A Digital Orrery

JAMES H. APPLEGATE, MICHAEL R. DOUGLAS, YEKTA GÜRSEL, PETER HUNTER, CHARLES L. SEITZ, MEMBER, IEEE, AND GERALD JAY SUSSMAN, MEMBER, IEEE

Abstract — **We have designed and built the Orrery, a special computer for high-speed high-precision orbital mechanics computations. On the problems the Orrery was designed to solve, it achieves approximately 10 Mflops in about 1 ft^3 of space while consuming 150 W of power. The specialized parallel architecture of the Orrery, which is well matched to orbital mechanics problems, is the key to obtaining such high performance. In this paper we discuss the design, construction, and programming of the Orrery.**

Index Terms — **Computer architecture, N-body computations, numerical computation, orbital mechanics, parallel computation.**

INTRODUCTION

THE Orrery is a specialized but programmable high-performance computer designed for finding solutions with high numerical precision to the equations of motion for systems with a small number of bodies which move in nearly circular orbits, such as the solar system. At each step in such a calculation, the bodies in the system are considered pairwise to compute the acceleration of each due to the others, and the total acceleration of each body is accumulated. These accelerations are then integrated, using a traditional integration algorithm such as Cowell's method (Cowell and Crommelin [8], Brouwer and Clemence [3]) to compute the next state of the system. This method has been used successfully (by Cohen, Hubbard, and Oesterwinter [6]) to compute the orbital elements of the outer planets for one million years.

We built the Orrery as a cost-effective instrument to attack questions about the dynamical state and long-term stability of the solar system. For example, the stability of the orbit of Pluto is an open question. The perihelion of the orbit of Pluto lies within the orbit of Neptune. Unless prevented by some mechanism, a close encounter between the two planets will eventually occur, and the orbit of Pluto will be disrupted. In a 120 000 year numerical integration, Cohen and Hubbard [5] discovered a stabilization mechanism based on the fact that the orbits of Pluto and Neptune are in a 3/2 resonance — that is, Pluto makes two orbits for every three of Neptune. The origin and stability of this resonance over the age of the solar system is still an open question (Brouwer [2], Williams and Benson [22], Nacozy and Diehl [16]). Indeed, did Pluto form in the resonance or was it later captured into it? The solution will shed light on the formation and evolution of the solar system.

For another example, the distribution of asteroids in the asteroid belt is uneven. Pronounced gaps (the Kirkwood gaps) exist in the inner belt at periods resonant with Jupiter. The distribution in the outer belt is characterized by concentrations of asteroids at periods resonant with Jupiter. While the origin of these features is still a subject of debate (e.g., Greenberg and Scholl [9]), the idea (Wisdom [23], [24]) that the gaps are chaotic regions in phase space has had considerable success in describing the 3/1 Kirkwood gap. Numerical integrations with the Orrery will provide an important check on Wisdom's approximate theory. Understanding the nature of the Kirkwood gaps will solve a long-standing astronomical problem, and in addition, it may shed light on questions concerning the occurrence of chaotic motion in small dynamical systems (e.g., Chirikov [4], Hellerman [12]).

In an N-body problem, the interaction of each body with the $N - 1$ other bodies determines the resultant force, hence the acceleration, that will influence the body on each step. In a serial computer this force computation and accumulation step takes $O(N^2)$ time, while the integration for the N bodies requires $O(N)$ time. The parallel architecture of the Orrery allows the force computation to be performed in $O(N)$ time, and the integration in $O(1)$ time, by the concurrent use of N identical hardware units, called "planet computers."

The Orrery is composed of $N = 10$ planet computers, each of which has one body assigned to it in the simplest programs. More generally, one can compute M-body problems by assigning (M/N) bodies to each planet computer. All the planet computers execute the same instruction, broadcast from a central control computer, in parallel. This architecture is referred to in the parallel processing literature as a single-instruction multiple-data (SIMD) organization. An earlier example of this class of architecture was the Illiac IV (Bouknight [1]), a machine that was of similar performance for a more general class of problems, although with its earlier technology its volume and power consumption were several orders of magnitude greater than those of the Orrery.

The planet computers are connected in a ring, by which each can send data to one neighbor. This simple communication plan is sufficient to communicate the position and

Manuscript received November 22, 1984. This work was supported in part by the National Science Foundation under Grant AST 831-3725, in part by NASA under Grant NGL 05-002-003, and in part by the Defense Advanced Research Projects Agency, ARPA Order 3771, monitored by the Office of Naval Research under Contract N00014-79-C-0579.

J. H. Applegate is with the Department of Astronomy, Columbia University, New York, NY 10027.

M. R. Douglas is with the Department of Physics, California Institute of Technology, Pasadena, CA 91125.

Y. Gürsel is with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

P. Hunter and C. L. Seitz are with the Department of Computer Science, California Institute of Technology, Pasadena, CA 91125.

G. J. Sussman is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

0018-9340/85/0900-0822$01.00

mass for a body in one computer to all the other computers, so each can calculate its acceleration due to the other bodies. The ring is also sufficient for sending results of these force computations back to the original computer, so that for symmetrical forces, the force between each pair of bodies is calculated only once. For a ring with an odd number of planet computers, say seven, body 1 interacts with bodies 2, 3, and 4 in their computers, while bodies 7, 6, and 5 interact with body 1 in its computer, similarly for all seven bodies, for $(N(N-1))/2$ force calculations, after which four communication steps return each mass, position, and accumulated force to its home computer.

The Orrery is used as a back-end processor, attached to a small conventional host computer (e.g., an IBM PC or an HP 9826). The host computer is used to set up and access the states of the particles, and to set up the control sequences for the Orrery.

A Ring of Processors for N-Body Calculations

For a small number of bodies (such as the sun and major planets of the solar system, $N = 10$) we allocate one planet machine to each body. The N planet machines are connected in a ring, such that data can be sent from machine i, $i = 0, \cdots, (N-1)$, to machine $(i+1) \bmod N$, as shown in Fig. 1. The SIMD controller sequences through a stored program whose execution produces a sequence of microcode instructions that are broadcast to and executed in parallel by all the planet computers. There are no data-dependent steps in the program, so the SIMD controller needs no inputs from the planet computers. The best way to understand how an N-body problem maps to the ring architecture of the Orrery is to follow through a simplified (each force is computed twice) N-body code (see the flowchart in Fig. 2).

The computation starts by initializing the accelerations. It then performs $N-1$ acceleration accumulation cycles (AAC's), during which the accelerations of each body due to the others are accumulated. At the end of the $N-1$ AAC's each planet computer independently does an integration step (IS), which gives each body its new state. The pattern of initialization followed by $N-1$ AAC's followed by an IS is repeated until the dynamical system being studied has evolved the desired number of steps.

