View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

N89-15943

A MULTI-SATELLITE ORBIT DETERMINATION PROBLEM IN A PARALLEL PROCESSING ENVIRONMENT

M. S. Deakyne and R. J. Anderle

General Electric Valley Forge
Military and Data Systems Operations
Engineering Orbit Analysis Unit

ABSTRACT:
The Engineering Orbit Analysis Unit at GE Valley Forge had available to it an Intel Hypercube Parallel Processor. It was decided to investigate the performance and gain experience of parallel processors with a multi-satellite orbit determination problem. A general study was selected in which major blocks of computation for the multi-satellite orbit computations would be used as units to be assigned to the various processors on the Hypercube. Problems encountered or successes achieved in addressing the orbit determination problem would be more likely to be transferable to other parallel processors.

Our prime objective was to study the algorithm to allow processing of observations later in time than those employed in the state update. We would exploit our expertise in ephemeris determination in addressing these problems and use the facility to bring a realism to the study which would highlight the problems which may not otherwise be anticipated. Our secondary objectives were to gain experience of a non-trivial problem in a parallel processor environment, explore the necessary interplay of serial and parallel sections of the algorithm in terms of timing studies, to explore the granularity (coarse vs. fine grain) to discover the granularity limit above which there would be a risk of starvation where the majority of nodes would be idle or under the limit where the overhead associated with splitting the problem may require more work and communication time than is useful. We could also see the pros and cons of local versus shared memory.

Traditional algorithms for filtering and smoothing within the orbit determination problem have been sequential in nature. Real time filter algorithms imposes constraints on the implementation of the problem on any parallel computer. The computations preceding the state update are extensive and can be solved by small vector processor(s). The computations, arrays and execution time of the update are all extensive, and the third component of concern would be the algorithmic bottleneck which occurs in the updating of the parameters of the state when process noise is used to represent unmodeled errors.

A MULTI-SATELLITE ORBIT DETERMINATION PROBLEM IN A PARALLEL PROCESSING ENVIRONMENT

M. S. Deakyne and R. J. Anderle

INTRODUCTION:

The Orbit Determination Algorithm is a computational intensive problem which can be investigated in terms of increased efficiency with vector, pipeline, and parallel processors. As described below, the approach most intimately connected with the physics of the problem is parallel processing. In 1987, our objective was to decide if parallel processors could be used effectively to determine the orbits of satellites and use the Hypercube to bring a realism to the study which would highlight problems which may not otherwise be anticipated.

The first basic challenges were to become familiar with the many intricate details of the computer architecture and operating system and then to transfer the structure of the algorithm onto the machine architecture of the Hypercube. The complexity and high computational demands of the Orbit Determination Algorithm lent itself to be first logically decomposed into relatively big, computationally independent units. These units would be used as the major blocks of computation assigned to the various processors on the Hypercube. At this stage, we were gaining experience of the Orbit Determination problem in a parallel computing environment. We were discovering the extent of the parallelism within the existing traditional algorithm.

Our next challenge and our prime objective was to study the algorithm to allow processing of observations later in time than those employed in the state update - 'Look Ahead Techniques'. In this stage, we were trying to invent a new piece to the filter algorithm, fundamentally parallel in nature to solve our problem. Problems encountered and successes achieved on the algorithmic level would be more likely to be transferrable to other parallel processors.

THE ALGORITHM:

Traditional algorithms for filtering and smoothing within an Orbit Determination problem have been sequential in nature. Real time filter algorithms impose constraints on the implementation of the problem on any parallel processor. The major segments of an orbit determination problem are:

1. The evaluation of the accelerations of the satellites due to the forces modeled.
2. The numerical integration of these equations of motion.
3. The calculation of the process noise representing unmodeled forces.
4. The calculation of the residuals between the models and the observation.
5. The update of the parameter estimates and the covariance of the estimates.

The update is referred to as filtering when the current time estimates of the parameters are made based on observations prior to that time. Smoothing is when the parameter estimates at a given time are based on observations made after, as well as before, the given time.

The computations preceding the state update are extensive and can be solved by small vector processors. The computations, arrays, and execution time of the update are extensive and can be addressed by vector and/or pipeline processors. The third major area of concern is the algorithmic bottleneck which occurs in the updating of the parameters of the state when process noise is used to represent unmodeled errors.

