# A Parallel Computer Architecture for Continuous Simulation

**J.O. HAMBLEN,** Member, IEEE
**C.O. ALFORD,** Member, IEEE
Georgia Institute of Technology

**A parallel computer specifically designed for the solution or** ***ordinary differential equations*** **is described. The first version of the machine contains thirty-two processors, running in an asynchronous Multiple instruction multiple data (MIMD) mode, communicating with high speed parallel busses. Synchronization is accomplished by a microprogrammable communication controller. A number of processors have been designed and built for the machine. The processor types offer a wide variation in solution speed and accuracy. To permit easy comparisons with analog and hybrid systems, performance is measured by finding the highest frequency** ***since wave which can be integrated in real-time with an accuracy of*** **0.1 percent or higher. Using this performance measure the performance limit of the current machine is 2000 Hz. The structure is capable of solving systems described by differential equations up to order sixty-four at these performance limits.**

Manuscript received July 9, 1987.

IEEE Log No. 24684.

Authors' address: School of Electrical Engineering, Georgia Institute of Technology, Atlanta, GA 30332.

0018-9251/88/1100-0719 $1.00 © 1988 IEEE

## I. INTRODUCTION

The simulation of continuous dynamic systems requires the solution of ordinary differential equations with initial value conditions. Such systems occur in aerospace, mechanical, biological, electrical, and chemical systems. In the past, there have been four distinct approaches to computational machines for continuous system simulation; analog computers, hybrid computers, digital differential analyzers, and digital computers. A recent approach is to design a parallel digital structure which unites ideas from all four of these approaches. Such a system takes full advantage of recent developments in very large scale integration (VLSI) technology. This fifth approach is special purpose in the same sense that an analog computer is special purpose. The resulting computer is intended to solve those problems described by ordinary differential equations at an accuracy level comparable with reliable physical measurements.

## II. PARALLEL COMPUTER ARCHITECTURES

A medium size analog computer is capable of performing the integration of 40 state variables and the associated function generation. Using fourth-order Runge-Kutta integration, a digital computer requires 20,000 integration time steps per second to achieve the equivalent accuracy and speed. A typical state equation requires 250 operations per state variable per integration step. Thus a digital computer requires a sustained throughput in excess of 200 MFLOPS (millions of floating point operations per second) to match the performance of a typical analog computer [1].

Only the most recent models of multimillion dollar general purpose vector supercomputers, such as the CRAY X-MP, begin to approach this level of sustained performance on real simulation problems [2]. Thus economics provides the major impetus for a special purpose machine.

A number of commercially available parallel computers have appeared recently. Many of these general purpose computers are being used for simulation applications [3–11]. Special purpose commercial simulation machines are also available [12]. The floating point performance of these machines is not yet in the supercomputer class [2]. Utilizing VLSI components, many of these parallel computers offer price/performance advantages over vector supercomputers. The commercially available general purpose parallel computers typically have a small number of processors built using a general purpose microprocessor with a math coprocessor and a limited bandwidth interconnection network. The performance of these parallel machines cannot be improved by orders of magnitude without major modifications to their current architecture or technology.

## III. NUMERICAL CONSIDERATIONS

The classical solution method for ordinary differential equations with initial value conditions is shown in Fig. 1. For each integration time step it is necessary to perform function or derivative evaluations and numerical integration sequentially for all of the state variables [13–20]. The major differences are in the numerical integration methods used. Multiple function evaluations per time step are required by many integration methods. Many software packages allow the user to choose one of several common integration routines to obtain the best performance.

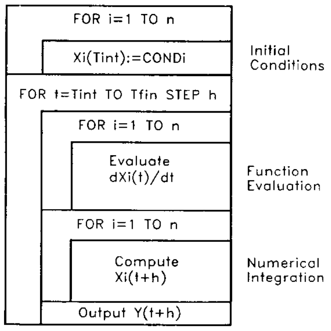


Fig. 1. Sequential solution of system equations.