Initially, each body has a well-defined state. The system starts up in fixed field mode. Each body must initialize its acceleration to respond to the fixed fields. Fixed fields produce accelerations that depend only on the state of each body in isolation, without considering the accelerations due to interactions with other bodies. Fixed fields may model ambient gravitational potentials, velocity-dependent drag, ambient electric or magnetic fields, etc. In the solar-system problem, if the sun is considered a body, the fixed fields are zero, so each body must initialize its acceleration to zero. (One may also fix the sun and write the equations as a fixed field, perturbed by the major planets.)

After initializing the accelerations, the system enters the acceleration accumulation mode. The state of the body in each computer is put into its "R buffer," a register which contains the message it will send to its right neighbor. In each AAC, the computer sends out the old contents of its R buffer to its right neighbor and reads in the new value from its left

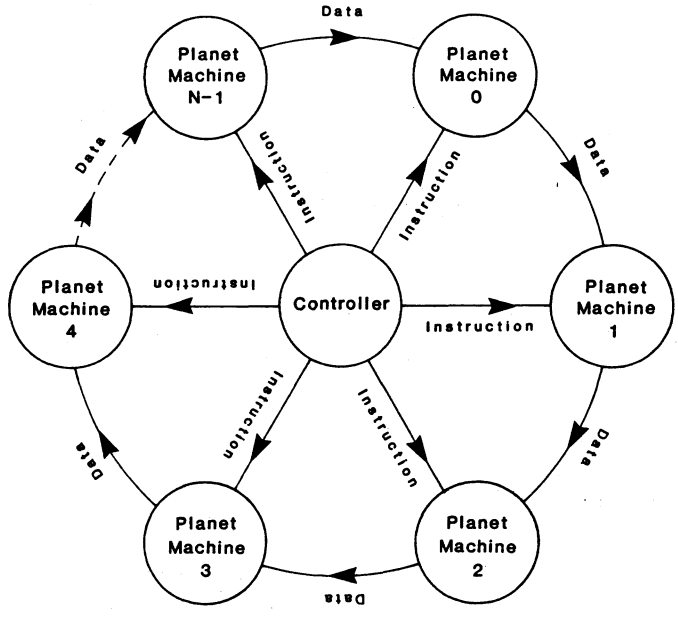

Fig. 1. Ring of computers.

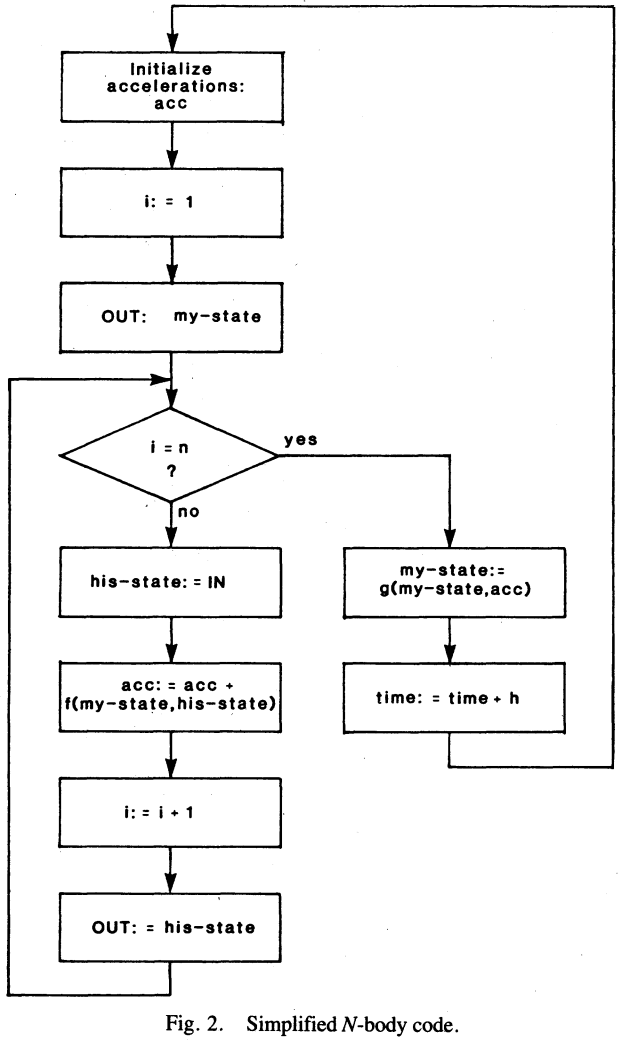

Fig. 2. Simplified N-body code.