Within an orbit determination process, using an extended Kalman filter, one must integrate the equations of motion and perturbation equations for all satellites and then compute the process noise before a time update of the covariance can be computed. The residual is found before the Gain is computed, and the measurement update must await for all of the above before its calculation can be performed. Then onto the next measurement. With a single satellite, the force and integration can be done in parallel and the different process noise contributions (i.e. drag, gravity, solar radiation pressure, and clocks) can be done in parallel, independently of each other. With a multi-satellite configuration, the parallelism can be increased by doing all of the above for each satellite in parallel. In the mode of the extended Kalman filter, the algorithmic bottleneck is the measurement update of the state and covariance. The update works in isolation.

APPLYING THE ALGORITHM TO THE MACHINE:

The Hypercube machine is a loosely coupled 32 node multi-processor connected together with a binary n-cube network. Each node had its own sizeable memory with no shared memory and no global synchronization with the host. Communication was achieved by message passing and the computation was data driven.

For a first attack to the parallelism within the orbit determination problem, the Hypercube was a good match to the Orbit Determination problem since the algorithm could be decomposed naturally into logically large and separate independent sub-algorithms. However, the Orbit Determination problem sub-algorithms were diverse in terms of requiring unsynchronized communicating with other pieces of the algorithm which put a challenge on balancing the load and interprocessor communication. Because of the amount of data exchanged, the lack of shared memory was felt as message passing became more and more cumbersome and stilted. And we had no advantage with the Hypercube in terms of dealing with the intensive computational aspect of our problem. We came to believe that the ideal machine would be a coarse grain machine which would allow the underlying concepts of the algorithm to be expressed via the division of the nodes, implementation of the vector package within each node and more efficient mode of communication among the nodes.

However, given our problem and Hypercube facility, we proceeded. The total problem was broken into coarse large sub-problems divided logically along physical concept boundaries. Chunks of code, each dealing with a physical concept, had been then extracted from various sources of standard sequential filter software. Each sub-problem was assigned to a separate process and placed on a separate node. The solution of each node had to be exchanged among the different nodes as the algorithm proceeded. Message passing was a point to point communication path. If there were no direct communication paths between the nodes, the message was routed by intermediate nodes. To handle these messages, a message-delivery scheme was written - the node executive - and was placed on each node as the control center of the flow of data and to coordinate the various node processors.

PRELIMINARY RESULTS:

We achieved a cycling program and began to immediately output timing data. Timing information was difficult to interpret since all the clocks were independent from each other. Intervals of time, concerning wait time, calculation time, and communication time, were output. Reconstructing relative time was difficult. However, from our preliminary results, we found at the end of 1986 an unexplained difference in total run time on the host computer and the overall wait time on each of the nodes. Also, we found that each of the nodes was spending an unacceptable amount of time waiting for information. Placing a synchronization handshake between nodes and the controller did not decrease the difficulty because no message could be broadcast simultaneously and the handshake introduced additional pauses.

During all the work of 1986, the Hypercube machine was physically separated from the group of engineers (i.e. We and the machine were in two different rooms). In 1987, the engineers and the machine were placed in the same room and we could run our program and watch the interplay of the nodes via blinking lights. (Each node on the Hypercube had two lights. When the red light was on, the node was waiting for data; When the green light was on, it was in its calculation mode.) Only then, by viewing these lights did we realize our problems and constraints of implementing this non-trivial problem into the parallel environment. One of our objectives was to explore the algorithm in terms of timing studies. However, by merely observing the lights during an execution of our software, we found that the serial sections of the algorithm were completely dominating the time over the parallel sections. In fact, it was so dominating that it masked completely any saving of time in our different implementations in the parallel sections. Not only was this discovered but also several sections we thought we implemented in a parallel mode were being executed in a sequential mode.

These blinking lights also emphasized the newness and difference of the parallel environment. As we watched the interplay of lights and correlated them to the running sections of the algorithm, we realized that to think of a certain number of processors performing the same task in the same time interval was easy to grasp. But to think and be logically able to handle the different tasks in parallel requiring different intervals of time for calculation and communication and then to tie them together in an efficient parallel mode without reverting to standard inefficient modes of sequential thinking was a challenge.