An equivalent parallel digital solution method analogous to the operation of an analog computer, or a digital differential anayzer, is shown in Fig. 2 [21–26]. A theoretical linea speedup by a factor of $n$, the order of the system, is possible if $n$ processors are used. Additional

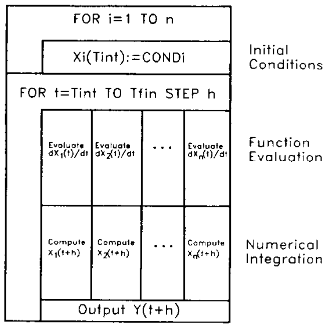


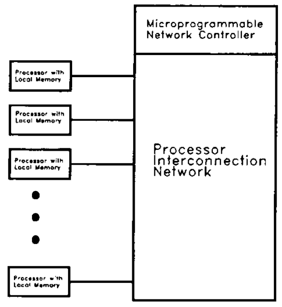
Fig. 2. Parallel solution of system equations.

parallelism can be found in the numerical integration methods; however, this parallelism, a factor of two to four, is small compared with $n$ [22, 27, 28]. New parallel solution methods have also been suggested [29–33]. Using classical methods, coupling in the system of equations means that data must be exchanged among processors every time a function evaluation is performed. This requires that an interprocessor connection network be provided. The time required for the transfer of data will reduce the speedup to a value less than $n$. For maximum speedup the processors must be connected by a high bandwidth interconnection network. The functional dependence of the differential equations determines the connection paths required. This implies that the interconnection network must be reconfigurable for high performance.

The solution of ordinary differential equations with initial value conditions is ideally suited for parallel processing. This class of problems exhibits an extremely high degree of parallelism. Many computations can be performed in a processor before it is necessary to exchange global data. The amount of global data or state variables that must be shared among processors is small in relation to local data and program size. These are the characteristics that must be present for efficient parallel processing utilizing a large number of processors.

## IV. SYSTEM ARCHITECTURE

Based on an analysis of these numerical considerations, the architecture shown in Fig. 3 was developed [23]. A number of asynchronous processors, each with local program and data memory, are connected


Microprogrammable Network Controller
Processor with Local Memory
Processor with Local Memory
Processor with Local Memory
Processor Interconnection Network
Processor with Local Memory
Fig. 3. Parallel computer architecture.

to a high speed interconnection network controlled by a microprogrammable communication controller. The communication controller is responsible for synchronizing the system and controlling the switch points in the interconnection network.

To demonstrate the utility of this architecture, an experimental prototype capable of supporting 32 processors was designed and built. The experimental prototype is intended to be used as a research tool. The prototype was used to obtain accurate performance data and to gain additional insight into implementation problems and limitations.

Each processor performs function evaluation and numerical integration on a subset, typically one or two, of the system state variables. Programs and data are maintained in the local processor memory. Every time a function evaluation is performed the new values of the state variables are transferred on the interconnection network. This is the minimum amount of information that must be exchanged among processors. Decomposition of the problem in this manner maintains fast interprocessor communication times. Further decomposition increases the parallelism at the expense of increased communication with a resulting decrease in performance. Processors with high speed analog-to-digital (A/D) and digital-to-analog (D/A) channels are used for analog inputs and outputs.

## V. NETWORK ARCHITECTURE

The interconnection network must be capable of parallel high speed data transfers among arbitrary processors. Clearly, a network which allows all processors to communicate directly to any other processors in parallel is desirable if it is economically viable. Networks, such as hypercubes, which require processor forwarding of data to support an arbitrary transfer are too slow to meet performance goals. Crossbar and Banyan networks are possible candidates for the interconnection network. These networks grow in complexity by an $O(n^2)$ when switch points, interconnection wiring, and control circuitry are taken into account [34]. Crossbar switching networks are nonblocking, require only one level of switching, and have a higher degree of symmetry making fabrication less difficult. A crossbar switching network was selected for use in the experimental prototype.

