10/
3-28-90 98-①

SANDIA REPORT

SAND89—2534 • UC—705
Unlimited Release
Printed January 1990

Implementation of Midcourse Tracking and Correlation on Massively Parallel Computers

Accomplishments for Fiscal Year 1989

Ronald D. Halbgewachs, James L. Tomkins, John P. VanDyke

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

SF2900Q(8-81)

SAND--89-2534

DE90 008166

SAND89-2534
Unclassified
UC-705

Unlimited Release
Printed January 1990

Implementation of Midcourse Tracking and Correlation on Massively Parallel Computers

Accomplishments for Fiscal Year 1989

Ronald D. Halbgewachs
Advanced Systems Development Division

James L. Tomkins
Parallel Computing Science Division

John P. VanDyke
Parallel Computing Science Division

Sandia National Laboratories
Albuquerque, NM 87185

ABSTRACT

Sandia National Laboratories joined with two other laboratories, Los Alamos National Laboratory and Naval Research Laboratory, to study and implement a highly parallelized tracker/correlator algorithm. Significant progress was made at Sandia on a specific algorithm and code. This report summarizes the accomplishments by Sandia during FY '89 on this project.

The work on this project is supported by the Strategic Defense Initiative Office of the Department of Defense and the Department of Energy under Contract No. DE-AC04-76DP00789.

CONTENTS

A. EXECUTIVE SUMMARY

A joint effort was started among three laboratories (Sandia National Laboratories, Los Alamos National Laboratory, and Naval Research Laboratory) to implement a highly parallelized tracker/correlator algorithm for the Strategic Defense Initiative. Since April, 1989, Sandia has focused on the parallelization of a specific serial tracker/correlator algorithm (referred to as TRC) for post-boost and mid-course tracking. The NCUBE Hypercube (1024 processors with distributed memory) was to be the hosting system at Sandia.

The TRC code was transported to a CRAY X-MP 416 computer for detailed analysis and study utilizing the tools available on this system. This implementation was also made to permit larger test problems to be run. Memory requirements and the data organization of the TRC were a major concern for the parallel implementation.

Two major breakthroughs on the TRC have been accomplished by Sandia. Both greatly affect the serial version of the TRC and at the same time bring the parallelization problem within reach.

The most important was the tremendous reduction in the amount of memory required to run the TRC. Sandia's effort has taken the requirement, based on problem size, from an $O(n^2)$ memory allocation to a linear memory requirement of $O(cn)$, $c < 1$. For example, a test scenario of 16 Post Boost Vehicles each carrying 9 Reentry Vehicles (160 objects total), receiving reports from 16 sensor platforms on nonuniform 10 second reporting intervals, for a total scenario time of 431 seconds, originally required 18.3 Mbytes of memory. (This reported amount was actually from a version already modified by Sandia to eliminate a code problem. An earlier test of an unmodified version, run at another site, required 60 Mbytes of memory.) The new Sandia version executes the identical problem in 1.4 Mbytes. With these changes, Sandia was able to successfully run the full range of test scenarios set out for the parallelization group, including the largest scenario of 2000 objects.

The second breakthrough was in the complete elimination, without loss of program functionality, of a costly input/output (I/O) sequence of operations originally contained in the TRC. Since the hypercube has distributed memory, large amounts of data file I/O would result in time consuming communications between processors. From these changes, the amount of time required to run Sandia's serial version nearly halved the running time for the original modified version.

Using these results, work continues on the parallelization of the TRC algorithm at Sandia. Implementation has begun on the distributed node processors of the NCUBE. Target track generators are also being implemented on the hypercube. Additional studies have been started on other algorithms to better understand the problems associated with multiple sensor, multiple target tracking and correlation.

B. TECHNICAL REPORT

The following report describes the technical work performed by Sandia National Laboratories, Parallel Computing Science Division, on the implementation of midcourse tracking and correlation for massively parallel computing during the 1989 fiscal year.

1.0 Project Introduction

Since April 1, 1989, Sandia National Laboratories has been actively working on a project to examine and implement a highly parallelized tracker/correlator algorithm for the Strategic Defense Initiative. Sandia's principal efforts during this six month period have been to focus on the parallelization of a specific serial tracker/correlator algorithm (TRC) for post-boost and mid-course tracking. An intensive analysis of the algorithm and code was undertaken and the results of this work are described in this report.

1.1 Background

In December of 1988, a meeting was held at Sandia National Laboratories to initiate a cooperative effort among researchers and practitioners of parallel computing to study parallelism of tracker/correlator algorithms for a variety of parallel architectures. The result of this meeting was a collaboration among three laboratories (Sandia National Laboratories, Los Alamos National Laboratory, and Naval Research Laboratory) with active research and development programs in parallel computing. The Naval Research Laboratory would be assisted under contract by Ball Systems Engineering and D. H. Wagner & Associates. Coordination of these groups would be through SDIO/POET.

Each of these laboratories has a strong mission interest in high performance computing. Consequently, each had existing research and development programs to study highly parallel approaches for solving large scientific problems. At Sandia, this project had access to a 1024-processor NCUBE Hypercube and a four-processor CRAY X-MP; at Los Alamos, it had access to an eight-processor Alliant and a 24-processor Encore; and at NRL, there was access to a 128-processor BBN Butterfly. By collaborating and sharing computing resources, the project has been able to highly leverage the staff and resources working on the tracking problem.

To begin the project, each of the laboratories were supplied a complete software source code tape of the TRC. Two reports, one detailing the algorithm design [B&W 1] and the other describing the algorithm implementation [B&W 2], were also provided for a common starting point of reference.

1.2 Program at Sandia National Laboratories

Support for Sandia's program has been through the Electronic Systems Division of the U. S. Air Force. The remainder of this report covers only the work that has been done specifically at Sandia National Laboratories. It must be emphasized that throughout this reporting period, there has been excellent cooperation among the groups in the laboratories working on this project. All of the efforts put forth have been aimed at producing a better product and there has been cooperation in sharing information and planning for the project.

Preliminary studies in the area of target tracking included review of major references in the field, e.g. [Blackman 86] and [Bate 71]. A course in Multiple Target Tracking [Drummond 88a] was also attended. Parallel methods developed for other projects as well as general expositions of parallel algorithms have been drawn on, e.g., [Gustafson 88 & 89], [Bertsekas 89], and [Fox 88].

2.0 Project Objective

The overall objective of this project has been to study parallelism for tracker/correlator algorithms using a variety of parallel computer architectures. Specifically for FY '89, the objective has been to investigate the tracker/correlator developed for the Naval Research Laboratory by Ball Systems Engineering and D. H. Wagner & Associates, and to develop a highly parallel version of the TRC algorithm.

Sandia has taken major steps toward accomplishing the parallelization of the TRC algorithm as will be described later in this report. Through this effort and the current studies of other tracker/correlator algorithms (described near the end of this report) Sandia has made proper progress toward the overall objective.

3.0 Tracker/Correlator (TRC) Status

From the start of this project at Sandia, it was recognized that to put a parallel version of the TRC on the NCUBE the algorithm and implemented code were, out of necessity, going to be very closely re-examined. Since the NCUBE does not have shared memory into which one can simply place large arrays, the data structures and data organization were critical. One of the time consuming parts of using individual processors with only local memory is the communication necessary between processors. Therefore, both time and memory are primary controlling and critical factors in analyzing the algorithm.