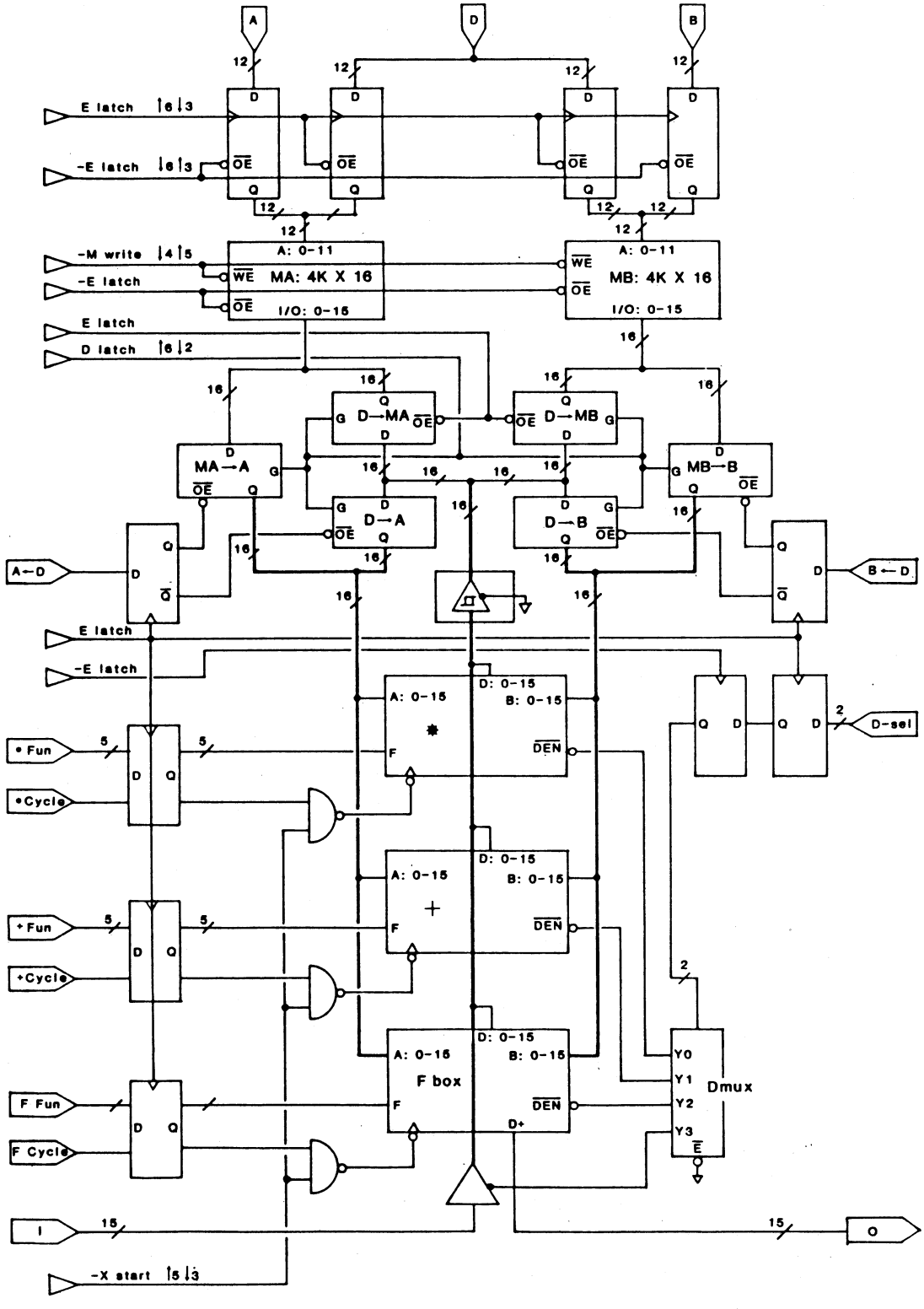

Fig. 3. Planet computer design.

neighbor. The computer also computes and accumulates the acceleration on its own planet due to the planet whose state it is accepting into its R buffer. Thus, in $N - 1$ AAC's each body sees and accumulates accelerations due to the states of each of the $N - 1$ other bodies. Actually, as mentioned above, one can be more clever with the algorithms and avoid the redundant computation of symmetric forces.

After the accelerations are accumulated, the communications stop, and each planet computer independently computes the next state of the body for which it is responsible. We use linear multistep algorithms, which form the next state from a linear combination of previous states and accelerations. These are efficient algorithms, in terms of the number of force calculations required per step, and they are easy to program for the Orrery. We can program these integrations either as pure predictor or predictor–corrector algorithms

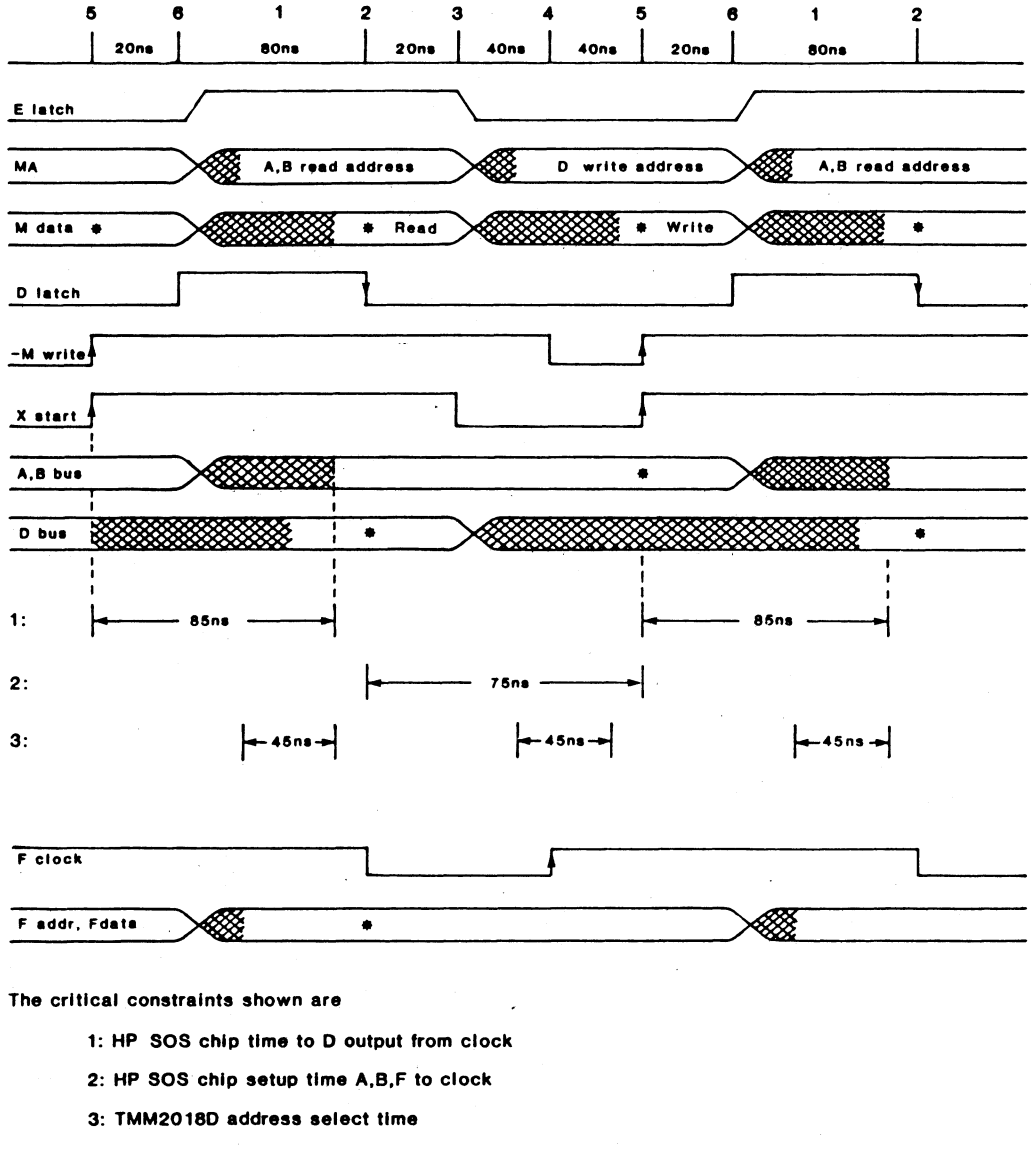

Fig. 4. Timing of Orrery.

(see Hamming [11]), although the latter requires a second acceleration accumulation per step for the corrector. The solar-system problems are especially amenable to this kind of integration because the nearly circular orbits can be efficiently integrated with constant step size.

The Planet Computers

Each planet computer is a data path machine, with most of the instruction decoding "factored out" into the SIMD controller. It has a relatively general-purpose three-bus architecture with a two-port memory and three execution units. (See Fig. 3.)

Two of the execution units are Hewlett–Packard (HP) floating-point adders and multipliers, which have generously been provided for this project by the HP CICO division (Ware [21]). These advanced silicon-on-sapphire (SOS) chips can perform a 64 bit floating add or floating multiply in about 1.25 μs (although we do not run them quite at full speed). The third execution unit, the "function box," is a table lookup device (designed to have the same timing as an HP floating-point chip) that stores approximations to important special functions such as raising to the $-3/2$ power. These approximations are used as starting values for Newton's method iterations. The function box may be used to perform various simple functions that only change the exponent, such as dividing a number by 2, or that change the top eight mantissa bits of a floating number. In addition, communication with the neighboring machines is accomplished by routing a value from the function box of one machine to the D bus of the next.