The physical size of a fully parallel switching network is prohibitive. Tri-state bidirectional data busses are used to reduce the size of the network by a factor of four. Thus a combination of space and time switching is required to transfer data. High speed microcode memory is used to enable and control the direction of each switch point in the network. This allows multiple destinations for a single packet of input data and use of different routing strategies. Typically, simulation problems require four or more time slots on the network for a complete data transfer.

Data is transferred on 16-bit busses in the experimental prototype. Each switch point uses two 74LS245 octal bus transceivers. A four by four switch matrix is implemented on a circuit board. Sixteen boards are required for the thirty-two processor prototype. Daisy chained ribbon cables run horizontally and vertically through the network to provide the large number of interconnections required.

The communication controller contains a high speed microprogrammable state machine as shown in Fig. 4. Each microinstruction controls a time slot on the network. Fields in the microcode specify the processors requiring input, the processors providing output data, and the switch configurations required in the network. The processor microcode fields control maskable comparators that signal when the selected processors are ready to transfer data. The controller hardware tests and sets four handshake lines on all processors in parallel. Pipelining is used to configure the switch points prior to the transfer of data. The data path through the network contains a single gate delay of 8 ns. The major communication path delay is a 100 ns signal rise time in the ribbon cable connections. The rise time results from the capacitance between signal and ground wires in the data cable. Custom VLSI circuits could be used to reduce the physical size of the network and increase the performance.

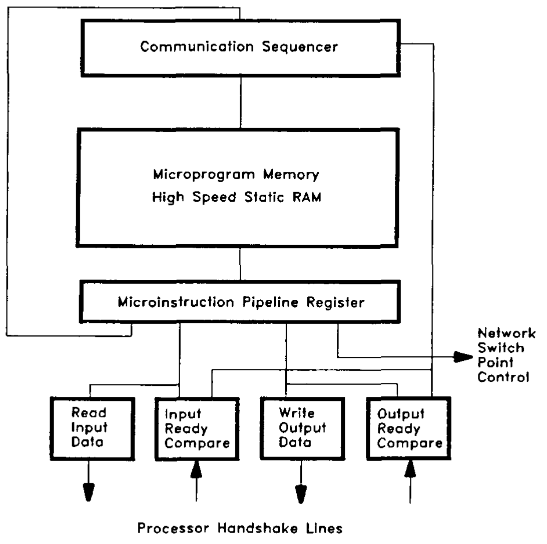


Fig. 4. Microprogrammable communication controller.

Processor programs must output and input data in an ordered sequence that coincides with the communication controller microprogram. The arrival of data via the network is used to synchronize the processors. The number and order of input and output variables is problem dependent and will vary from processor to processor. The network interface is buffered to allow processors to perform other operations while transfers are occurring.

## VI. PROCESSOR ARCHITECTURE

The system architecture is capable of supporting many types of processor modules. All that is required is a compatible interface to the switching network and an *IEEE Standard 796* card format [35, 36]. The standard interface to the network is a 16-bit parallel transistor-transistor logic (TTL) bidirectional port with four handshake lines. Five processor modules have been developed for the prototype computer. They include a general purpose microprocessor with a numeric coprocessor, a processor with high speed A/D and D/A converters, a high speed microprogrammed fixed point processor, and two high speed microprogrammed floating point processors.

The microprocessor based design uses an Intel SBC 86/12 processor with an 8087 numeric coprocessor. When this processor was selected it was anticipated that the next generation of VLSI floating point units would approach the performance goals for the machine. These processors served to provide floating point capability in the interim.

The high speed microprogrammed fixed point processor architecture is shown in Fig. 5. With current VLSI technology it is necessary to use several speedup techniques to attain the data rates required. These include microcoding of programs, separate program and data memories, and pipelining of both instructions and data.