Many times there is a trade-off between memory and execution time. That is, when one attempts to optimize memory usage there is often a penalty to be paid in the time required to solve the problem. However, the work accomplished on the TRC at Sandia has, in fact, resulted in a tremendous improvement in the memory requirements of the serial version and at the same time, produced improved execution time of the serial TRC.

3.1 Approach

Initially, it was agreed by the project group that a parallel version of the serial TRC should be implemented on all of the systems indicated in (1.1) with minimal changes to the methods and structure of the serial version. Following that work, a truly parallel version of the TRC algorithm would be developed to optimally exploit the MIMD (Multiple Instruction Multiple Data) parallelism in distributed memory computers such as hypercubes. Finally, the project would develop new highly parallel tracker/correlator algorithms and software.

At Sandia the project currently stands between task one and task two described above. The general approach taken at Sandia in addressing this project has been careful and systematic. The team studied the algorithm closely and gained an understanding of how the data structures impacted the execution of such an algorithm on distributed processors. It was believed that concurrency could be attained by dividing the problem down into cluster processing on each processor. Since multiple clusters could be handled in the serial version, should there ever be more clusters than processors, then extra clusters would simply be assigned to processors in a modulo fashion. That is, simply start back at the first processor for the next round of cluster assignments, proceeding to the second, and so on until all clusters were assigned. As clusters split, a new cluster would be assigned to the next available processor.

Each of the NCUBE/Ten processor nodes had a relatively small amount of memory, 512 Kbytes. Part of that memory would be used by the executable TRC program, the operating system, and the interprocessor communication buffer. Thus, TRC array storage became a major factor in the NCUBE implementation. It was also noted from serial executions of the code that the number of final objects tracked by the TRC could be considerably larger than the truthfile objects. While this problem has not been fully resolved and there will naturally be some differences in the number of objects due to sensor inaccuracies, some aspects specific to the TRC are understood and will be discussed in later sections.

3.2 Summary of Task Milestones for FY'89

When Sandia submitted its proposal for work on this project, certain points were made regarding the tasks to be performed. This section will briefly respond to those points and then each of the items noted will be expanded upon in the sections to follow.

Sandia has carried out an intensive analysis and review of the TRC algorithm and implementation. A number of errors in the original implementation were discovered and corrected. Of major importance has been the changes made to the code in computer memory utilization and the elimination of a linked history file that required very large amounts of execution time simply for data movement. Although Sandia considers most of this analysis to be completed, there are still areas requiring some study depending upon new discoveries about code/algorithm behavior.

Work is currently ongoing to produce a version of the TRC on the hypercube nodes. An early, stripped-down version of the TRC was transported to the host processor on the hypercube. However, the performance of this code was poor since that host was a micro-processor with less power than many current, desk-top systems. (Note that an NCUBE2 hypercube, soon to be installed at Sandia, will have a SUN 4/280 workstation as its host.) There still remains work to be done to reach the full implementation on the NCUBE hypercube.

Joint development of a set of test problems for this project has proven to be beneficial to all participants. These test problems permit the participants to study scalability on each of the different machines and also to examine how clustering and separation of objects affects correlation. Three of the test cases were sufficiently large to eliminate many host systems that might be considered to run these problems: (a) 8 PBVs, 9 RVs/PBV, (b) 16 PBVs, 9 RVs/PBV, and (c) 20 PBVs, 99 RVs/PBV. For example, Sandia first executed the TRC on a SUN 3/140 workstation and had little trouble with small problems (~10 objects). Much beyond that size problem, the system exhausted its memory space.

3.3 Implementation on Different Systems

It was apparent that a larger machine would be helpful to study the serial performance and behavior of the TRC. The CRAY systems available at Sandia, CRAY 1-S and CRAY X-MP 416, provided a good set of analysis tools and the necessary computing power to extend the range of test scenarios. Particularly with the X-MP, significant benefit could be realized by all investigators of mid-course tracking. The space target tracks and sensor platform generators (SPACETRACKS and SPACEGEN) were fully implemented on the CRAY systems so test scenarios could be generated directly on the same machine used to execute the TRC. Major improvements in the serial version of TRC have been accomplished. In fact, without the changes made, the CRAY 1-S was unable to execute the data set of 2 PBVs, 9 RVs/PBV.

Use of a SUN workstation for the modified TRC will still be maintained as an option since it is a way for Sandia to obtain any future updates for the TRC. As noted above, within a few months a SUN 4/280 will be installed as the NCUBE host. Data entry could then be made directly to the NCUBE from any SUN system.

3.4 Major Breakthroughs

Two major breakthroughs on the TRC have been accomplished by Sandia. Both greatly affect the serial version of the TRC and at the same time bring the full parallelization problem within reach. Although both breakthroughs were fairly complex to develop and implement, in the end, relatively few lines of code needed to be modified. For the discussion that follows, a target map is defined to be composed of the current position, velocity, and covariance matrix description for each target object.

The first breakthrough was to overcome the tremendous amounts of memory storage required to run the TRC and it came in two stages. The first stage was the correction of an error in the way that a binning array was initialized for map merging. Without this correction, even small test scenarios pushed the memory limits of most computers. The memory requirements shown in Table 1 are for the original TRC code <u>with this error corrected</u>, but without the second part of the memory changes that altered the program arrays. Without this error correction, most of the problems shown could not have been run. (For example, an earlier test run at another site of the 16 PBVs - 9 RVs problem required 60 Mbytes of memory.)

The second stage of this breakthrough occurred when it was understood how much array space was unused in the TRC implementation. The redesign of several arrays eliminated this sparse representation of data. For the discussion at this point, the best way to describe the changes are by comparing the results in Table 1 with those of Table 2. Note that Sandia's effort (shown in Table 2) has significantly reduced the memory requirement from an $O(n^2)$ problem to a linear memory requirement $O(cn)$, $c < 1$. With these changes, Sandia was able to successfully run the full range of test scenarios set out for the parallelization group, including the largest scenario of 2000 objects.

All problems in Table 2 were run for the full time prescribed for the scenarios. Handover took place at 243 seconds after first launch and tracking/correlation continued until 674 seconds, for a total time of 431 seconds. All tests were performed with 16 sensor platforms (8 infrared and 8 radar) each reporting on 10 second intervals as designated in the test scenarios. For Table 1, only two problems were fully run, while the memory requirements were determined for the other problems but not run to completion. There was no attempt to go beyond those problems shown in Table 1.

These results have been summarized in graphic form in Figure 1. Memory requirements for both versions of the TRC code have been plotted for comparison. In this format, the great differences become most evident. Figure 2 details the linearity of Sandia's version for the leftmost corner of Figure 1 and emphasizes the continuity of the linear curve.

