE JUDGES

Jack Dongarra is a professor of computer science at the University of Tennessee and on the staff of the Oak Ridge (Tenn.) National Laboratory. Ken Miura is a staff member at the Fujitsu America facility in San Jose, Calif. Alan Karp, who chaired the judging committee, is a senior staff member at the IBM Palo Alto (Calif.) Scientific Center. Horst Simon is a staff member at Computer Sciences Corp. working under contract at NASA's Ames Research Center in Moffett Field, Calif.

HONORABLE MENTIONS

The first of the honorable mentions went to the team from Thinking Machines and Mobil Research that won the performance prize for 1989. The team members improved the performance of the application they submitted last year by a factor of 2.5 by throwing away last year's low-level hand-generated code and applying improved compiler technology to the original Fortran-90 source code. Many of these compiler improvements were produced by the same Thinking Machines compiler team that won this year's prize for compiler parallelization!

The 14 Gflops they reported is truly remarkable, as is their price/performance of about 1.4 Gflops per million dollars. The CM-2G is the most expensive machine to achieve more than 1 Gflop per million dollars on a real problem. Because the problem submitted is identical to the one that won last year's performance prize, the description that follows concentrates on the changes made to the software.

The CM-2G is a data-parallel, SIMD computer containing 65,536 one-bit processors, eight gigabytes of memory, and, optionally, 2,048 64-bit floating-point processors. Data parallelism is used to program the CM-2. Basically, the programmer acts as if each data element, say a grid point in a finite-difference solution, is processed by a separate processor having its own private memory.

One hardware change is important: the increased memory. In general, the larger the problem, the higher the performance. The main reason for this behavior is that communication between points assigned to a single floating-point unit is faster than between those assigned to different ones. The four-fold increase in the memory between the CM-2 used last year and the CM-2G used this year let the entrants run a larger problem.

The primary change made in the software is the compiler's view of the target machine. Each 64-bit floating-point unit and its associated four megabytes of memory is called a node and acts much like a vector processor. The CM is viewed as having 2,048 64-bit vector-processor nodes instead of a collection of 65,536 one-bit processors. Loop spreading and strip mining are used to map problems that have more points than 2,048 times the vector length (currently four) onto the machine.

This change is reflected in how CM Fortran stores floating-point numbers. The previous release of the compiler stored all 32 bits of each floating-point number on a single, bit-serial processor. Unfortunately, this approach does not mesh well with the Weitek floating-point chips.

On a given machine cycle, each CM processor can access one bit from its local memory. Thus, it takes 32 memory cycles to access a floating-point number. But the floating-point unit expects to get a 32-bit number on every cycle. Thinking Machine engineers handled this problem by inserting a hardware transposer between the CM processors and the floating-point unit. Over a period of 32 cycles, the transposer takes in 32 floating-point numbers, one from each of the 32 one-bit processors. It then feeds these numbers, one per cycle, into the floating-point unit. The process is re-

versed when the output is to be stored. While this approach does let the Weitek chip's pipelining be used, it impedes the machine's overall performance.

The newest release of the CM Fortran compiler stores the data slicewise. In this storage scheme, each bit of a floating-point number is stored on a different one-bit processor. Now, on one cycle, 32 bits of the number can be sent directly to the floating-point unit, bypassing the transposer array. (Two cycles are used for 64-bit numbers.) Slicewise data storage also changes the programmer's view because a problem with as few as 2,048 grid points will use all 2,048 64-bit processors, replacing the view where 65,536 points were needed to use the whole machine.

This new view of the hardware was extended to the communications. The older library was set up so each one-bit processor would pass a single piece of data to a specific neighbor. Because all the data moved in the same direction on each step, it would take four communications steps to distribute data in a 2D grid. The newly microcoded communications primitives let nodes, as opposed to processors, communicate on the 2D grid. There are enough wires from each node to let them pass data in all four directions in one communications cycle.