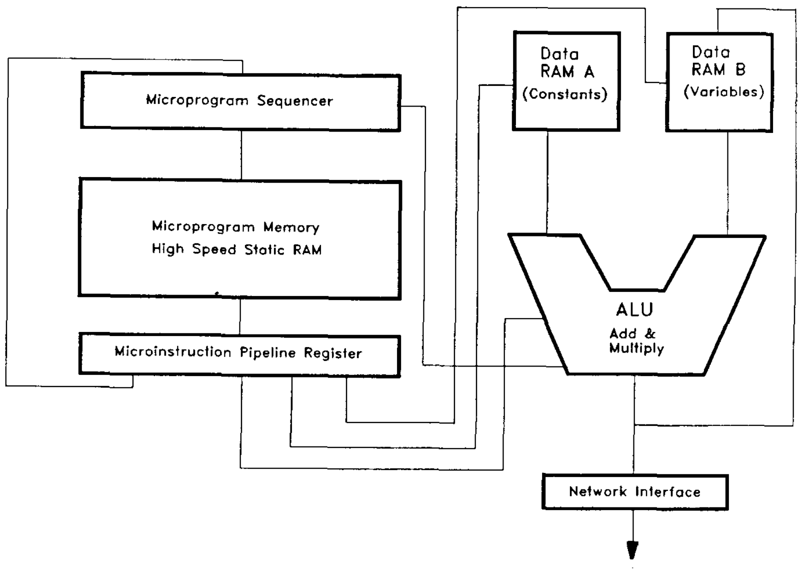


Fig. 5. Microprogrammable processor architecture.

Only microprogramming allows efficient control of all parallel operations supported by the hardware. Pipelining allows the overlapping of fetch and execute cycles. Separate program and data memories allow instruction and operand fetch operations to occur simultaneously. For high performnce, a high speed adder and hardware multiplier are required.

The fixed point processor performs two operand fetches, a multiply, a double precision addition, and a store in 250 ns. Implementation of the module requires 113 integrated circuits. Programmable logic arrays were used where appropriate. A TRW 1010J provides the multiply and accumulate operation and an AMD 2910 was used for the microprogram sequencer. Processor memory uses 50 ns static RAMS.

Two microprogrammed floating point processors have recently been designed and built. These processors use an architecture that is virtually identical to the fixed point unit. Data is maintained in *IEEE Standard 754* 32-bit floating point format [37]. Both processors use an AMD 2910 microprogram sequencer and have separate data and microprogram memories. The first design uses the Weitek 1032 floating point multiplier and 1033 floating point adder [38, 39]. The Weitek floating point ALU contains a six stage data pipeline. Filling and flushing of the internal pipeline slows down the processor and this demands special care when programming.

The second processor design uses the AMD 29325 floating point ALU and the AMD 29334 register file. After evaluating a prototype of both units, the AMD 29325 processor design has been selected for incorporation in the next version of the machine. The peak performance of a single AMD 29325 based processor is 10 MFLOPS [38]. New processor designs containing four to eight T800 floating point transputers [40] or the bipolar integrated technology 2110/2120 floating point ALUs [38] are being investigated.

## VII. BENCHMARKS

As part of the experimental program several simple continuous system simulation benchmarks have been implemented on the prototype machine described in this paper. Results obtained using the prototype were compared with traditional serial results to verify correct operation and to validate the parallel solution method.

To program these benchmarks on the prototype a number of software tools were developed. In the prototype all processor and control memories can be downloaded by a general purpose host computer. The host can also start, stop, reset, single step, and examine memory contents in all processors and the communication controller. These features are useful in multirun simulations. A compiler was written to generate the microcode for the communication sequencer. The input to this compiler is a simple language which describes the data transfers required between processors.

Additionally, the program in local memory of each processor must be developed. For the microprocessor based processor a compiler was used. High speed processor benchmark programs were written in microcode. Ultimately, a compiler for a continuous system simulation language could be developed for the machine which would generate all of the required code modules [23, 41, 42].

The benchmarks selected were a second-order linear system, the pilot ejection problem, PHYSBE, and a linear single axis autopilot [16, 23]. Speedups demonstrated on the prototype using the 8086/8087 microprocessor are shown in Fig. 6. On any parallel computer, linear speedup cannot be obtained unless processor communication time is zero. Based on program execution times, a more realistic model for the machine assumes a ten percent overhead from processor communication. This speedup, $0.9N$ can be obtained on systems of ODEs that produce equal processor computational loads.