Problem	Objects	CPU secs.	Memory CRAY words	 ~ bytes
1 PBV - 9 RVs	10	---	---	---
2 PBVs- 9 RVs	20	42.3	326,144	1,305,000
4 PBVs- 9 RVs	40	77.1	610,816	2,443,000
8 PBVs- 9 RVs	80	---	1,445,120	5.8 M
16 PBVs- 9 RVs	160	---	4,579,072	18.3 M

Table 1. TRC Performance Using Corrected Original Implementation

Problem	Objects	CPU secs.	Memory CRAY words	 ~ bytes
1 PBV - 9 RVs	10	9.4	141,056	564,000
2 PBVs- 9 RVs	20	26.8	170,496	682,000
4 PBVs- 9 RVs	40	46.8	189,952	760,000
8 PBVs- 9 RVs	80	92.7	243,456	974,000
16 PBVs- 9 RVs	160	217.3	359,424	1,438,000
1 PBV -99 RVs	100	133.6	276,736	1,107,000
99 PBVs- 0 RVs	99	242.7	226,304	905,000
20 PBVs-99 RVs	2000	5034.4	3,471,104	13.9 M

Table 2. TRC Performance with Sandia Modifications

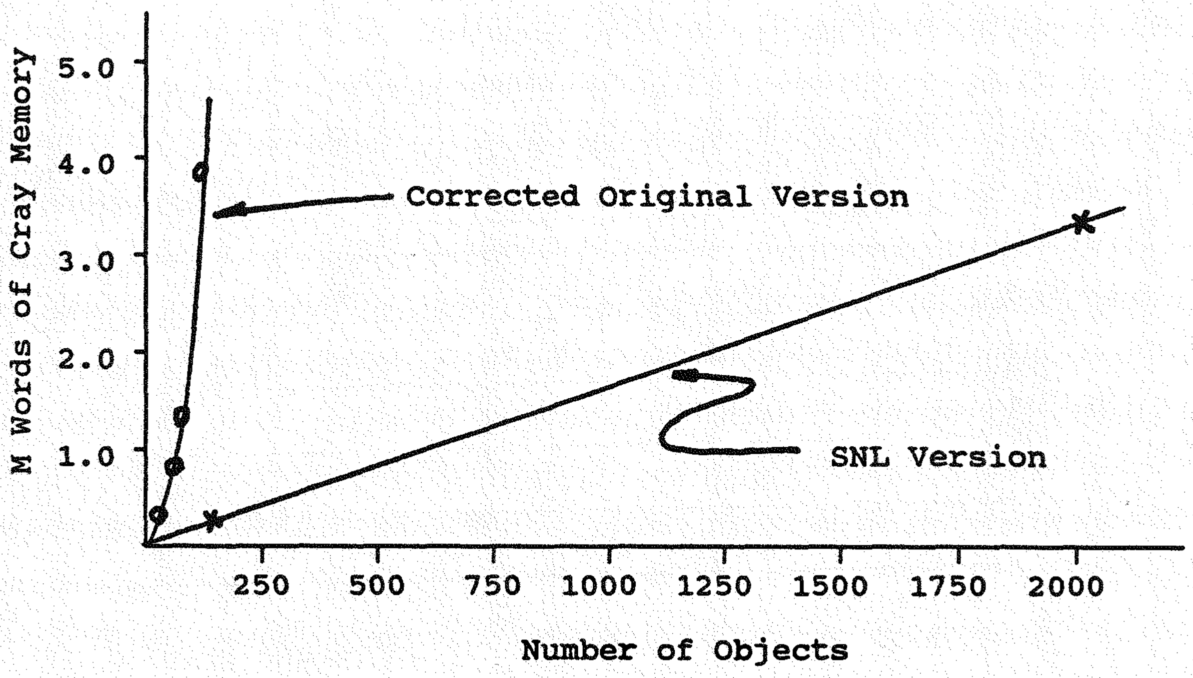


Figure 1. Memory Comparison between SNL and Corrected Original TRC Versions

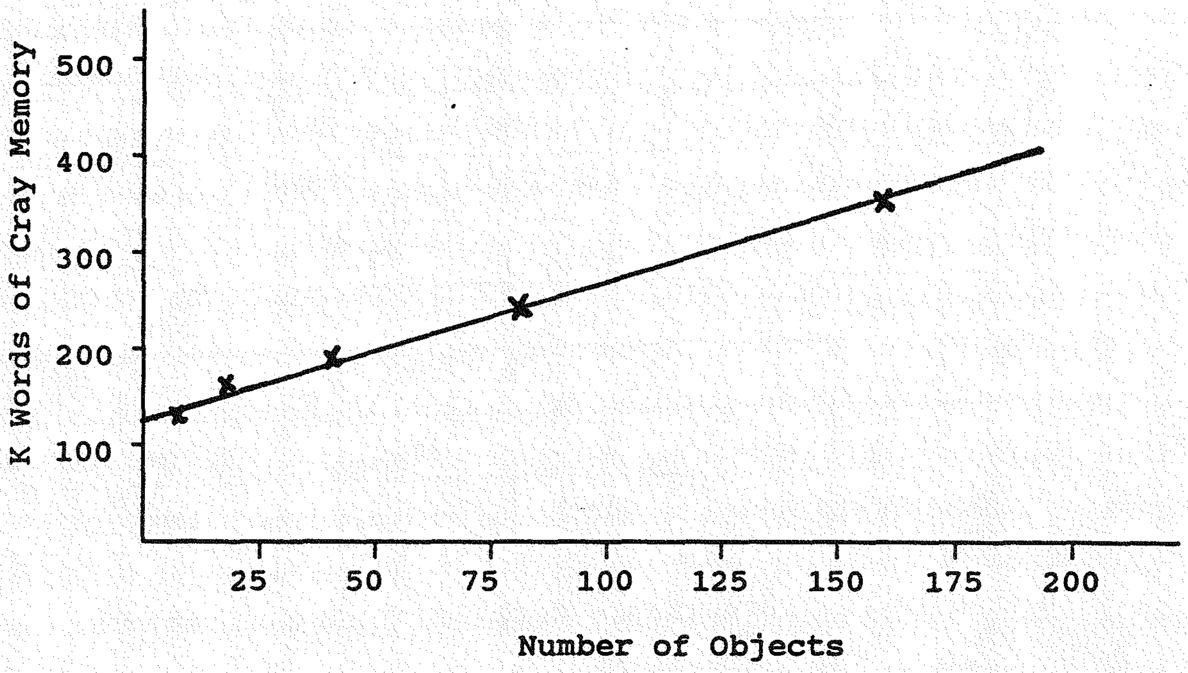


Figure 2. Linearity of Memory for Sandia Version of TRC

The second major breakthrough was the complete elimination of a costly sequence of input/output (I/O) file operations. A linked history file was maintained to relate target maps with sensor reports for Constructed Track Complex assignment and also to maintain map/report information for map merging. This file was continually accessed throughout the entire run of the TRC. These changes significantly reduced the amount of time required to run various scenarios. The improved times shown in Table 2 compared to Table 1 are primarily a result of changing I/O requirements for the TRC. Figure 3 compares the timing results extracted from the tables. Based on the results obtained, the Sandia version appears to have cut the execution time nearly in half.

From Figure 3 additional observations can be made. First, the timing curve for the current Sandia version is not quite linear, but on the order $O(n + an^2)$ with the coefficient, $a < 1$. Results from running the original TRC are too few to significantly plot, but timing results reported from other sites indicated non-linear growth also. That is, the curve shown in Figure 3 also moves upward away from a straight line at a much steeper slope. Second, the timing for one problem, 99 PBVs - 0 RVs, is well above the curve for the other problem sets. Both observations have their genesis in a pair of $O(n^2)$ operations within the TRC. As currently designed, the report-to-cluster assignment operation is a function of (number of reports*number of clusters). An area to be addressed is the improvement in preprocessing cluster selection. When cluster splitting occurs, this assignment operation increases in magnitude. The report-to-maps assignment is a product operation also, but is actually capped by the number of maps in a particular cluster. Therefore, the total report-to-map assignment will usually play a lesser role.