The HP chips can be run in either "scalar" or "vector" mode. In scalar mode an add or multiply operation takes four microcycles: a cycle to accept the input operands, two "think" cycles, and a cycle for extracting the result. In vector mode, the first think cycle after taking in new operands can be overlapped with the extraction of the result from computation with the previous set of operands. The control structure allows a programmer to use the HP chips in either scalar or vector mode, or in any combination that may be most effec-

tive. The data path allows a result to be presented as an operand to another execution unit and to be stored in memory in a single microcycle. It also allows the think cycles for any execution unit to be used for data transfers among the other execution units and memory.

The Orrery is organized around these microcycles — each line of code in a program for the Orrery specifies the actions that happen in one 800 ns microcycle.

Each execution unit has two input ports and one output port. These ports, as well as all the data buses of the planet computers, are 16 bits wide. The transfer of the 64 bit operands and the results in a microcycle is accordingly performed in four nanocycles, each taking 200 ns. Each nanocycle is itself broken into two phases, a read phase and a write phase. (The detailed timing diagram is shown in Fig. 4.) In the read phase, values specified by the A and B addresses are fetched from the memory and held in the memory-buffer latches ($MA \rightarrow A, MB \rightarrow B$). Simultaneously, a result is taken from an execution unit (or the left-neighbor machine) and stored in the result-buffer ($D \rightarrow A, D \rightarrow B, D \rightarrow MA, D \rightarrow MB$) latches. The A and B arguments are selected from either the result-buffer latches or the memory-buffer latches by enabling the correct tristate outputs. In the write phase, the selected arguments are latched by the target execution units, and simultaneously, the result picked up in the read phase is stored in the memory location given by the D address.

The function box contains a 4K × 16 RAM, whose addresses can be constructed from the bits of the input data. There is an internal bus (the F bus) for the RAM's address and data. The bits of the function box A or B arguments are distributed to various address-part registers. They are then recombined to make up the table addresses for the various functions that may be stored in the RAM. The function box is timed to accept its argument in four 16 bit chunks and store them in its memory. It then uses a "think" cycle to perform the required lookups in its tables and arrange the answers for output. After one think cycle it is ready to put out the new floating point answer as four 16 bit chunks on the D bus.

As an example, we describe the process of computing the starting approximation for the Newton–Raphson iteration that computes the $-3/2$ power of an argument. The starting approximations are stored in two tables, each 1K × 16 in size, a new mantissa table, and a new exponent table. The computation of the starting approximation consists of computing the addresses used in the table lookups.

The new mantissa table is a table whose elements are the mantissas of the $-3/2$ power of the number at the midpoint of the interval defined by the address and address $+1$, where the addresses are interpreted as mantissas of floating point numbers. The starting approximation must be accurate to 9 bits for the Newton–Raphson iteration to converge to 55 bits of accuracy in three iterations. A new mantissa accurate to 9 bits requires knowing the old mantissa to 10 bits of accuracy. However, one of these bits comes for free; the most significant bit of the old mantissa is always a one since the argument of the $-3/2$ power is always positive. Nine of the ten bits of the new mantissa address are the second through tenth most significant bits of the old mantissa. The tenth bit of the address is the even/odd bit of the old exponent. Separate new mantissa tables are required for even and odd old exponents because, apart from a bias, the new exponent is $-3/2$ times the old exponent. If the old exponent is even, multiplication by $-3/2$ yields an integer; in this case, the old mantissa is on the interval $[1/2, 1)$, and the new mantissa table is constructed accordingly. If the old exponent is odd, one is added to it, and the old mantissa is divided by 2; in this case, the new mantissa table is constructed assuming the old mantissa lies on the interval $[1/4, 1/2)$.

One further complication arises because the starting approximation must be a normalized floating-point number. Allowing for odd old exponents, the old mantissa lies on the interval $[1/4, 1)$. This interval is mapped onto the interval $(1, 8]$ by the function $x^{-3/2}$; thus, 1, 2, 3, or 4 must be added to the new exponent, depending on the value of the new mantissa. Actually, the case of $bias = 4$ never arises because the midpoints of intervals are used in the new mantissa table. The amount by which the new exponent must be biased is encoded in the bottom two bits of the new mantissa and is used in forming the new exponent address.

The new exponent address consists of the 8 bit old exponent and the 2 bit exponent bias from the new mantissa. The elements of the table are $(-3/2)(oldexponent) + bias$ if the old exponent is even, and $(-3/2)(oldexponent + 1) + bias$ if the old exponent is odd. The flow of data in the process of computing the starting approximation is shown in Fig. 5.

The Controller Engine

The Orrery controller has two purposes: 1) storing, sequencing, and broadcasting the microcode instructions, and 2) communicating with the host computer. The communication with the host computer includes both the programs and the data in the ring.

The microcode is broadcast by a rather elaborate state machine, consisting of two coupled state machines, the "microengine," and the "nanoengine" (see Fig. 6). The microengine is responsible for generating those bits of the microinstruction that change only on microcycle boundaries: for example, the top bits of the A, B, and D addresses, whether or not the A, B, or D addresses are modified by the index register, whether or not the index register is to be loaded or is to count in this cycle, whether or not the host computer should be awakened in this cycle, how the microengine should compute its next state in this cycle, and what the nanoengine should do in this microcycle. The nanoengine is responsible for generating those bits of the microinstruction that change on nanocycle boundaries: for example, which 16 bit chunk of a floating-point number is currently being addressed, which parts control the buses, the opcode that must be sent to each execution unit, and other detailed logical controls. The nanoinstruction also determines whether the microengine is to be resumed in its sequence. There may be up to 16 nanoinstructions for each microinstruction (although we have found no use for long sequences of nanoinstructions