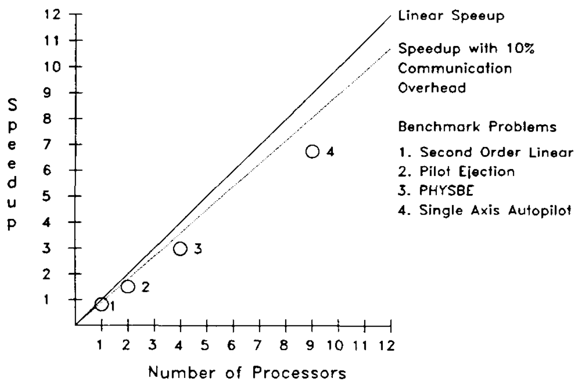


Fig. 6. Speedup obtained on benchmarks.

Performance below this level is due to unequal processor computational loads. Unbalanced processor execution times will arise in nonlinear ODE systems because of different function or derivative evaluation times.

High speed microprogrammed processors were also evaluated by finding the highest frequency sine wave which can be integrated in real-time with an accuracy of 0.1 percent. Using this benchmark the performance limit of the prototype machine is in excess of 2000 Hz.

## VIII. CONCLUSIONS

Using current technology, the prototype machine is capable of solving 64th-order ordinary differential equations at a solution bandwidth in excess of 1000 Hz. A special purpose machine built using parallel VLSI circuits offers the potential of mainframe performance levels at a hardware cost reduction of an order of magnitude or more. The architecture presented is capable of solving ordinary differential equations at speeds comparable with modern analog computers. Such a machine can serve as a replacement for hybrid systems and supercomputers in large real-time simulations.

Additional work is needed in the development of VLSI chips designed to support parallel architectures, the development of parallel compilers for continuous system simulation languages, and new integration methods designed for parallel computers.

## ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of R. Nabow, F. Rohling, L. McMillan, W. Tan, and other numerous students, in the design and construction of the prototype machine.

## REFERENCES

[1] Korn, G.A. and Vichnevetsky, R. (1976)
Analog/hybrid computation and digital simulation.
*IEEE Transactions on Computers, C-25*, 12 (Dec. 1976), 1312–1320.

[2] Dongarra, J.J. (1987)
Performance of various computers using standard linear equations software in a FORTRAN environment.
*Simulation, 49*, 2 (Aug. 1987), 51–62.

[3] Karplus, W.J. (1986)
Parallelism and pipelining: the road to more cost-effective scientific computing.
In *Multiprocessors and Array Processors; Proceedings of the Third Conference*, 1987, San Diego CA, SCSI, pp. 1–13.

[4] McBryan, O.A., and Van de Velde, E. (1986)
Hypercube algorithms and implementations.
*SIAM Journal of Scientific & Statistical Computations, 8*, 2 (Mar. 1987), 5227–87.

[5] Rattner, J. and Wojcik, A.S. (1985)
Concurrent scientific computing.
In *Proceedings of the American Federation of Information Processing Societies 1985 National Computer Conference*, 1985, Chicago IL, pp. 157–161.

[6] Test, J.A., Myszewski, M., Swift, R.C., and Karplus, W.J. (1986)
The alliant FX/series: automatic parallelism in a multiprocessor mini-supercomputer.
In *Multiprocessors and Array Processors; Proceedings of the Third Conference*, 1987, San Diego CA, SCSI, pp. 35–44.

[7] Bell, C.G., et al. (1986)
The encore continuum; a complete distributed work station-multiprocessor computing environment.
In *Proceedings of American Federation of Information Processing Societies 1985 National Computer Conference*, 1985, Chicago IL, pp. 147–155.

[8] Billig, R.R., Corbin, S.S., and Moore, R.L. (1986)
A fast backplane cluster heralds a 1000-MIPS computer.
*Electronic Design, 35*, 16 (July 1987), 81–86.