Another improvement to the compiler is how it handles fundamental grid operations. The finite-difference discretization of a partial differential equation involves several grid points. The particular choice of grid points for a given discretization is called a stencil. In two dimensions, the discretization that combines each point with its north, south, east, and west neighbors is called the five-point stencil. Other, more complicated stencils are also used.

Last year, Thinking Machines introduced a stencil library that used highly tuned microcode to improve the performance of the communications of some commonly used stencil calculations. This year, a compiler was written that automatically recognizes any kind of stencil, even an irregular one. Basically, any CM Fortran statement of the form $A = C_1 * S(X) + C_2 * S(X) + \ldots$ can be compiled as a stencil, as long as A and X are arrays and S is any of the functions that shift data in the grid.

The stencil compiler uses several special coding tricks to improve performance. For example, at the start of the computation, all the data needed by the node will be copied into the node's local memory. Furthermore, you can use optimizations to better use the floating-point chip's registers and pipelines. Thinking Machines has made substantial effort to minimize the movement of data between the floating-point unit and the node's memory.

The other honorable mention is the first cash award made for work done in Pascal. The compiler this team from Tel Aviv University wrote is remarkable in several respects. The same compiler can be used for both shared-memory and distributed-memory machines. Furthermore, no directives are needed to help the compiler distribute the data among processors.

The input language, TAU-Pascal, includes only a few, minor extensions to Pascal. The Portable Parallelizing Pascal Compiler consists of a front end that takes a sequential, TAU-Pascal program and produces an explicitly parallel C code for the Virtual Machine for Multiprocessors software, which implements a set of parallel-processing functions for a diverse set of machines. The compiler will produce good parallel code for any machine that can run the virtual-machine software. The data partitioning and code generation are optimized for the target machine by consulting a table of parameters that specifies the relative cost of certain operations, like floating-point arithmetic and communication.

The compiler produces a parallel program that runs in single-program, multiple-data mode. This means that each parallel process executes the same lines of code on different parts of the data. Some parts of the program are executed by a subset of the processors by using the task ID to control statement execution. For example, program initialization and termination are handled by a single processor. The task ID is also used to split up work on an array among the processors.

The compiler does not use the most sophisticated dependence analysis to deter-

Continued on page 102

FIFTH ANNUAL BELL PRIZE

The deadline for the fifth annual Gordon Bell Prize is May 31, 1992. The deadline has been pushed back five months to accommodate a new schedule: The prize winners will now be announced at the Supercomputing conference held each year in November instead of at the Compcon conference held each February. The announcement location was changed to reflect the closer tie-in of the Supercomputing technical focus to that of the Bell Prize. Finalists will also be asked to present their results at a special session at the conference.

The rules for the prize will also change to add a new category to cover novel advances in computation in an attempt to reward algorithmic improvements applied to parallel processors. The other three categories will remain the same. The judges will award two prizes from the four categories plus any honorable mentions merited.

Complete rules will be available later this year.

Software companies *are* still hiring — but, like for their clients' purchasing, the decision-making process takes longer than before.

The software job market is much more competitive. With a perception of fewer jobs available, hiring companies face a buyer's market. Uncertainty has fueled a surge of résumés by software professionals looking for better positions, better pay, and more security.

The average software job seeker is now competing against a higher caliber of résumé. An East Coast executive observed that "there was a bottoming out in prospect quality about two and a half years ago. There was a dearth of good people to hire. Now I can't believe the quality of the people who are looking to change jobs." On the parts of the industry affected, he said, "I find more shifting of people in development environments for business use. They're affected more by business shifts. There seems to be more stability in the PC and Unix worlds."

For many software professionals, the recession and the competitive job market are reasons to stay put. It's better to keep what security you have than to put it at risk. The result appears to be an observable, if not measurable, increase in employee productivity. People who know they are staying put do more to make it work better. They're willing to put up with more flak and are willing to extend themselves to find a better way.

OPPORTUNITIES. Fortunately, this dark cloud definitely has silver linings. The recession provides many opportunities for software firms willing and able to capitalize on them.