At this point, we scanned the literature in terms of parallel software techniques and re-visited the existing software package on the Hypercube. Our main objectives now were to explore the necessary interplay of serial and parallel sections of the algorithm in terms of the timing studies, to explore the granularity (coarse vs. fine), and to explore the granularity limit above which there would be a risk of starvation where the majority of nodes would be idle or under the limit where the overhead associated with splitting the problem may require more work and communication time than is useful. We were also exploring the pros and cons of local memory versus shared memory.

Implementing changes into existing software and trying to debug the software was horrific. Unless the debug information messages were written to specifically isolate only certain nodes and certain processes, the person would receive a torrent of messages from all the nodes and the information would be lost in the deluge. All

operations on the multiple processors would not necessarily occur in precisely the same order from execution to execution and would not even be time ordered within the same execution. All debug messages affected the timing of the processes and had to be commented out for timing studies. Often the program would cycle with the debug messages in the system only to crash when the messages were removed. Debugging had to be done in a fine grain piecemeal fashion with the messages being highly restrictive to certain nodes and certain processes.

FINAL RESULTS OF THE HYPERCUBE STUDY:

Over the course of the study, we were able to decrease the original run time of the overall execution time by a factor of eight and we did find a proportionate reduction in execution time with the increasing number of nodes employed in the problem. See Table 1 and Table 2 for a summary of the Four Test Cases in terms of calculation time and wait times for 1986 and 1987, respectively. See Table 3 for a Summary of overall run time for the Four Test Cases. See the Appendix for information and explanation of the different Test Cases and a summary of the 1986 Results.

At this stage of experience and output, we were able to finally hone into the new algorithmic aspects of our study. We defined four different 'Look Ahead Techniques' to attack directly the algorithmic bottleneck of the update. See Table 4.

As we began to implement these 'Look Ahead Techniques', we continually bumped up against the machine architecture in terms of memory allocation on the nodes, message passing, and the demands of load balancing and inter-processor communication. To preserve the generality of our study, we scanned the literature and established contacts with Corporate Research Development Labs (CRD). Our evolving approach was to bring together the estimation expertise, the experience of the users in the parallel environment, and the architectural expertise and computing resources of the laboratories. If this approach was followed, it would make it possible to review the real-time speed and numerical performance of the orbit determination package in terms of the implementation, independent of the particular machine architecture, while maintaining the correct view on the algorithmic level.

Our objective now was to define a benchmark orbit determination problem to use to evaluate and demonstrate new improvements to the algorithm using various mapping architectures of existing parallel computers. We developed and wrote a sequential orbit determination package which contained the same realistic models of the satellite dynamics, gravity, drag, solar radiation, GPS, and ground clock noise contained in the Hypercube program.

CONCLUSIONS:

The test results finally showed an improvement in efficiency of ephemeris computations with an increase in the number of nodes utilized. Experimentation and experience caused us to stop our implementation of the 'Look Ahead Techniques' on the Hypercube and re-direct the IR&D effort to a broader baseline. The Hypercube machine was a viable necessary tool to gain experience in parallel processing and bring the realism to the study. However, the Hypercube type of machine architecture, which we used in this study, is not the best one which matches the structure of the orbit determination problem in terms of increased efficiency. (New Upgrades to the Hypercube have been noted in the literature which eases message passing.) But the Orbit Determination problem is still a viable problem for parallel processing.

Our experience should be expanded to machines such as the Warp II, Cray, the Butterfly, and the Connection Machine to determine the efficiency of the implementation with the focus on the measurement update.

The project should be conducted with parallel support from M&DSO and CRD. The achievable throughput, cost, and reliability of large scale filters in a parallel environment is a very important and known next step to accomplish.