[9] Matelan, N. (1986)
The FLEX/32 multicomputer.
In *Proceedings of the 12th Annual International Symposium on Computer Architecture*, 1985, Boston MA, IEEE, pp. 209–213.

[10] Matelan, N., and Wojcik, A.S. (1986)
The architecture and implementation of the FLEX/32 multicomputer.
In *Proceedings of the American Federation of Information Processing Societies 1985 National Computer Conference*, 1985, Chicago IL, pp. 139–145.

[11] Karplus, W.J., and Fenner, P.R. (1986)
The Flex/32 for real-time multicomputer simulation.
In *Multiprocessors and Array Processors; Proceedings of the Third Conference*, 1987, San Diego CA, SCSI, 127–133.

[12] Geril, H.M., Van Schieveen, P., et al. (1986)
The System 100: an efficient hardware/software architecture for real-time and time-critical simulation.
In *Proceedings of the 2nd European Simulation Congress*, 1986, Antwerp, Belgium, SCSI, pp. 413–19.

[13] Benyon, P.R. (1968)
A review of numerical methods for digital simulation.
*Simulation, 11*, 5 (Nov. 1968), 219–237.

[14] Henrici, P. (1962)
*Discrete Variable Methods in Ordinary Differential Equations.*
New York: Wiley, 1962.

[15] Milne, W.E. (1970)
*Numerical Solution of Differential Equations.*
New York: Dover, 1970.

[16] Gear, C.W. (1971)
*Numerical Initial Value Problems in Ordinary Differential Equations.*
Englewood Cliffs, N.J.: Prentice Hall, 1971.

[17] Korn, G.A., and Wait, J.V. (1978)
*Digital Continuous System Simulation.*
Englewood Cliffs, N.J.: Prentice-Hall, 1978.

[18] Hastings, C. (1955)
*Approximations for Digital Computers.*
Princeton, N.J.: Princeton University Press, 1955.

[19] Carnahan, B., Luther, H.A., and Wilkes, J.O. (1969)
*Applied Numerical Analysis.*
New York: McGraw-Hill, 1969.

[20] Phillips, G.M., and Taylor, P.J. (1973)
*Theory and Applications of Numerical Analysis.*
New York: Academic Press, 1973.

[21] Korn, G.A. (1972)
Back to parallel computation: proposal for a completely new on-line simulation system using standard microcomputer for low-cost multiprocessing.
*Simulation, 19*, 2 (Aug. 1972), 37–45.

[22] Franklin, M.A. (1978)
Parallel solution of ordinary differential equations.
*IEEE Transactions on Computers, 27*, 5 (May 1978), 413–420.

[23] Hamblen, J.O. (1984)
The design and performance of a parallel computer architecture for simulation.
Ph.D. dissertation, School of Electrical Engineering, Georgia Institute of Technology, Atlanta, 1984.

[24] Miranker, W.L., and Liniger, W. (1967)
Parallel methods for the numerical integration of ordinary differential equations.
*Mathematics of Computation, 21* (1967), 303–320.

[25] Birta, L.G., and Abou-Rabia, O. (1987)
Parallel block predictor-corrector methods for ODE's.
*IEEE Transactions on Computers, C-36*, 3 (Mar. 1987), 299–311.

[26] Worland, P.B. (1976)
Parallel methods for the numerical solution of ordinary differential equations.
*IEEE Transactions on Computers, C-25* (Oct. 1987), 1045–1048.

[27] Gear, C.W. (1986)
Potential for parallelism in ordinary differential equations.
Report UIUCDCS-R-86-1246, University of Illinois, Urbana-Champaign, Ill., 1986.

[28] Crosbie, E.J.H., Kerckhoffs, R., and Luker, P. (1986)
Parallel algorithms for ordinary differential equations: an introductory review.
In *Proceedings of the 1986 Summer Computer Simulation Conference*, 1986, Reno NV, SCSI, pp. 947–952.

[29] Sips, H.J., et al. (1986)
A domain decomposition method for the solution of ODEs.
*In Proceedings of the 2nd European Simulation Congress,* 1986, Antwerp, Belgium, SCSI, pp. 226–232.