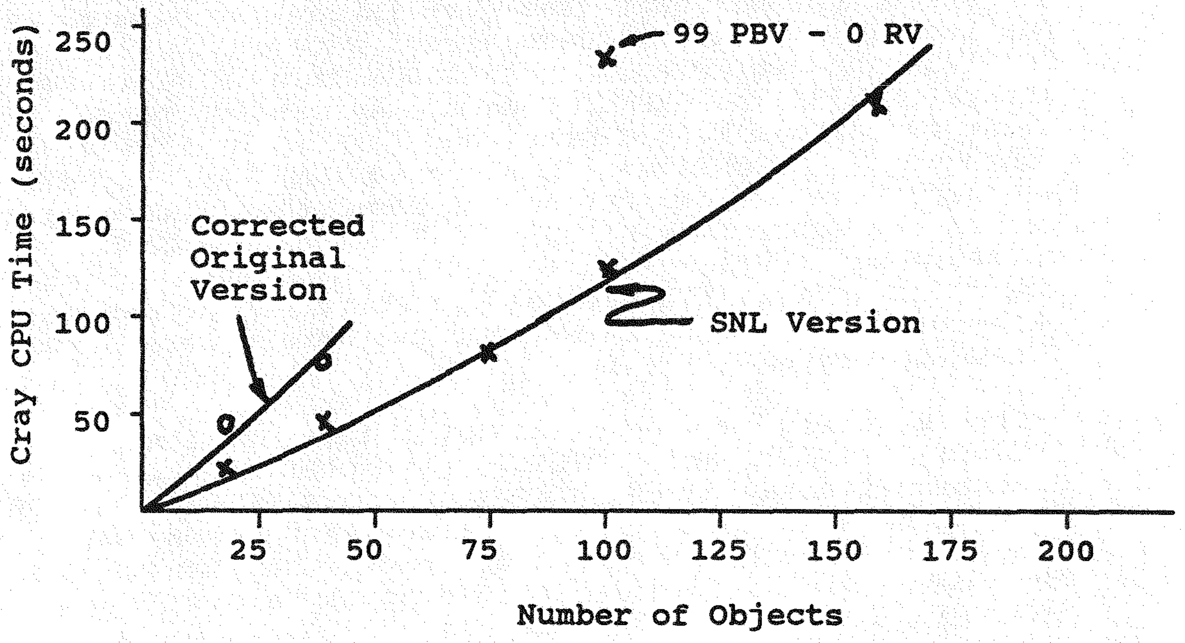


Figure 3. Time Comparison between SNL and Original TRC Versions

4.0 Modification Details for the TRC

Previous sections of this report have provided high-level information about a number of changes that have been made to Sandia's version of the TRC. Most of these changes were the result of determining how to implement the TRC on the NCUBE. Several changes were the result of investigating strange phenomena as they occurred and discovering what happened.

Not all of the material presented here resulted in changes, but rather addresses concerns that Sandia had with the TRC. This part of the report will provide details and specific discussions related to key modifications to the TRC and special areas of concern.

4.1 Memory Utilization Changes

A number of the multi-dimensional arrays in the TRC are sparse matrices, i.e., are filled primarily with zeroes. This situation exists regardless of the size of a particular problem being run and results in memory requirements for any given problem that are essentially a square function of the number of maps needed to run the problem. At this time two areas in which large amounts of memory can be saved have been identified. These are in the history id assignment process and in the Constructed Track Complex (CTC) output routine. Although there are other areas in which memory use can be improved, the above two areas were found to have the greatest potential for savings.

4.1.1 Changes to HISTASGN and GENNEAR

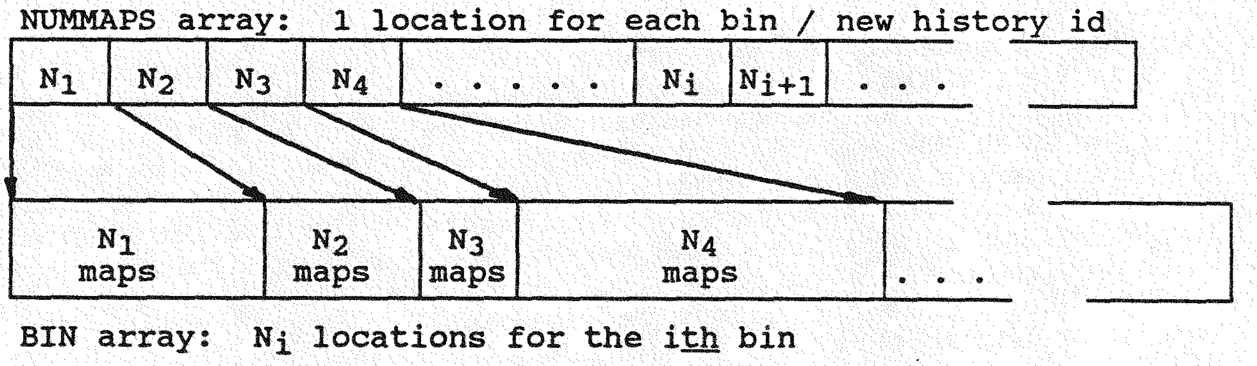
The subroutine HISTASGN used five large, two-dimensional arrays for storage. Three of these arrays were in the local common block PHISTBLK and two were in the include file <HIST>. Of these five arrays RPTHIST and PRPTHIST were only used for debug printout. These two arrays were dimensioned (MXSCAN by MXHIST + 1). The third array, REPORT, was dimensioned (NMAXCLIST by MXHIST + 1) and was used to store report numbers associated with a history id for use in the subroutine HISTASGN. The fourth array, PBIN, was used to store map addresses assigned to each previous history id and the fifth array, BIN, was used to store map addresses assigned to each new history id. These last two arrays were dimensioned (NMAXCLIST by MXHIST + 1).

In the modified version of HISTASGN three of the above arrays (REPORT, RPTHIST, and PRPTHIST) have been deleted as has a one-dimensional array named IDXSORT. Arrays RPTHIST and PRPTHIST were deleted because their debug function wasn't considered important enough to warrant the amount of memory they used. The information they contained was available in other arrays and, with appropriate indexing, could still be accessed.

The function of the REPORT array was incorporated in a redefined PBIN array and as a result, the REPORT array was no longer needed. This redefined PBIN array is dimensioned (4 by MXMAPS) where (MXHIST = MXMAPS) and for each map, M, it contains:

PBIN (1,M) = previous history id
PBIN (2,M) = new history id
PBIN (3,M) = report number, and
PBIN (4,M) = map address.

The information stored in the BIN array (map addresses) was not changed. However, the map addresses are now stored linearly and must be accessed differently. As shown in Figure 4, BIN (1) through BIN (I) now contain the map addresses of the maps associated with new history id, I, where I = NUMMAPS (1) and NUMMAPS is an array containing the number of maps associated with each new history id. The function of NUMMAPS has not been redefined. Similarly, BIN (I+1) through BIN (I+J) are map addresses associated with history id 2 where J = NUMMAPS(2), and so on for the remaining history ids.