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE 1

1987 HYPERCUBE EPHEMERIS PROJECT

TIMING DATA
1986
15 OBSERVATIONS

PROCESS	NODE ASSIGNMENTS				TOTAL WAIT TIME (SEC)				TOTAL CALCULATION TIME(SEC)			
	1	2	3	4	1	2	3	4	1	2	3	4
INTEGRATOR	1	1	1	7	422	402	414	428	9.2	9.4	9.3	52.1
INTEGRATOR	2	2	2		422	400	412		9.4	9.6	9.5	
INTEGRATOR	3	3	3		422	405	395		7.1	7.1	18.1	
INTEGRATOR	4	4	4		422	400	422		8.1	7.1	2.7	
INTEGRATOR	5	5			434	410			1.5	1.7		
INTEGRATOR	6	6			438	409			0.0	1.3		
INTEGRATOR	7	7			434	418			1.3	0.0		
INTEGRATOR	8				437				0.0			
INTEGRATOR	9				437				0.0			
INTEGRATOR	10				437				0.0			
GRAVITY NOISE	12	1	1	7	425	396	409	449	3.1	7.8	7.9	24.5
GRAVITY NOISE	13	2	2		425	394	412		3.4	7.9	8.1	
GRAVITY NOISE	14	3	3		408	372	393		11.1	12.5	14.7	
DRAG NOISE	15	6	3	7	422	397	407	475	4.1	5.0	9.4	15.3
DRAG NOISE	16	7	4		421	392	410		4.4	3.7	6.4	
S. RAD. NOISE	17	1	1	7	422	395	408	475	4.5	13.9	13.9	14.6
S. RAD. NOISE	18	5	2		422	392	419		4.6	5.7	6.0	
CLOCK NOISE (GPS)	19	4	4	7	404	359	393	475	10.8	18.2	15.6	11.7
CLOCK NOISE (GRD)	20	2	3	7	401	396	394	477	10.9	18.9	16.8	13.4
RESIDUAL	11	3	21	21	426	387	415	496	2.7	6.4	4.1	3.9
UPDATE	21	21	21	21	345	329	343	423	75.4	74.9	74.9	75.1

TIME PER OBSERVATION (SEC)

	1	2	3	4
1	25	24	24	31
2	31	30	32	36
3	30	30	32	38
4	27	27	28	34
5	27	27	28	33
6	27	27	27	33
7	29	26	28	34
8	27	27	28	33
9	28	27	28	33
10	27	28	28	33
11	28	27	28	33
12	27	27	27	33
13	31	27	28	33
14	32	27	28	32
15	29	27	28	33

NOTE:
- SIGNIFICANT AMOUNT OF DEBUG DATA TRANSMITTED
- NO FLICK (50 MILLISECOND WAIT TIME AWAITING EACH MESSAGE)

TOTAL RUN TIME	425	408	422	503

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE 2

1987 HYPERCUBE EPHEMERIS PROJECT

TIMING DATA
1987
15 OBSERVATIONS

PROCESS	NODE ASSIGNMENTS 1	2	3	4	TOTAL WAIT TIME (SEC) 1	2	3	4	TOTAL CALCULATION TIME(SEC) 1	2	3	4
INTEGRATOR	1	1	1	7	33	34	47	65	9.2	9.4	9.3	35.0
INTEGRATOR	2	2	2		33	35	47		9.1	9.0	9.1	
INTEGRATOR	3	3	3		36	37	44		6.2	6.2	12.6	
INTEGRATOR	4	4	4		36	37	46		6.2	6.2	2.2	
INTEGRATOR	5	5			34	34			1.2	1.2		
INTEGRATOR	6	6			0.0	32			0.0	1.0		
INTEGRATOR	7	7			32	0			0.9	0.0		
INTEGRATOR	8				0.0				0.0			
INTEGRATOR	9				0.0				0.0			
INTEGRATOR	10				0.0				0.0			
GRAVITY NOISE	12	1	1	7	39	40	53	90	1.4	1.6	1.3	5.1
GRAVITY NOISE	13	2	2		39	40	53		1.6	1.5	1.4	
GRAVITY NOISE	14	3	3		39	41	53		2.4	2.4	2.4	
DRAG NOISE	15	6	3	7	38	40	53	90	2.0	1.9	1.9	3.9
DRAG NOISE	16	7	4		38	40	53		2.0	2.1	1.9	
S. RAD. NOISE	17	1	1	7	40	41	52	89	1.9	2.0	1.9	3.8
S. RAD. NOISE	18	5	2		40	41	52		2.0	2.3	1.9	
CLOCK NOISE (GPS)	19	4	4	7	40	41	53	93	0.8	0.8	0.7	0.7
CLOCK NOISE (GRD)	20	2	3	7	40	41	54	93	0.8	0.8	0.8	0.8
RESIDUAL	11	3	21	21	40.5	42	53	94	1.2	1.4	1.2	1.2
UPDATE	21	21	21	21	31	32	49	84	7.2	7.3	7.3	7.3