[30] Mitra, D. (1986)
Asynchronous relaxations for the numerical solution of differential equations by parallel processors.
*SIAM Journal of Scientific & Statistical Computations, 8*, 1 (Jan. 1987), S43–58.

[31] Horiguchi, S., Kawazoe, Y., and Nara, H. (1986)
An algorithm of parallel processing for integration of ordinary differential equations.
*Transactions on Institute of Electronic Information and Communication Eng. E, E70*, 1 (Jan. 1987), 49–55.

[32] Lei, S., et al. (1986)
Clustering technique for rearranging ODE systems, parallel processing techniques for simulation.
In *Proceedings of the First European Workshop on Parallel Processing Techniques for Simulation*, 1986, Manchester UK, pp. 31–43.

[33] Evans, D.J. (1987)
Construction of extrapolation tables by systolic arrays for solving ordinary differential equations.
*Parallel Computing, 4*, 1 (1987), 33–48.

[34] Franklin, M.A. (1981)
VLSI performance comparison of Banyan and Crossbar communication networks.
*IEEE Transactions on Computers, C-30*, 4 (Apr. 1981), 283–291.

[35] IEEE Standard 796 (1983)
*Microcomputer System Bus.*
Washington, D.C.: IEEE Computer Society Press, 1983.

[36] Intel Corporation (1981)
*iSBX bus specification.*
Intel Corporation, Santa Clara, Calif., June 1981.

[37] IEEE Standard 754 (1985)
*IEEE Standard for Binary Floating-Point Arithmetic.*
Washington, D.C.: IEEE Computer Society Press, 1985.

[38] Rauch, K. (1987)
Math chips: how they work.
*IEEE Spectrum, 24*, 7 (July 1987), 25–30.

[39] Aliphas, A., and Feldman, J.A. (1987)
The versatility of digital signal processing chips.
*IEEE Spectrum, 24*, 6 (June 1987), 40–45.

[40] Hamblen, J.O. (1987)
Parallel Continuous System Simulation Using The Transputer.
*Simulation, 49*, 6 (Dec. 1987), 249–253.

[41] Makoui, A. (1986)
Software interface for multiprocessor simulation.
Ph.D. dissertation, Computer Science Department, University of California, Los Angeles, 1986.

[42] Makoui, A., and Karplus, W.J. (1987)
ALI: A CSSL/multiprocessor software interface.
*Simulation, 49*, 6 (Aug. 1987), 63–61.

**James O. Hamblen** (S'74—M'77) was born in Lafayette, Indiana on August 15, 1954. He received the B.S. degree from Georgia Institute of Technology, Atlanta, in 1974, the M.S. degree from Purdue University, Lafayette, Ind., in 1976 and the Ph.D. degree from Georgia Institute of Technology in 1984, all in electrical engineering.

Dr. Hamblen is an Assistant Professor of Electrical Engineering at Georgia Institute of Technology. From 1979 to 1984, he was a Graduate Research Assistant at Georgia Institute of Technology. From 1977–1978, he was a Senior Engineer at Martin Marietta Aerospace, Denver, and from 1976 to 1977, he was a Systems Analyst at Texas Instruments, Austin. His research interests include parallel computer architectures, VLSI design and continuous system simulation.

Dr. Hamblen is a member of Eta Kappa Nu, Tau Beta Pi, and Phi Kappa Phi.

**Cecil O. Alford** (M'64) was born in Gay, Georgia. He received the B.S. and M.S. degrees from Georgia Institute of Technology, Atlanta, in 1956 and 1960, and the Ph.D. degree from Mississippi State University, Greenwood, in 1966, all in electrical engineering.

Dr. Alford is a Professor of Electrical Engineering at Georgia Institute of Technology. From 1963 to 1966, he was an Assistant Professor of Electrical Engineering at Mississippi State University and from 1966–1968, he was an Associate Professor of Electrical Engineering at Tennessee Technological University, Cookeville. His research interests include computer architectures and robotics.

Dr. Alford is a member of Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi, and Sigma Xi.