NUMMAPS array: 1 location for each bin / new history id
N_1
N_2
N_3
N_4
. . . . .
N_i
N_{i+1}
. . .
N_1 maps
N_2 maps
N_3 maps
N_4 maps
. . .
BIN array: N_i locations for the ith bin

Address of first map in bin 4 is
MAPAD = BIN ($N_1 + N_2 + N_3 + 1$)

Figure 4. New Access to BIN Array Using Linear Indexing

The changes to the subroutine HISTASGN have not changed its function nor its basic structure. However, because maps associated with a cluster are not accessed sequentially by history id the new storage scheme results in the requirement that the PBIN array be sorted by previous history id for each cluster after the map data for the cluster has been stored in PBIN. In addition, the PBIN array is sorted by report number for each previous history id in a cluster. The second sort replaces the call to subroutine SORTINDX and the need for the array IDXSORT. Subroutine SORTINDX was replaced by a much simpler sort process because the number of reports associated with a single history id is relatively small. The earlier version had a greater overhead in calling the subroutine SORTINDX than the actual amount of time required to perform the sort.

During the process of modifying HISTASGN and as a result of trying to run the TRC code it was discovered that HISTASGN processed time-continued maps and included them in the maps sent to the subroutine GENNEAR through the BIN and NUMMAPS arrays. (There were some errors discovered in HISTASGN during this process that have been corrected. These errors are discussed later in the report in Section 4.4.) The new version of HISTASGN only sends to GENNEAR maps for which reports have been received since the last snapshot. This last change eliminated some unnecessary work in the map merging calculations.

All changes to GENNEAR involve indexing of the BIN array to retrieve and store data in a one-dimensional rather than two-dimensional manner. There have been no changes to the function or structure of GENNEAR.

The changes to the way memory was used in the subroutine HISTASGN have resulted in a huge reduction in its memory requirements. For example, total memory required by the REPORT, PBIN, and BIN arrays for an 8 PBV - 9 RV/D problem that required 330 maps was 326,700 CRAY words (SUN computer equivalent approx. 1,300,000 bytes or 1.3 Mbytes). The same memory requirement using the new version of HISTASGN became 1650 CRAY words (SUN computer equivalent approx. 6600 bytes). Even more significant than this specific savings was the fact that the memory requirement for HISTASGN now only grows as a linear function of the number of maps.

4.1.2 Changes to CTCOUT and CONVRT

Three arrays in the <NICTC> include file were identified as possible candidates for memory savings. Among them, they used nearly three-fourths of the total memory required to run scenarios. This factor results after the savings from changes to the subroutine HISTASGN were removed. Another reason for looking at these arrays was that they are non-linear functions of the problem size. The arrays are CTCMAPS, CTCRPT, and CTCRPTWT. The array CTCMAPS was used to store the map addresses associated with a Constructed Track Complex (CTC) and was dimensioned to (MXCTC by MXMAP) where MXCTC equals MXMAP. Arrays CTCRPT and CTCRPTWT were used to store the report addresses and report weights associated with each CTC. These two arrays were each dimensioned (MXCTC by MXRPT). Array CTCMAPS was used only in subroutine CONVRT and arrays CTCRPT and CTCRPTWT were used only in subroutine CTCOUT.

Subroutine CONVRT was only used to transfer data between common blocks for output to the network interface. Since Sandia is not connected to the network this entire subroutine was not needed. However, for the present the subroutine has been left intact except that the array CTCMAPS references have been commented out. The function of the CTCMAPS array could have been maintained by using two arrays of dimension MXMAP where one array held the number of maps associated with each CTC and the other held the map addresses. This is analogous to the way in which the BIN and NUMMAPS arrays were modified for the new version of HISTASGN and this kind of change would result in memory use that would be linear rather than as a squared function of the number of maps.

The arrays CTCRPT and CTCRPTWT were used in a loop over CTCs in subroutine CTCOUT. These arrays were not used outside this loop and they were constructed inside the loop before they were used at each snapshot for each CTC. In the modified version of CTCOUT these arrays have been made one-dimensional with the dimension MXRPT. As was described above for the array CTCMAPS, these arrays were part of the network interface and as such are not applicable at Sandia. However, it would be possible to retain the full capability by the addition of a one-dimensional array dimensioned to MXCTC. This array would contain the number of reports associated with each CTC and the report addresses and report weights would be stored linearly within the CTCRPT and CTCRPTWT arrays respectively.

The modifications to subroutines CTCOUT and CONVRT have resulted in large savings in memory requirements. For example, in the case of the test scenario with 8 PBVs - 9 RV/D resulted in a savings of more than 600,000 CRAY words of memory (equivalent of approx. 2,400,000 bytes or 2.4 Mbytes).

Altogether the changes to HISTASGN, CTCOUT, and CONVRT have reduced the total memory requirements for the test scenario used in the examples above from more than 1,600,000 CRAY words to about 300,000 words. In equivalent values this means a reduction from 6.4 Mbytes to 1.2 Mbytes. The major significance of this work is that these changes have resulted in the memory needed by the TRC code, based on problem size, to be improved from an $O(n^2)$ problem to one of $O(n)$.

4.2 Linked History File Changes

The linked history file was recognized by the group very early into the project as a potential problem area in the parallelization effort. The original version of TRC made heavy use of this file causing frequent input/output operations to take place. On a parallel system such as the NCUBE, this would have meant frequent I/O operations from code where I/O is indirect at best. The running characteristics of the code were analyzed on a machine, such as the CRAY, where I/O normally wasn't thought to be a particular problem. It was discovered that a very significant fraction of total run time goes to reading and writing the linked history file.

A linked history file entry was written from the subroutine FILLSCN each time a map is updated by incorporating information from a sensor report. The linked history file was read in four subroutines. In HISTASGN and CTCASGN the information examined was the scan number and the report number for the last relevant entry, i.e. the scan number was checked to determine if the linked history file was from the current scan. In CTCOUT and SNAPSHOT the information being retrieved after multiple reads was the map id at the previous snapshot, and the report numbers used to update the map at each scan since the last snapshot.

For the subroutines HISTASGN and CTCASGN use of the linked history file was very easy to replace. Changes to these subroutines to eliminate these file accesses were to add the sensor report number used to update this map to the integer map array IMAP and a variable that indicated the first linked history index in a scan was added to a common block. For this, changes were made to the include files <MAP> and <GENRL> and to the subroutines FILLSCN, CTAMAIN, HISTASGN, and CTCASGN.

The use of the linked history file in the subroutines SNAPSHOT and CTCOUT was to produce detailed ASCII printout of what happened. Elimination of the linked history file preserved this function exactly. An existing error (refer to Section 4.4 on Errors and Appendix A) was corrected in SNAPSHOT where a map address (index into the array IMAP) was being substituted for a map id (unique map index). Two approaches were devised for retaining the functionality of the linked history file. The first approach is currently installed in the TRC. The second approach has been presented as an alternative. Using memory to replace the linked history file increased the total memory requirements very slightly. For example, with 8 PBVs - 9 RV/Ds, the extra memory was 3600 CRAY words (SUN equivalent of 14,400 bytes). In light of the great amount of run time dedicated to the linked history file, this was viewed as an excellent trade-off.

There were already arrays indexed by the variable RPTAD (report address) that contained report number and scan number for all reports since the last snapshot. The subroutine CTCOUT already made limited use of the information in these arrays. The new approach creates an array, for each map, that contains the map id at the last snapshot (actually the address of the map) and the count or reports and the list of report addresses that fully describe the report history since the last snapshot.

An alternative approach was to create a common block that contained the information formerly written to the linked history file for a period of one snapshot. This information was stored in memory and written over for each snapshot. In a sense, this became a circular buffer. The information was accessed by record number in the same manner that the linked history file was read. A separate array was used to store the map id from the previous snapshot as this was the only information needed from a previous snapshot. Functionality of the subroutines FILLSCN, CTCOUT, and SNAPSHOT was not changed by these modifications.