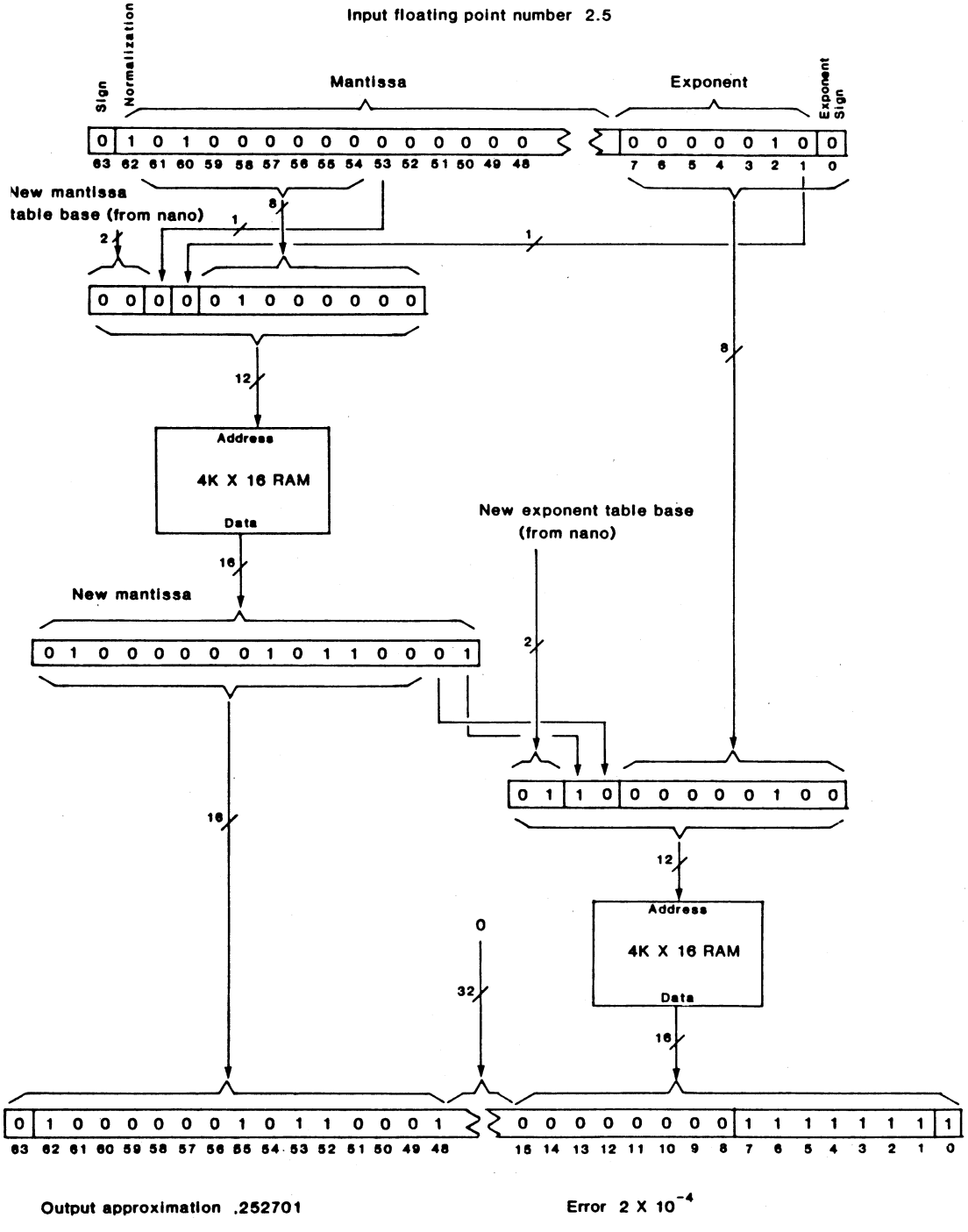

Fig. 5. Initial approximation.

as yet). The formats of the micro- and nanoinstructions are shown in Fig. 7.

The controller also contains the host-computer interface. The host computer must be able to start and stop the Orrery, to fill the microcode and nanocode memories, and to put data into and extract data from the ring. The host interface is an 8 bit parallel bidirectional data bus controlled by eight parallel control wires. Six of the control wires are set by the host and two are set by the Orrery. The host uses five of the six control wires to set up a port instruction and the remaining wire to strobe that instruction into the Orrery, which then acknowledges the port instruction with a handshake on one of its response wires. The other response wire is used by the Orrery to asynchronously call the attention of the host. The port instructions allow the host to change the values of special control registers and to set up or read out microcode instructions or ring data from 8 bit segments. All of this serial data movement is organized around a long shift register.

Construction and Packaging

We packaged the Orrery as one board for each planet computer (see Fig. 8) and one board for the controller/host interface, all plugged into a common back plane. The boards are

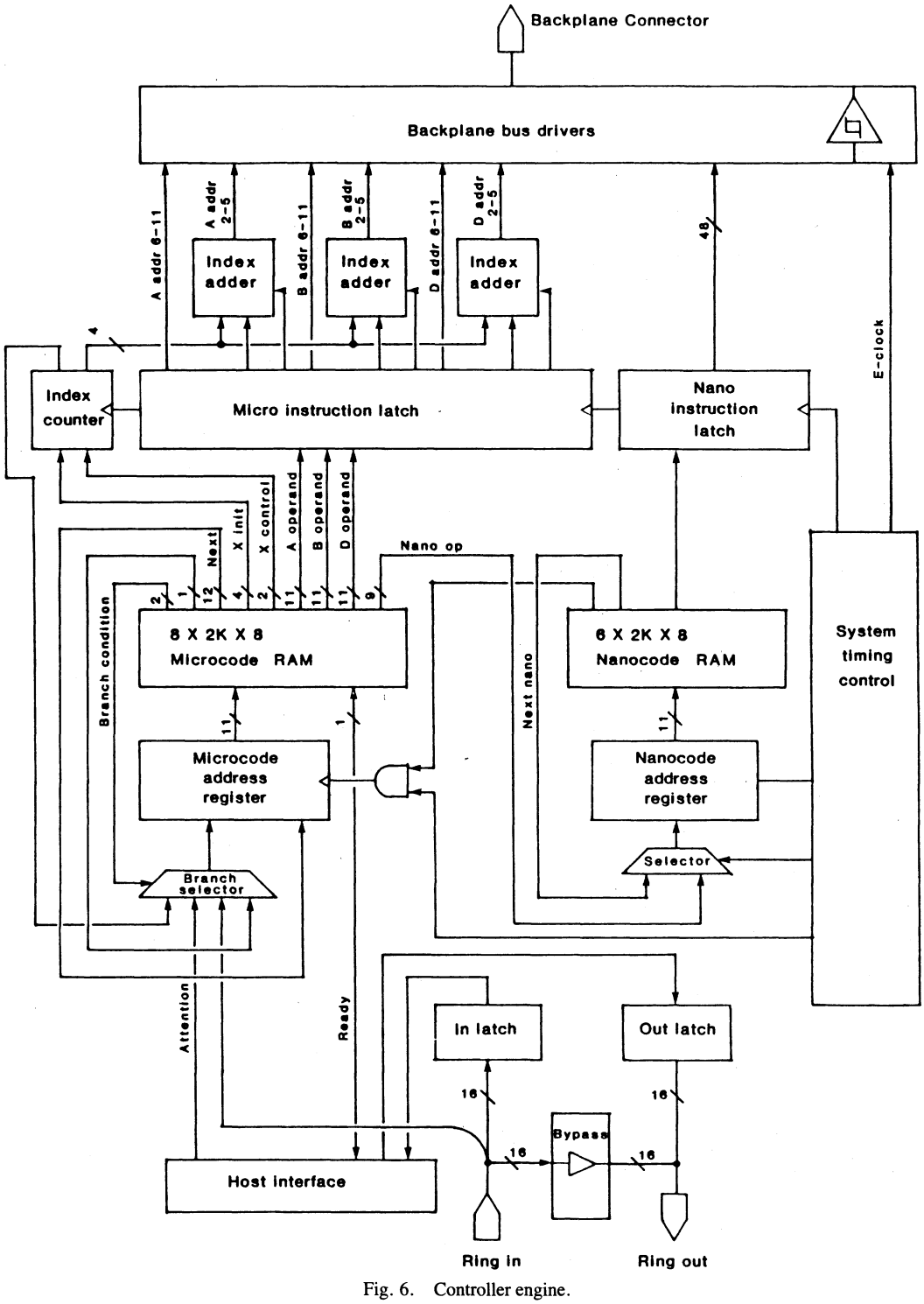

Fig. 6. Controller engine.

a standard multibus outline, and the card frame and back plane are standard multibus assemblies. The planet computer boards have 70 packages on each, and the controller board has 84 packages. The power dissipation is low, so cooling is easily accomplished with muffin fans mounted on the card cage.