One company president pointed out that "topics like reverse engineering are taking on more momentum. Companies are thinking less about new development and more about reusing what they've invested in before. It's a wonderful time to be in that business."

He cited some other opportunities: "The recession has definitely affected travel. Companies and other software vendors are holding back. This means that if you still send out your salespeople, you can have presence on site when your competitors don't. That's a way to show you care about the customer and get the sale.

"It's also an excellent time to grow. I've never seen more qualified people with résumés out. If you have the cash flow, now is the best time to build a business," he said. ◆

BELL PRIZE

1990 WINNERS

Continued from page 97

mine if a loop can be parallelized safely. A loop will be parallelized only if each iteration modifies a different array element or if the loop contains certain simple grid operations. The loop may also be parallelized if it contains certain reduction operations, like inner product, or specific access patterns on columns of matrices. The current version will not parallelize a loop nested in a previously parallelized loop.

The compiler supports only a limited set of data partitionings: replication (each process gets a complete, read-only copy), replication for data reduction (certain associative and commutative operations are allowed on the copies), row distribution (each process gets a contiguous block of rows), overlapped row distribution (each process gets a contiguous block of rows with some duplication between the processes owning neighboring blocks), interleaved row distribution (the rows are dealt out as in a card game), and interleaved column distribution. As limited as these patterns are, they are sufficient to parallelize a wide variety of programs.

The team submitted 14 programs, each run on three very different parallel systems: a Sequent Symmetry shared-memory machine with 26 processors; MOS, an experimental shared-memory system with eight NS32532 processors; and a network of eight T800 transputers. The application set is diverse, including computational kernels, like the solution of dense systems of linear equations by conjugate gradients and matrix multiplication, to such applications as a particle in cell simulation of electron beams and the simulation of 150 bodies attracted by gravitational forces. The judges worried about how realistic these programs were, but we were convinced when the team used the compiler to parallelize the Wave program that won the first year's Bell Prize. In all cases, the compiler generated code that ran at between 60-percent and 99.8-percent efficiency on all three parallel systems.

The judges run into a unique problem with this entry. As typically happens, we needed some questions answered before we felt comfortable awarding a prize. Unfortunately, just at the time we needed a quick response to meet our deadline, Tel Aviv was under attack from Iraqi missiles! Fortunately, someone from the Tel Aviv team managed to get out long enough to answer our e-mail questions.

OTHER ENTRIES. The Bell Prize is attracting entries in many areas, several of which we did not anticipate. An entry from Isabel Beichel and Francis Sullivan of the US National Institute of Standards and Technology demonstrated the first robust and efficient code for 3D triangulation, an important part of some molecular-dynamics simulations. Arjen K. Lenstra of Bellcore, Hendrik W. Lenstra of the University of California at Berkeley, Mark S. Manasse of the Digital Equipment Corp. Systems Research Center, and John M. Pollard submitted their widely publicized factorization of the ninth Fermat number, an integer with 155 digits.

The remaining entries were more along the lines we were expecting. Laurence Feldman of the US Air Force, Brian Dodd of Cray Research, and Nazareno L. Rapagnani of the University of Arizona submitted a calculation of the incompressible flow around an F-16 aircraft flying at Mach 1.8 that took only 30 seconds to complete. Peter Highnam, Andy Pierprzak, and Indranil Chakravarty of Schlumberger and Bill Moorhead, Charlie Liu, Biondo Biondi, and David Race of Thinking Machines did a seismic-migration problem on a CM-2. Weiqiang C. Liu, Pietro Rossi, and Mark Bromley of Thinking Machines and Anthony D. Kennedy, Robert Edwards, and Daan Sandee of Florida State University used a quantum-chromodynamics program running on a CM-2 to attack one of the "grand challenge" problems of computational science: Monte Carlo computations of lattice quantum chromodynamics with dynamical Wilson fermions. Philip Emeagwali, one of last year's winners, submitted his version of the National Corp. for Atmospheric Research shallow-water model running on a CM-2. ◆

> The Bell Prize is attracting entries in many areas, several of which we did not anticipate.