	TIME PER OBSERVATION (SEC) 1	2	3	4
1	5	3	4	
2	2	2	3	
3	3	2	4	
4	3	2	3	
5	2	2	3	
6	2	2	3	
7	2	2	3	
8	2	2	3	
9	4	2	4	
10	2	2	3	
11	2	4	3	
12	2	2	3	
13	2	2	3	
14	2	2	3	
15	2	2	4	

NOTE:
- OBSERVATION FILE REFORMATTED PRIOR TO CYCLING
- GLOBAL BROADCAST TREE INCORPORATED

TOTAL RUN TIME	37	33	49	94

TABLE 3

EPHEMERIS PROCESSING IN PARALLEL PROCESSORS
IR&D STATUS REVIEW

RESULTS OF EXECUTION TIME

- PROPORTIONATE REDUCTION IN EXECUTION TIME WITH INCREASING NUMBER OF NODES EXPECTED

NUMBER OF NODES UTILIZED IN TEST CASE	21	8	5	2
RUN TIME (JAN 1987) (SEC)	425	408	422	503
RUN TIME (JUNE 1987) (SEC)	37	33	49	94

TABLE 4

EPHEMERIS PROCESSING IN PARALLEL PROCESSORS
IR&D STATUS REVIEW

LOOK AHEAD TECHNIQUES

METHOD 1 (PRESENTLY EMPLOYED)

- MAINTAIN STATES AT SAME EPOCH BY RESTARTING INTEGRATION OF ALL SATELLITES AT THE TIME OF OBSERVATION OF ANY SATELLITE
- SOLUTION EXACT BUT ALL SATELLITE INTEGRATIONS ARE STALLED FOR UPDATE CALCULATION

METHOD 2

- ALLOW STATES OF DIFFERENT SATELLITES TO HAVE DIFFERENT EPOCHS FOR THE TIME OF UPDATE
- RESTART OF INTEGRATION ONLY AT RESPECTIVE OBSERVATION TIMES
- INTEGRATION FOR EACH SATELLITE MUST AWAIT ITS OWN UPDATE
- NET RESULT MAY BE APPROXIMATE DUE TO PROCESS NOISE CORRELATIONS
- FOR TRAJECTORY/COVARIANCE OUTPUT, RESTARTS ARE NECESSARY DURING LONG OBSERVATION GAPS FOR ANY GIVEN SATELLITE

METHOD 3

- BATCH SEQUENTIAL
- SPECIFIED BATCH LENGTH, SAME EPOCH FOR ALL SATELLITES
- RESTART AT OBSERVATION TIMES ONLY IF UPDATE PARAMETERS EXCEED PROPOGATED STATE BY SOME TOLERANCE

METHOD 4

- BATCH SEQUENTIAL
- MINIMUM AND MAXIMUM BATCH LENGTH SPECIFIED
- DIFFERENT EPOCHS FOR DIFFERENT SATELLITES
- MAXIMUM BATCH LENGTH DEFINED AS INTERVAL BETWEEN OBSERVATIONS OF RESPECTIVE SATELLITES

APPENDIX

ALGORITHM TASKS

EPHEMERIS

OBJECTIVE
The objective of the hypercube ephemeris task was to decide if parallel processors can be used effectively to determine the orbits of satellites.

APPROACH
Within a satellite ephemeris computer program, there are many vector-type operations that could be performed in parallel and, thus, improve the throughput of the computations. However, exploiting this capability of parallel or vector processors would require a large number of processors; furthermore, the results of such a study would be highly dependent on the type of computers used. A general study was selected in which the major blocks of computation for multisatellite orbit computations were used as the units to be assigned to various processors. A multisatellite orbit solution including observations between satellites is a challenging problem for parallel processors, since there is a natural bottleneck that occurs in the updating of the parameters of such a solution when process noise is used to represent unmodeled errors. Problems encountered or successes achieved in addressing this problem are more likely to be transferrable to other computers.

TEST CONDITIONS
A typical multisatellite test problem was selected which consisted of the configuration shown in Table 1. The program that was designed has the capability of processing the above observations for 3 primary satellites, such as Landsat or Topex, 3 relay satellites, and 18 GPS satellites. The number of Doppler stations can be greater than the 15 selected for the test, but provision was not made for time-overlapping Doppler observations since it would not have contributed to the test objectives. The process noise models account for the statistical effects of atmospheric drag variations and unmodeled errors in the earth's gravity field, computed effects of solar radiation forces, and clocks aboard the primary satellites that are used to make measurements of range to the GPS satellites or Doppler effects seen at ground stations.