The separation of data and control in the Orrery is reflected in the packaging. The back-plane side of the circuit boards is used for control signals, like pins being bused for instruction broadcast (and for power distribution). The interplanetary ring communication is provided by ribbon cable interconnects on the opposite side of the board (see Fig. 9), which allows for simple expansion of a machine and rearrangement of boards.

Since only one SIMD controller board was required, it is constructed on a standard wire-wrap multibus board with power and ground planes. The planet computers are assembled in four-layer printed-circuit technology. The inner layers are power and ground planes, and the outer layers are signal runs with 8 mil traces and 8 mil spacing. The boards were made by MOSIS (Cohen [7], Lewicki *et al.* [15]) from a symbolic description produced with the Earl (Kingsley

Microcode instruction		
Bits	Field name	Possible values
63	signal ready to host	{ No signal, signal }
62-61	branch condition	{ normal, ring bit 15, attention, index=15 }
60-48	next-micro-address	
47-44	index-literal	{ 0 ... 15 }
43	load-index-register	{ load, no load }
42	count-index	{ no count, count }
41-32	a-address-bits 11-2	{ floating number address }
31	a-address-add-index	{ no index, index }
30-21	b-address-bits 11-2	{ floating number address }
20	b-address-add-index	{ no index, index }
19-10	d-address bits 11-2	{ floating number address }
9	d-address-add-index	{ no index, index }
8-0	opcode	times 4 gives nanocode address

Nanocode instruction		
Bits	Field name	Possible values
47	next micro instruction	{ continue nano, continue micro }
46-43	next nano address	{ 0 ... 15 } (the high bits of the nano address are constant throughout a microinstruction.)
42-41	abd-chunk address	{ 0 ... 3 }
40	a bus source	{ from memory, from d bus }
39	b bus source	{ from memory, from d bus }
38-37	d bus source	{ *, +, f box, neighbor's R buffer }
36-32	* operation	{ see HP documentation }
31	* clock	{ no clock, clock }
30-26	+ operation	{ see HP documentation }
25	+ clock	{ no clock, clock }
24	f box in	{ no clock, clock }
23-21	f bus part destination	{ none, unused, unused, unused, old mantissa, old exponent, new mantissa, low byte buffer }
20-18	f bus byte destination	{ none, unused, unused, unused, R buffer, F memory, D bus, high byte buffer }
17-16	f bus source	{ F memory, byte buffer, A arg, B arg }
15-14	f address select (bits 8-9)	{ literal, new mantissa, old exponent, unused }
13-12	f address select (bits 0-7)	{ unused, literal, old mantissa, old exponent }
11-0	f literal address	{ literal address when selected }

Fig. 7. Microcode instruction formats.

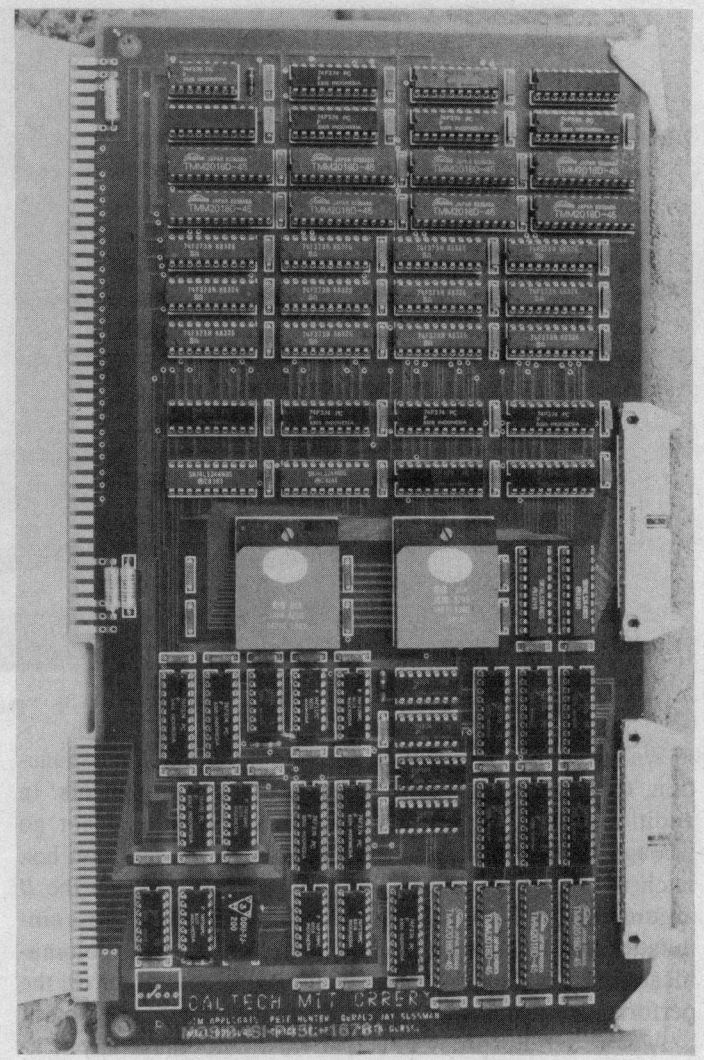

Fig. 8. The planet computer board.

[13]) computer-aided layout system. All of the logic of the planet computers (with the exception of the HP SOS chips and 74LS244 output buffers for the HP chips) are 74 Fxxx technology.

One of the reasons why this machine was so easy to bring up is that each board has a good ground plane, so signals are clean and there are no significant noise problems. Another way we avoid noise problems is to have only slow signals on the back plane. There are no signals on the back plane that change more often than 200 ns, except the system clock, which has a cycle time of 200 ns and a duty cycle of about 50 percent. The planet boards need many signal transitions and time references in each 200 ns interval. These signals are derived on each planet board independently with a lumped parameter delay line, although all these signals are entrained by the 200 ns system clock. Another design principle that we followed to improve reliability is that all registers are positively timed. We do not use the standard technique of clocking a register on data that are allowed to change on the same clock edge. This conservatism costs a bit in hardware and in speed, but the insensitivity of the design to skew and other timing problems is well worth the price. In fact, we are pleased to report that, although the machine was designed for a 200 ns cycle time, we have been running the machine with a 150 ns clock without any sign of unreliable behavior.