4.3 Sensor Resolution and Sensor Noise Considerations

Clearly sensor resolution is not infinite. The sensor simulation package (SPACEGEN) employed in the current work on the TRC takes this into account by adding random noise to the sensor readings. Each sensor report is adjusted by adding a random number multiplied by a resolution amplitude.

For the data sets that were shared for common use by this group, the noise amplitude was set to 0.0057 which has been interpreted by Sandia to be 10^{-4} radians. This noise level (resolution level) was such that definite problems with tracking and correlating were observed. In a meeting of the group at Los Alamos in June, it was noted that the TRC code had about 3.3 times as many resolved tracks as there were objects providing the simulated sensor data (actual count known from the scenario generator). The sensor report scatter was great enough that each object was being tracked as if it were 3 or 4 objects by grouping together only those sensor reports that had similar noise components. This significantly affects the memory requirements and other execution characteristics for any given test case.

In several examples where attempts were made to follow the tracking of a single object step by step, the apparent situation was that the new location of the object was fixed by the current sensor without significant dependence on any of the previous reports. This created a curious situation. The Kalman filter process was to incorporate each additional sensor report into a cumulative least square fitting of the data. It was also found that generating lower noise sensor data significantly improved the result.

The problem was found to arise from the following situation. In the Kalman filter process, the weighting for the current report was determined to be inversely proportional to the variance assigned to it. That is, the smaller the variance, the greater the assigned weight. From a measurement standpoint, the variance cannot simply be determined from an isolated observation.

Estimating the variance can be accomplished in at least three ways: (1) simply keep track of the differences between where the object was expected to be located and where the sensor reports indicate, (2) consider an input parameter in the initialization file so that just as a sensor noise level is postulated in generating the sensor data, so also is the variance correspondingly postulated for processing the sensor data, and (3) hard-code an "appropriate" number into the code to represent the variance.

The TRC program used option (3) describe above. But the value for the variance hard-coded into subroutine PPRSCAN corresponded more closely to an infrared sensor error of 10^{-5} radians than 10^{-4} radians. Hence, compared to the data being processed, the current sensor report was over-weighted by an order of magnitude. Significantly better tracking was observed when the hard-coded variance number was increased to more closely match the simulated sensor data. The possible alternative options listed previously suggest an additional area of analysis to assess which option is most reasonable.

Finally, the supplied data sets for the TRC code had the cluster splitting threshold set to 1000 meters. With this level of sensor report uncertainty, splitting occurred very frequently. But typically the map split off was traveling on a path that would not encounter another sensor report.

4.4 Error Corrections to the TRC Code

Although Sandia was not specifically looking for errors in the TRC implementation, during the course of moving the code among the SUN, NCUBE, and CRAY computers, several errors have become known and have been corrected. All of these errors are listed in Appendix A. One error in particular will be elaborated at this point since it was, in fact, a dominating factor in driving memory requirements to be extremely large in the original version of the TRC.

The subroutine GENNEAR seemed to require large amounts of memory for the table of distances between maps with the same report history. The space required was $N(N+1)/2$ where N was the number of maps in the history bin. Examination of the contents of the large bins turned up an error. The bins with a large number of entries were those for which no report had been received. The error was that this list was being accumulated over clusters. In the subroutine HISTASGN the array PNUMMAPS was used to store the number of maps associated with each history id where the history id was the index to the PNUMMAPS array. Each time HISTASGN was called, the PNUMMAPS array was zeroed. However, it was found that PNUMMAPS(0) also needed to be set to zero <u>for each cluster</u> to prevent maps with a previous history id of zero from being accumulated over all clusters.

This same type of problem can occur for non-zero history ids when cluster splitting takes place. When a cluster is split, the maps associated with the original cluster are split between the two new clusters. However, the history id associated with these maps is left unchanged. This can result in clusters that have duplicate history ids and the wrong number of maps stored in the PNUMMAPS array for a history id. As described above, to ensure that this cannot happen, the PNUMMAPS array must be zeroed for each cluster _before_ the map data are assembled in HISTASGN.

From this discovery arose a secondary question. For map merging purposes, should those maps for which no report has been received be treated as a bin even within a cluster, or, in fact, should the maps simply be ignored by GENNEAR? In Sandia's version, GENNEAR does ignore those maps.

4.5 Modifications to the SPACEGEN Codes

Sandia also envisioned moving the programs that generated the object tracks and sensor reports for the TRC to the NCUBE. Two effects would result from this plan. In the short term, data could simply be generated on the same machine as runs the TRC and files would not have to be transmitted over communication lines between different machines. In the long term, parallelization of the sensor report inputs to the TRC would move the execution of TRC closer to a real-time environment.

Since large data files were to be generated for larger scenarios it was prudent to also transport the generator code to the CRAY. Execution time and memory space again became significant factors in simply getting the data files created.

The generic reference to SPACEGEN actually involved several programs: SPACETRACKS, SPACEGEN, SPACEGENTOSDI, and a sort routine that might (or might not) be available via the hosting computer system. SPACEGEN generated sensor reports by taking the track truth files from SPACETRACKS and applying each sensor platform data to all tracks in sequence. The resultant file was then sorted with sensor platform report scan time steps as the primary sort key. For moderate and large data files, this sorting process can use large amounts of disk space and be time consuming. Plans have been made to modify SPACEGEN to place the time loop outside the sensor and target track loops so that final sorting is no longer required. This approach also more attunes the data generation to an actual threat scenario.

The programs SPACEGEN and SPACEGENTOSDI were used to generate sensor data reports in a suitable form for input to the TRC code. Both programs used the ground truth target track data generated by the SPACETRACKS program and they must store these data in array space.

4.5.1 Memory Utilization Changes

The target track data were stored in nine, 2-dimensional arrays that were dimensioned by the maximum number of targets and the maximum number of orbit legs for a target. (An orbit leg is defined to be a portion of a trajectory that conforms to a specific orbit equation. Slightly different orbits may be used by a boost vehicle at various times during flight.) In practice, the PBV type targets have 2N + 1 orbits legs, where N = number of Reentry Vehicles (RV) + number of Decoys (D). However, the RVs and Ds have only one orbit leg each. The 2-dimensional array storage scheme was based on the PBV number of legs and would therefore result in a substantial waste of memory. This problem only became apparent when generating TRC sensor input for the 20 PBV - 99 RV/D scenario. This problem required 3,700,000 words of CRAY memory (SUN approximate equivalent of between 14.8 and 29.6 Mbytes of memory, depending on the precision desired).

The Sandia modified versions of both SPACEGEN and SPACEGENTOSDI use the same nine arrays to store the groundtruth target track data, but the arrays are only one dimensional. Space is allocated to these arrays based upon the sum of the maximum number of PBVs multiplied by twice the number of RVs and Ds per PBV and the maximum number of targets (PBVs + RVs + Ds). The number of orbit legs for each target was already being stored. In the modified version, this value was used to increment the index for these nine arrays. These changes resulted in a reduction from 3,700,000 CRAY words (SUN approx. equivalent of 14.8 Mbytes) to 120,000 words (SUN approx. equivalent of 0.5 Mbytes) for the 20PBV - 99 RVs/Ds problem.