The processor modules shown in Table 2 consist of:

1. An executive for each node

2. Integrator-force assignable to nodes for any groupings of satellites

3. Residual computation assignable to nodes for any groupings of observation types and satellites

4. Process noise for gravity assignable to nodes for any grouping of host vehicles and relay satellites

5. Process noise for drag assignable to nodes for any grouping of host vehicles

6. Process noise for solar radiation pressure assignable to nodes for any groupings of relay satellites

7. Process noise for clocks assignable to nodes for any groupings of host satellites

8. A single time and observation update module

The controller receives input assigning the processes to nodes, initializes the computations, and sends extended observation messages to the appropriate nodes where the executive (on the basis of the codes contained in the observation record) determines which processes are to be performed on the respective node and where to send the results. As the current solution is performed, the update module sends it to the controller for output, and this signals readiness for another observation. An IBM 3090 program supports the system by generating simulated data which is down-loaded to the hypercube controller.

DESCRIPTION OF TEST CASES

The orbit computations were performed for 15 simulated observations using the node assignments shown in Tables 3 and 4. All the processes were loaded on each node except for Update which was loaded on node 21 with no other processes (excluding the node executive which was common to all nodes). Test 4 node assignments were selected to approach the computer run time expected for sequential processing. Tests 2 and 3 provide measures of gain to be achieved in parallel processing. Of course, in actual implementation, the processes would be decomposed into smaller elements in order to make maximum utilization of available nodes. Test 1 was designed to determine the approximate computation time required for each process. Although the length of time spent in the computation portion of each process was recorded, it included time spent during the 50-msecond samplings of other processors. Although the exits to the node executive were included in the process times obtained in Test 1, the results were as close to the actual computation time as could be obtained. Refer to Table 5 for memory requirements for processors.

TABLE 1

TEST CONDITIONS

Satellites:

Landsat Mapping Satellite

Topex Altimetry Satellite

Tracking and Data Relay Satellite System (TDRSS) -1 Relay Satellite

TDRSS -2 Relay Satellite

6 Global Positioning System (GPS) Satellites (Orbits Assumed to be Known)

Observations:

Ground Doppler Observations of Landsat and Topex

Range Observations From a Ground Site to TDRSS Satellites

Range-Sum Observations Through Relays to Landsat and Topex

Range Observations From Topex and Landsat to GPS

Process Noise:

Gravity for Landsat, Topex and Relays

Drag for Landsat and Topex

Radiation Pressure for Relays

GPS Receiver Clocks on Landsat and Topex

Doppler Beacons on Landsat and Topex

TABLE 2

PROCESSOR MODULES

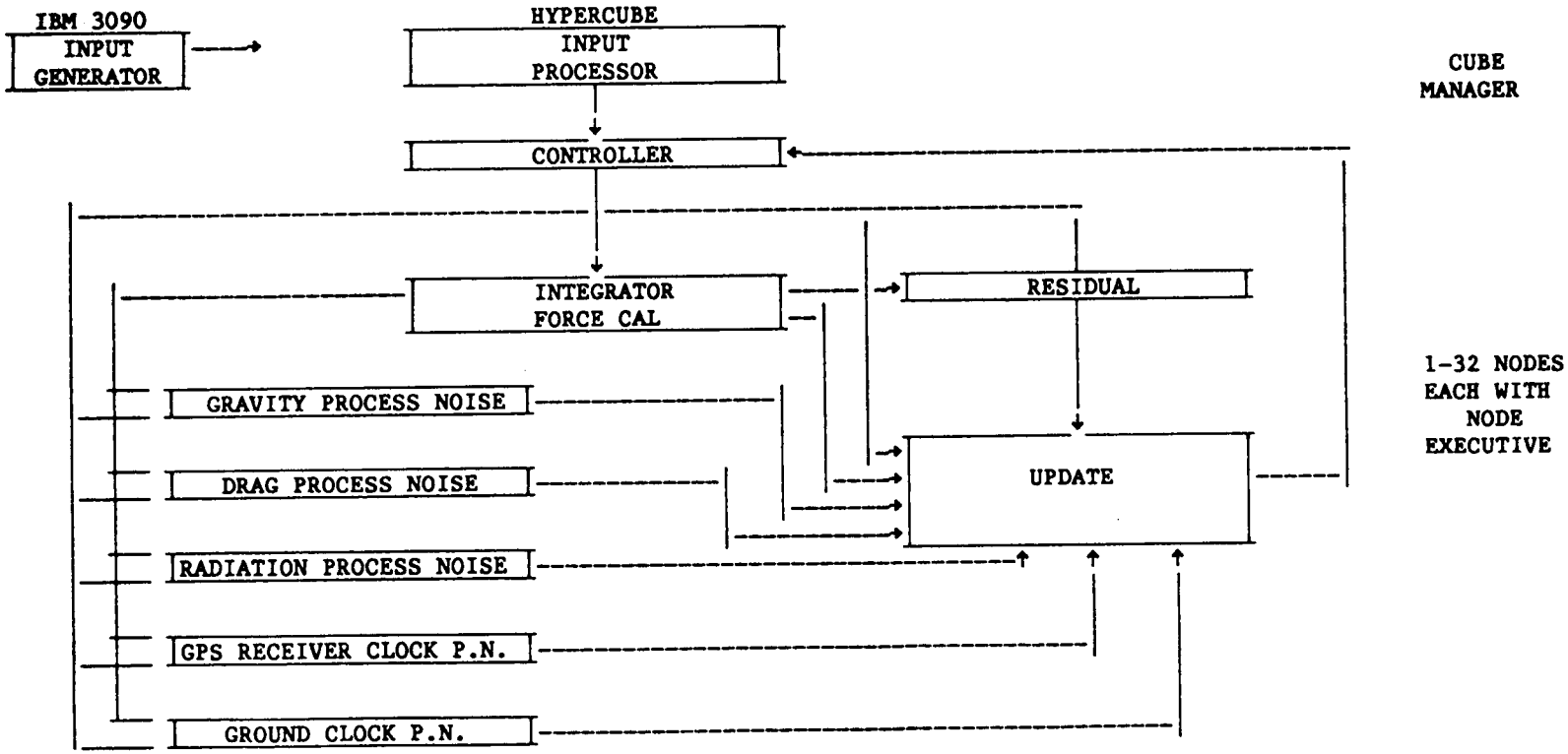

TABLE 3
NODE ASSIGNMENTS - INTEGRATOR/FORCE

Processor	Test 1	Test 2	Test 3	Test 4
Integrator/Force				
Landsat	1	1	1	7
Topex	2	2	2	7
Relay-1	3	3	3	7
Relay-2	4	4	3	7
GPS-1	5	5	4	7
GPS-2	6	5	4	7
GPS-3	7	6	4	7
GPS-4	8	6	4	7
GPS-5	9	7	4	7
GPS-6	10	7	4	7
Residuals	11	3	21	21

TABLE 4
NODE ASSIGNMENTS - PROCESS NOISE

Processor	Test 1	Test 2	Test 3	Test 4
Process Noise				
Gravity P.N. Landsat	12	1	1	7
Gravity P.N. Topex	13	2	2	7
Gravity P.N. Relay 1	14	3	3	7
Gravity P.N. Relay 2	14	3	3	7
Drag Noise Landsat	15	6	3	7
Drag Noise Topex	16	7	4	7
Rad. Noise Relay 1	17	1	1	7
Rad. Noise Relay 2	18	5	2	7
Clock 1 Landsat	19	4	4	7
Clock 1 Topex	19	4	4	7
Clock 2 Landsat	20	2	3	7
Clock 2 Topex	20	2	3	7
Update	21	21	21	21

TABLE 5
MEMORY REQUIREMENTS FOR PROCESSORS

Process	Memory Requirement (Bytes)	
	IBM 3090	Hypercube
Node Executive	N/A	47769
Integrator/Force	175005	42251
Residuals	74688*	3185*
Gravity Noise	67848*	26935
Radiation Noise	72168*	34349
Clock (GPS)	55144*	14277
Clock (Ground)	55984*	14577
Update		231988*

* Combination of these processes in one processor would reduce the storage requirements for these five routines from 398000 to 174000 bytes through the use of shared subroutines and common data.

** This figure is the memory requirement for the full computational Update. For test purposes, an abbreviated Update was used. The test version required 182825 bytes.