Results

We have measured the performance of the Orrery on the problem of Cohen, Hubbard, and Oesterwinter [6] — a high-precision integration of the orbits of the outer planets (Jupiter, Saturn, Uranus, Neptune, and Pluto). This can be configured as a five-body problem in heliocentric coordinates, or it can be configured as a six-body problem in center-of-mass coordinates. Tremaine (private communication) has run the heliocentric problem on a DEC VAX-11/780 in Fortran. The VAX does a 40 000 year integration in about 1 h of CPU time. The Orrery, configured with six planet boards to do the problem in center-of-mass coordinates, does the same integration in about 100 s (using the 150 ns clock). This makes each board about 6 VAX equivalents, and thus, a full 10-board Orrery is about 60 VAX equivalents.

What We Did Wrong

Although the Orrery is a good computer, there are a number of bad choices we made in the design. Some of these are minor points which could easily be fixed; others are more major — fixing them would require a significant redesign.

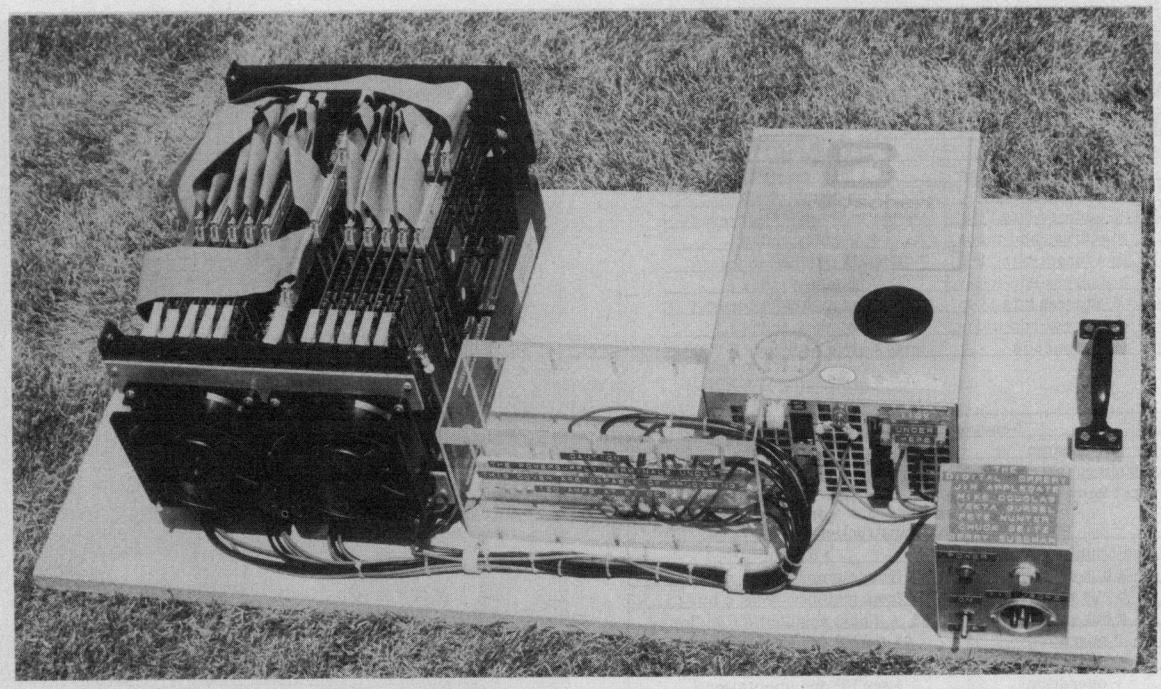

Fig. 9. The digital Orrery.

We never use the function box to compute a binary function, thus it need not latch both the A and B arguments. In addition, the interplanetary communications need not go through the function box. We could have the function box latch the A argument and the communications latch the B argument, thus decoupling them and allowing the computation of a function to go on in parallel with communication. Such a modification would marginally improve the performance and reduce the chip count on the planet board.

A more serious problem is that we did not make any flexible provisions for conditionals computed on the planet computers to influence the flow of control in the controller. The only planet board condition that the microcode can currently branch on is the high bit of the value on the interplanetary ring. (This can be used to look at the sign of a floating-point number being passed on the ring.) We should have allowed for more general conditionals, communicated by open-collector logic on the back plane. These would be useful for testing for error conditions such as floating overflow or underflow, or for changing step size when locally determined to be necessary by a planet computer. This change is easy, and we may patch our planet computers to put out such information.

A much more serious problem is that we currently have no good way of implementing higher precision computation than the 55 bit precision provided by the HP parts. This is a real problem in that the high-order integrators we are using are sensitive to roundoff error in the low-order bits. In particular, Cowell's predictor is of the form

$$x_{n+1} = 2x_n \ominus x_{n-1} \oplus h^2[c_0a_n + c_1a_{n-1} + \cdots + c_ma_{n-m}]$$

where x_i are the positions and a_i are the associated accelerations. To get high accuracy with this scheme we must store a high-precision value for x_i, and we must perform some of the operations (marked with a "$\bigcirc$") in high precision. Because we do not have any built-in mechanism for high-precision computation, we use an expensive procedure (see Knuth [14, pp. 220, 221, Theorems B, C]) to do the required high-precision computations. These few high-precision computations cost about 20 percent of the total time used in integrating orbits. Prior thought would have revealed this problem early in the design when we could have done something about it. (@#%!)

The controller could be significantly improved, as well. The current controller design does not allow similar program segments to share code. What is needed is a mechanism for microcode subroutines, and a set of base registers to point at the segments of memory that are the arguments of those subroutines. This change will be necessary to allow the machine to manipulate many particles on each board. We currently must have multiple copies of almost identical programs in the microcode memory to manipulate multiple particles on each board. Thus, the number of particles we can handle is limited by the amount of microcode memory available.

Acknowledgment

The authors are grateful to the Hewlett–Packard Company, Cupertino Integrated Circuits Organization (HP–CICO), for supplying the special floating-point adder and multiplier chips which made this project possible. F. Ware, W. McAllester, and D. Zuras of HP–CICO took great interest in this project. They were helpful and supportive. They provided the necessary documentation, and they made themselves available to answer questions about the use of their chips. In addition, Hewlett–Packard Laboratories supplied computation in support of the design process. The authors also thank K. Fry and D. Cohen of the MOSIS crew for fabrication of the planet computer printed circuit cards. T. Knight and S. Tremaine of M.I.T., and J. Lamping of Stanford participated in initial discussions of the feasibility

and uses for such a computer. The authors thank B. Heepe for drawing the diagrams. The initial impetus to make such a machine came from reading the work of J. Wisdom on the Kirkwood gaps in the asteroid belt.