4.5.2 Error Corrections

A few minor errors were also found in the SPACEGEN suite of programs and those are described fully at the end of Appendix A. One error was noted in the SPACETRACKS subroutine QROOT which computes the zeroes of particular quartic equations. In this subroutine, complex variables are equivalenced to real variables. Within the computer memory this can cause difficulties since the imaginary part is equivalenced quite possibly with whatever resides in the second word of storage. Although all FORTRAN compilers permitted this to pass, execution of statements involving these complex variables could have generated spurious results. As it turned out, the variables were used in a comparison that apparently had zeroes already in memory and there were no side effects.

5.0 Continuing Work on Parallelization

Sandia continues to move toward a massively parallel tracker/ correlator. All of the work up to this point on the serial version was actually geared to the parallel version and, in a sense, can be considered a part of the parallelization effort. Certainly, from the study and analysis performed, the team has a much clearer view of the areas of concern associated with this problem.

5.1 TRC on the NCUBE

The SPACEGEN suite of codes has been moved to the NCUBE and changes to the code are underway to permit the code to execute on the node processors. Several possible node arrangements are being examined for the TRC to execute on the node processors. Under the current NCUBE configuration at Sandia, a heterogeneous super node may be necessary to handle the TRC requirements. On the other hand, sufficient progress has been made on the serial version to suggest that perhaps the most critical segments of code can execute on single nodes for problems near 100 objects. This has positive implications for very large problems, e.g., 100,000 objects over 1024 processors. To achieve parallel load balance there will be several control nodes in communication with worker nodes for preliminary processing and cluster assignments. This form of heterogeneous processing has been utilized in another Sandia project [Gustafson 89].

All of these considerations are affected somewhat by the fact that Sandia plans to install a new NCUBE2 Hypercube that will contain 1024 nodes with node processors executing at nearly 10 times the rate of the current processors. Each node processor will have an associated 4 Mbytes of memory rather than the current 0.5 Mbytes.

5.2 Progress on Other Algorithms and Codes

Sandia was also requested to examine two other algorithms. Copies of the relevant codes have been obtained and work has been started to analyze these algorithms. The first algorithm was from Alphatech and was written in FORTRAN to run on the shared memory Alliant parallel computer. For this midcourse algorithm implementation, Alphatech has been running small test problems using facilities at Argonne National Laboratory. The second algorithm was from MITRE in Bedford, Massachusetts. The algorithm was written in Pascal and runs on a DEC VAX computer. Primarily a boost-phase tracker, this algorithm has no spawning of new objects from the boosters. The test data sets for this algorithm are fairly small with time starting at launch and extending for approximately 50 seconds after the launch.

Both algorithms are being studied to learn more about tracking and correlation of many objects. Reports and papers from other groups (Oliver E. Drummond of Hughes Aircraft [Drummond 88b] , Keh-Ping Dunn of MIT Lincoln Laboratory [Dunn 89], etc.) and proceedings from the SDI Panels on Tracking [SDIO 89] are also under review.

References

[B&W 88a] Ball Systems Engineering Division and D.H. Wagner Associates Inc., SDI Tracker/Correlator Algorithm Design Document, 2nd Edition, R-061-88, Naval Research Laboratory, July, 1988.

[B&W 88b] Ball Systems Engineering Division and D.H. Wagner Associates Inc., SDI Tracker/Correlator Algorithm Implementation Document, 2nd Edition, R-063-88, Naval Research Laboratory, July, 1988.

[Bate 71] Bate, Roger R., Mueller, Donald D., and White, Jerry E., Fundamentals of Astrodynamics, Dover Publications, New York, 1971.

[Bertsekas 89] Bertsekas, D.P. and Tsitsiklis, J.N., Parallel and Distributed Computation, Prentice-Hall, 1989.

[Blackman 86] Blackman, Samuel S., Multiple-Target Tracking with Radar Applications, Artech House Publishers, Dedham, Mass., 1986.

[Drummond 88] Drummond, Oliver E., Multiple Target Tracking, Course Lecture Notes, Technology Training Corporation, 1988.

[Drummond 88] Drummond, Oliver E. and Blackman, Samuel S., " Multiple Sensor, Multiple Target Tracking Challenges of the Strategic Defense Initiative", Proceedings of the 1st National Symposium on Sensor Fusion, 1988.

[Dunn 89] Dunn, K.P., Mori, S., Chang, K.C., and Chong, C.Y., "Tracking Performance Evaluation: Prediction of Track Purity", Signal and Data Processing of Small Targets 1989, SPIE Proceedings, Vol. 1096, March 1989.

[Fox 88] Fox, Geoffrey C., et al.,Solving Problems on Concurrent Processors, Volume 1, Prentice-Hall, 1988.

[Gustafson 88] Gustafson, J.L., Montry, G.R., and Benner, R.E., "Development of Parallel Methods for a 1024-Processor Hypercube", SIAM Journal on Scientific and Statistical Computing, Vol. 9, No. 4, July 1988.

References (Cont.d)

[Gustafson 89] Gustafson, J.L., Benner, R.E., Sears, M.P., and Sullivan, T.D. "A Radar Simulation Program for a 1024-Processor Hypercube", Proceedings of Supercomputing '89, ACM Press, 1989, pp. 96-105.

[SDIO 89] Frenkel, Gabriel, (ed.), Proceedings of the SDI Panels on Tracking, Issue No. 3/1989, Institute for Defense Analyses, July, 1989.

APPENDIX A

Details of Error Corrections for the TRC

1. In the subroutine INITPBV, the file name LUTTYO was misspelled with a "zero" instead of the letter "oh". The SUN simply opens and writes to a disk file. Both NCUBE and CRAY FORTRAN abort the program when executing the statement.

2. In the subroutine CTCOUT, the index, RPTAD, used in a print loop was not defined within the loop, making the printout meaningless.
(Add " RPTAD = CTCRPT(ICTC,IRPT) " within the loop 560)

3. The subroutine TEMPTURE contained a zero divide when both radiation values were zero. When executing on the SUN the problem was ignored. On the NCUBE and the CRAY, the program aborted. Setting the temperature to zero was not an acceptable solution because it would be used as a divisor later. The problem was resolved by setting the temperature to 1.0.

4. The predicted impact latitiude was wrong. In the subroutine IMPACT, it appeared that the calculation was changed from using arctan(y,x) to using z/r without a corresponding change to use arccos.
(Use " POS(1) = 90.0 - RTOD * DACOS(P3(3)/DLEN) ")

5. The predicted impact longitude and time went bad late in tracking. In the subroutine IMPACT, the polar coordinate description of the current position of the object forced the sine of E0 to be positive, i.e., the value becomes wrong as the object moved from the 2nd to the 3rd quadrant.
```
(Change   " ESINEO = DSQRT( ESINEO ) "
     to   " ESINEO = DSIGN ( DSQRT( ESINEO ), RRDOT ) "
 and add to the definition of E0
          " IF ( E0 .LT. 0 ) E0 = E0 + TWOPI "  )
```

6. In the subroutine HANDOFF, the variable SCNTIM was not updated from the initial definition. If later Post Boost Vehicles (PBVs) were scheduled for more than the threshold value, HNDOFFDELTA, later, those PBVs were never processed.
(A possible fix to this problem is to add the <NISCAN_AUX> include file to the subroutine CTAMAIN and set SCNTIM to the value contained in the variable TSCAN containing the current scan time after the call to the subroutine NETINT.)