RESULTS

The execution times initially obtained for the previous test cases are given in Table 6. Supplementary timing data showed that the small reduction in run time with increase in the number of nodes was accompanied by large wait times (100-600 seconds) on the nodes and large execution time (75 seconds) for update. The best estimate of the actual computation time (as opposed to execution time, which includes the wait time) for sequential processing is that obtained for Test 4, which gave a value of 211 seconds, including 75 seconds for Update computations, or a computation time of 135 seconds for all processes except Update. Using this value as a base, the differences in execution time for the other test cases were used to estimate the computation

times for those cases. The measured computation time for the processors in those cases could not be used because the time measured within each processor also included time spent in other processors on the same node during the 50 millisecond sequencing among processors. The computation time for Update was considered separately for the comparisons, since proportionate reduction in computation time with increasing number of nodes would not be expected for Update, which was on a separate node in each case.

The reduction in estimated computation time from Test 4 to Test 3 to Test 2 is within a factor of two of that expected for the increase in number of nodes. Proportional reduction in execution time is not expected for Test 1, since the processor assignments required that several of the nodes operate in sequence in this case; however, the increase in execution time is anomalous.

Although the estimated reduction in computation time with increase in number of nodes was satisfactory, the excessive wait time for the nodes is not acceptable. The second major concern was the extent of the time required for update processing. The time was particularly disappointing because the matrix operations required for an actual update were bypassed during these tests to expedite test results which were more pertinent to the objectives of the test. The initial test results were obtained while a significant amount of debug data was being transmitted from the nodes to the system log. They were also made without the benefit of the use of the Flick command which prevents

TABLE 6
NODE LOADING COMPARISONS

	Test 1	Test 2	Test 3	Test 4
1. Observed Results				
Number of Nodes Used	21	8	5	2
Execution Time (Seconds)	425	408	422	502
2. Estimated Calculation Time Excluding Update				
Number of Nodes Used	20	7	4	1
Estimated Calculation Time (Seconds)	58	41	55	135

C . 3

an unnecessary 40-millisecond wait time in a processor that is awaiting information each time the node sequences through the processor. A rerun of the test was made with the debug communication deleted and with the Flick command installed in the node executives (but not in the processors, which were thought to be of lesser concern). The execution time was nearly halved with these modifications, and the computation time reduced to a third of the original value for Test 4. The computation time for Update was reduced in order of magnitude to about 7 seconds. However, the computation time did not decrease with an increase in the number of nodes; in fact, the execution time increased slightly.

The cause of the failure of the execution time to decrease significantly with increase in number of nodes has not been specifically identified. The timing data accumulated to date fails to account for more than half the wait time recorded by the processors. In addition to the difficulty of interpreting timing results for a sequencing node, reruns of the same test case occasionally gave different results. A rerun of the test cases with the revised INTEL operating system would resolve that question, or additional timing data installed in the processors would identify the source of the unexpected wait time.

There are two modifications to the existing hypercube ephemeris program that would have a dramatic effect on the efficient utilization of the processors, even after the cause of the current anomalous results is identified:

1. The integration/force computations can be separated and performed in different processors. Since the integration and force computation for a given satellite are essentially sequential operations, the nodes with either an integration or force processor must also be assigned other processes, in order that the gain in efficiency can be realized.

2. Computations for some satellites can proceed ahead of the update computations, which would allow additional parallel computations to be performed. This modification would require some revision of the update algorithm and some additional logic in the controller and node executive.

CONCLUSIONS

The test results failed to show an improvement in efficiency of ephemeris computations with an increase in the number of nodes because of unexplained wait times occurring during execution. It is expected that additional testing would reveal the cause of the unexpected wait times, and tests with a modified program would demonstrate that ephemeris programs could be run efficiently on parallel processors.

RECOMMENDATIONS

It is recommended that:

1. The test cases be rerun with the latest INTEL operating system

2. Test 1 be rerun with additional timing data recorded to determine the cause of the unidentified wait times

3. The current processor modules be further subdivided, particularly by separating the integration and force computations

4. The update and integrator processors be modified to hold the epoch of the states fixed for scheduled periods of time, and the controller and node executives be modified to allow observation to be processed at controlled intervals ahead of the observation time for the last update

5. Studies and tests be conducted to develop an algorithm for automatic assignment of processors to nodes as a function of the available number of nodes and the nature of the ephemeris task