REFERENCES

[1] W. J. Bouknight *et al.*, "The Illiac IV system," *Proc. IEEE*, vol. 60, p. 369, 1972.
[2] D. Brouwer, "The theory of orbits in the solar system and in stellar systems," in *Proc. IAU Symp. 25*, G. Contopoulos, Ed. New York: Academic, 1966.
[3] D. Brouwer and G. M. Clemence, *Methods of Celestial Mechanics*. New York: Academic, 1961.
[4] B. V. Chirikov, "A universal instability in many-dimensional oscillator systems," *Phys. Rep.*, vol. 52, p. 283, 1979.
[5] C. J. Cohen and E. C. Hubbard, "Libration of the close approaches of Pluto to Neptune," *Astron. J.*, vol. 70, no. 10, 1965.
[6] C. J. Cohen, E. C. Hubbard, and C. Oesterwinter, "Elements of the outer planets for one million years," in *Astron. Papers Amer. Ephemeris Nautical Almanac*, vol. XXII, pt. I, U.S. Naval Observatory.
[7] D. Cohen, "The MOSIS story," in *Proc. 4th Jerusalem Conf. Inform. Technol.*, IEEE cat. 84CH2022-2, 1984, p. 650.
[8] P. H. Cowell and A. C. D. Crommelin, "Investigation of the motion of Halley's Comet from 1759–1910," in Appendix, *Greenwich Observations 1909*, Bellevue, England: Neill, 1910.
[9] R. Greenberg and H. Scholl, "Resonances in the asteroid belt," in *Asteroids*. Tucson, AZ: Univ. Arizona Press, 1979, p. 310.
[10] P. Goldreich and S. Tremaine, "Dynamics of planetary rings," *Annu. Rev. Astron. Astrophys.*, vol. 20, p. 249, 1982.
[11] R. W. Hamming, *Numerical Methods for Scientists and Engineers*. New York: McGraw-Hill, 1973.
[12] R. H. G. Hellerman, "Self-generated chaotic behaviour in nonlinear mechanics," in *Fundamental Problems in Statistical Mechanics V*, E. G. D. Cohen, Ed. New York: North-Holland, 1981.
[13] C. Kingsley, "Earl: An integrated circuit design language," Dep. Comput. Sci., Calif. Inst. Technol., Pasadena, Tech. Rep. 5021:TR:82, May 1982.
[14] D. E. Knuth, *The Art of Computer Programming, Vol. 2*. Reading, MA: Addison-Wesley, 1981.
[15] G. Lewicki, D. Cohen, P. Losleben, and D. Trotter, "MOSIS: Present and future," in *Proc. M.I.T. Conf. Adv. Res. VLSI*. Dedham, MA: Artech Books, 1984, pp. 124–128.
[16] P. E. Nacozy and R. E. Diehl, "On the long term motion of Pluto," *Astron. J.*, vol. 83, p. 522, 1978.
[17] C. Oesterwinter and C. J. Cohen, "New orbital elements for the moon and planets," in *Celestial Mechanics 5*. Dordrecht, The Netherlands: D. Reidel, 1972, pp. 317–395.
[18] H. Scholl, "Recent work on the origin of the Kirkwood gaps," in *Dynamics of the Solar System*, R. Duncombe, Ed. Dordrecht, The Netherlands: D. Reidel, 1979.
[19] E. L. Stiefel and G. Scheifele, *Linear and Regular Celestial Mechanics, Perturbed Two-Body Motion*. New York: Springer-Verlag, 1971.
[20] V. Szebhely, *Theory of Orbits, The Restricted Problem of Three Bodies*. New York: Academic, 1967.
[21] F. A. Ware, "A 64-bit floating point processor chip set," in *Proc. ISSCC*, vol. 24, 1982.
[22] J. G. Williams and G. S. Benson, "Resonances in the Neptune–Pluto system," *Astron. J.*, vol. 76, p. 167, 1971.
[23] J. Wisdom, "The origin of the Kirkwood gaps, etc.," *Astron. J.*, vol. 87, p. 577, 1982.
[24] ——, "Chaotic behavior and the origin of 3/1 Kirkwood gap," *Icarus*, vol. 56, pp. 51–74, 1984.

James H. Applegate received the B.S. degree in astrophysics from Michigan State University, East Lansing, in 1976 and received the Ph.D. degree in physics in 1980 from the State University of New York at Stony Brook.

He was a Postdoctoral Fellow at NORDITA, Copenhagen, Denmark, and then became a Bantrell Research Fellow at the California Institute of Technology, Pasadena, from 1981 to 1984. He is now Assistant Professor of Astronomy at Columbia University, New York. His principal research interests lie in dense-matter astrophysics and in stellar dynamics.

Michael R. Douglas received the B.A. degree in physics from Harvard University, Cambridge, MA, in 1983. He is currently working on the Ph.D. degree in physics at the California Institute of Technology, Pasadena.

His research interests include theoretical particle physics, programming languages, and artificial intelligence.

Yekta Gürsel was born in Samsun, Turkey. He received the Ph.D. degree in theoretical physics from the California Institute of Technology, Pasadena.

He worked in the Gravitational Physics Laboratory at the California Institute of Technology. He now works on computer architecture design in the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (M.I.T.), Cambridge.

Peter Hunter was a technician in the Department of Computer Science Laboratory at the California Institute of Technology, Pasadena. He has worked on a number of architecture and layout projects. He did most of the printed-circuit layout for the Orrery planet computer card.

Charles L. Seitz (S'68–M'69) received the B.S., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology (M.I.T.), Cambridge.

He is now a Professor of Computer Science at the California Institute of Technology, Pasadena, where his research and teaching activities are in the areas of VLSI architecture and design, concurrent computation, and self-timed systems. Prior to joining the faculty of the California Institute of Technology, he worked as an industrial consultant from 1972 to 1977, principally for the Burroughs Corporation, was an Assistant Professor of Computer Science at the University of Utah, Salt Lake City, from 1970 to 1972, and was a Member of the Technical Staff of the Evans and Sutherland Computer Corporation from 1969 to 1971. While at M.I.T. he was an Instructor of Electrical Engineering.

Dr. Seitz is a member of the Association for Computing Machinery and of the IEEE Computer Society, and was the recipient of the Goodwin Medal for "conspicuously effective teaching" at M.I.T.

Gerald Jay Sussman (M'80) received the S.B. and Ph.D. degrees in mathematics from the Massachusetts Institute of Technology (M.I.T), Cambridge, in 1968 and 1973, respectively.

He has been involved in artificial intelligence research at M.I.T. since 1964. He has also worked in computer languages and in computer architecture. He spent the 1983–1984 academic year in the Theoretical Astrophysics Group at the California Institute of Technology, Pasadena. He is presently a Professor of Electrical Engineering at M.I.T.

Dr. Sussman has recently coauthored (with H. Abelson and J. Sussman) the introductory computer science textbook used at M.I.T.