7. When the option was selected to list the sensor reports from the subroutine LOADSCAN, the azimuth and the elevation were reversed between heading labels and values displayed.

8. The display line sent to the terminal for each scan:
"SCAN xxx at Time xxx.xxx   x   IR Reports processed in
 xxx.xxx seconds"
is inconsistent and confusing. The scan number and the seconds used for this output are for the scan just completed, but the time and the number and type of reports are for the scan just beginning.

9. GENNEAR seemed to require a lot of space for the table of distances between maps with the same report history. The space required is n(n+1)/2 where n is the number of maps in the history bin. The bins with a large number of entries were those for which no report had been recieved. The error was that this list was being accumulated over clusters. In the subroutine, HISTASGN, the array element PNUMMAPS(0) must be zeroed at the beginning of processing for each cluster.

10. All PNUMMAPS entries must be cleared between clusters. The same type of problem described in Item 9 can occur for non-zero history-ids when cluster splitting occurs. When a cluster is split the maps associated with the original cluster are split between the two new clusters. However, the history id associated with these maps was left unchanged. This can result in clusters that have duplicate history ids and the wrong number of maps stored in the PNUMMAPS array for a history id. Clearing the PNUMMAPS array ensures this cannot happen.

11. Inconsistent (wrong) dimensions in the group of routines SCENEPRUNE, DUPSCENEOUT, COPYDATAOUT. The array IDXSAME (aka. DUPSCN) was dimensioned NMAXSCENE in one place and NMAXLIST the others. What the dimension should be was not clear. Consistency is desirable, however. SCENEPRUNE is simply the driver, where the storage is defined. DUPSCENEOUT looks at all the scenes and constructs a list of duplicates. COPYDATAOUT looks at the list of duplicates and copies all of the scenes that are <u>NOT</u> in the duplicate list. A duplicate is a scene that is the same as an earlier scene. The issue is confused by the fact that while all this work is required to mark a scene as a duplicate, the list being constructed in DUPSCENEOUT includes all duplicates when there are multiple duplications. That is, the duplicate list can be larger than the scene list. In the Sandia version, DUPSCENEOUT has been changed to eliminate the unnecessary accumulation of multiple duplicates. This allows one to know how to dimension the array IDXSAME. The change to the code in DUPSCENEOUT is to simply jump out to label 10, after an entry is added to the "SAME" list. Hence an adequate dimension is NMAXSCENE.

12. Additional changes were made to DUPSCENEOUT which enhance clarity, but do not affect the output from the subroutine. Change the DO statements for loops 10 and 30 so that the loop range covers only "I" greater than "J", removing the if test for that condition. Also move the loop 40 inside the test on number of maps, so it is not executed when the result is not needed.

13. The subroutine SNAPSHOT uses the variable MAPID when it wants the unique map identifier for a map that is time-continued. However, in SNAPSHOT and <u>some</u> other places, MAPID is the map address (map index). What should be used is the value from IMAP(LMAPIDENT,MAPID). This occurs in SNAPSHOT (depending upon version used) 32 code lines after the linked history file is read.

14. In HISTASGN, format 6060 defines a single output line of 630 characters. This is unacceptable in CRAY and NCUBE FORTRAN. (Change "FORMAT (10(13X, 10I5))" to FORMAT (13X,10I5)").

15. In CHLINV, an external reference to a non-existant subroutine, PRTERR, was deleted. Also in GENNEAR, an external reference to a non-existant file, GETEXT, was deleted.

16. In QROOT of SPACETRACKS, the equivalence of complex variables to real variables was eliminated. Comparisons are made using the actual real variables.

17. Subroutine PROCPBV read in the RV/D sequence data improperly in SPACETRACKS. This data was part of the session data and was read in separately for each session within the data for a PBV. The error occurred because the character string positions that the characters (R or D) were read into were not incremented for multiple sessions. This resulted in tracks that were output with incorrect identifiers. The problem was corrected by using a new character variable to read the data and then moving the data into the appropriate string position in the original character string variable.

DISTRIBUTION:

David Israel
SDIO/POET
Suite 300
1225 Jefferson Davis Highway
Arlington, VA 22202

LTC Jon Rindt
SDIO/ENA
The Pentagon
Washington, DC 20301-7100

Maj Stephanie Loverro
ESD/ATS
United States Air Force
Hanscomb, AFB 01731-5000

Steven McBurnett
U.S. Naval Research Laboratory
Code 5570
4555 Overlook Ave. SW
Washington, DC 20375

Albert Perrella
SDIO/POET
Suite 300
1225 Jefferson Davis Highway
Arlington, VA 22202

LtCol Charles Lillie
SDIO/ENA
The Pentagon
Washington, DC 20301-7100

MAJ Steve Johnson
SDIO/ENA
The Pentagon
Washington, DC 20301-7100

James Sanderson
MEE-10
Los Alamos National Laboratory
Mail Stop K488
P.O. Box 1663
Los Alamos, NM 87544

Kurt Askin
U.S. Naval Research Laboratory
Code 5570
4555 Overlook Ave. SW
Washington, DC 20375

Lawrence Filippelli
Ball Systems Engineering
Suite 1006
2200 Clarendon Blvd.
Arlington, VA 22201

Harold Camp
SDIO/POET
Suite 300
1225 Jefferson Davis Highway
Arlington, VA 22202

Barry Belkin
Daniel H. Wagner Associates
Station Square 1
Paoli, PA 19301

Kathy Sommar
Daniel H. Wagner Associates
Station Square 1
Paoli, PA 19301

William Binzer
Ball Systems Engineering
Suite 1006
2200 Clarendon Blvd.
Arlington, VA 22201

Leslie Pierre
SDIO/ENA
Room 1E149
The Pentagon
Washington, DC 20301-7100

Virginia Kobler
USASDC
ATTN: CSSD-H-SBY
P. O. Box 1500
Huntsville, AL 35807-3801

Steve Risner
USASDC
ATTN: CSSD-H-SBY
P. O. Box 1500
Huntsville, AL 35807-3801

David Castanon
Alphatech
111 Middlesex Turnpike
Burlington, MA 01803

Daniel Holtzman
Vanguard Research, Inc.
10306 Eaton Place
Suite 450
Fairfax, VA 22030-2213

Gabriel Frenkel
Institute for Defense Analysis
1801 North Beauregard Street
Alexandria, VA 22311

James Boyle
Argonne National Laboratory
Mathematics & Computer Science Div.
9700 South Cass Ave.
Argonne, IL 60439-4844

Barry Fridling
Institute for Defense Analysis
1801 North Beauregard Street
Alexandria, VA 22311

Keh-Ping Dunn
Massachusetts Institute of Technology
Lincoln Laboratory
P. O. Box 76
Room KB252
Lexington, MA 02173

400 - G. G. Weigand
1000 - V. Narayanamurti
1400 - E. H. Barsis
1410 - P. J. Eicker
1420 - W. J. Camp
1424 - R. E. Benner
1424 - J. L. Tomkins
1424 - J. P. VanDyke
3141 - S. A. Landenberger (5)
3141-1 - C. L. Ward for DOE/OSTI (8)
3151 - W. I. Klein (3)
5000 - E. H. Beckner
5100 - H. W. Schmitt
5170 - T. A. Sellers
5173 - P. J. Merillat
5173 - R. D. Halbgewachs (10)
5174/POET - R. F. Davis
5200 - J. Jacobs
8524 - J. A. Wackerly
9000 - R. L. Hagengruber