CONF-8911125--2

# DESIGN AND IMPLEMENTATION OF A MULTI-SENSOR FUSION ALGORITHM ON A HYPERCUBE COMPUTER ARCHITECTURE*

CONF-8911125--2

DE90 003682

CHARLES W. GLOVER

ENGINEERING PHYSICS & MATHEMATICS DIVISION
OAK RIDGE NATIONAL LABORATORY
OAK RIDGE, TENNESSEE

PAPER TO BE PRESENTED AT:

SPIE'S "ADVANCES IN INTELLIGENT ROBOTICS SYSTEMS & VISUAL COMMUNICATIONS & IMAGE PROCESSING '89"

Adams Mark Hotel
Philadelphia, Pennsylvania

November 5-11, 1989

---

*Research supported in part by the U.S. Air Force/Wright Aeronautical Laboratory under DOE Interagency Agreement, DOE-40-1579-85, under Contract No. DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc. with the U.S. Department of Energy.

---

# Design and implementation of a multi-sensor fusion algorithm on a hypercube computer architecture

Charles W. Glover

Oak Ridge National Laboratory
P. O. Box 2008, Oak Ridge, Tennessee 37831-6364

## ABSTRACT

A multi-sensor integration (MSI) algorithm written for sequential single processor computer architecture has been transformed into a concurrent algorithm and implemented in parallel on a multi-processor hypercube computer architecture. This paper will present the philosophy and methodologies used in the decomposition of the sequential MSI algorithm, and its transformation into a parallel MSI algorithm. The parallel MSI algorithm was implemented on a NCUBE™ hypercube computer. The performance of the parallel MSI algorithm has been measured and compared against its sequential counterpart by running test case scenarios through a simulation program. The simulation program allows the user to define the trajectories of all players in the scenario, and to pick the sensor suites of the players and their operating characteristics. For example, an air-to-air engagement scenario was used as one of the test cases. In this scenario, two friend aircrafts were being attacked by six foe aircraft in a pincer maneuver. Both the friend and foe aircrafts launch missiles at several different time points in the engagement. The sensor suites on each aircraft are dual mode RADAR, dual mode IRST, and ESM sensors. The modes of the sensors are switched as needed throughout the scenario. The RADAR sensor is used only intermittently, thus most of the MSI information is obtained from passive sensing. The maneuvers in this scenario caused aircraft and missile to constantly fly in and out of sensors field-of-view (FOV). This resulted in the MSI algorithm to constantly reacquire, initiate, and delete new tracks as it tracked all objects in the scenario. The objective was to determine performance of the parallel MSI algorithm in such a complex environment, and to determine how many multi-processors (nodes) of the hypercube could be effectively used by an aircraft in such an environment. For the scenario just discussed, a 4-node hypercube was found to be the optimal size and a factor two in speedup was obtained. This paper will also discuss the design of a completely parallel MSI algorithm.

## 2. INTRODUCTION

Sensor information is required by any autonomous or semi-autonomous platform in order to estimate the state of the environment in which the platform is operating. Information about the estimated state of the operating environment is then passed on to the platform control and command center where plans and decisions are made concerning the platforms actions in the environment.

Whether the platform is an aircraft, satellite, or a robot; sensor data is obtained from complementary multispectral sources. Each sensor presents a different picture of the environment which must be integrated to one coherent view, this is the role of Multi-Sensor Integration (MSI) software. In the case of a robot, the MSI estimate of the environment is passed on to an expert system

™Trademark of the NCUBE Corporation, Beaverton, Oregon.

where the information can be used for navigational planning, sensor redirection, and/or any form of resource allocation. For the combat aircraft or satellite case, the MSI software is to provide information concerning the number of targets in the environment, an estimate of each targets kinematic observables, and an estimate of the target type. This information is used by the aircrafts fire control program to allocate resources in order to maximize its probability of survival.

In combat situations it is important that MSI be performed as quickly as possible. For SDI or any other congested and complex scenarios, the MSI calculation would require a supercomputer. However, it is not practical to load a serial supercomputer on a combat platform. The question which arises is that can small transportable hypercube supercomputers be used to perform the MSI calculations in real-time? Before this question can be answered, several other questions must be addressed first, such as: Can serial MSI algorithms be adapted into a concurrent form without completely rewriting the entire program; and if so, what is the performance of such a transformed program; if not, can a concurrent MSI algorithm be devised? The goals of the research described in this report are to port very large sequential Multi-Sensor Integration (MSI) program to the NCUBE$^{TM}$ hypercube computer, identify and implement the existing concurrency in the program (Section 3), qualitatively document the performance of the resulting concurrent program (Section 4), and suggest alternative concurrent MSI algorithms (Section 5). Many of the issues addressed in this report for porting large sequential programs to hypercube computer architectures are generic and not limited to just this specific MSI program, the concurrent MSI algorithm described in Section 5 is completely general and can be applied to robotics as well as avionics.

## 3. MSI CODE CONVERSION FOR A HYPERCUBE

### 3.1. Architecture

Two sequential VAX FORTRAN programs, comprised of over 30,000 lines of code, were ported to the NCUBE host processor. One program is a general M-on-N engagement simulation driver. Simply-stated, this program models a user-specified engagement scenario and produces updates of the simulation states at specified time points. The output from this program serves to drive sensor models in the second program.

The second program simulates real world sensors, specifically RADAR, LADAR, ESM, and IRST, whose operating characteristics are defined by the user. The software provides realistic data of sensor observables and their associated errors for use in the MSI program, it is the MSI program which was divided and implemented concurrently onto the nodes of the hypercube computer.

This section presents the philosophy, rationale, and the steps that were taken in order to transform the sequential MSI code into a concurrent form. To facilitate the discussion a brief review of the structure of the sequential MSI code will be presented first.

### 3.2. Review of the sequential MSI code

The goal of the MSI algorithm is to provide an accurate estimate of the environment based upon data returned by the sensors. Before the state of the

environment can be estimated the MSI algorithm must solve the data association problem. The objective of the data association problem is to classify all the sensor tracks into the following bins:

- Confirmed Correlation to a MSI Track
- Unconfirmed Correlation to a MSI Track
- New Detection
- False Alarm

Figure 1 provides a schematic illustration of how the sequential MSI code accomplishes its goal. The first step in the procedure is to propagate all the sensor-1 tracks to a common time point by using a Kalman filter with a standard target acceleration model. The MSI tracks are initialized from the propagated tracks provided by sensor-1. Next, all sensor-2 tracks are propagated to the current MSI track time point. At this point the data association problem must be solved, the tracks from sensor-2 must be correlated with the MSI tracks from sensor-1; this is accomplished by binning the sensor-2 tracks into one of the four categories discussed above. The first step in this procedure is to perform gate tests on the attribute and kinematic variables between the MSI and sensor-2 tracks. If a sensor-2 track has been previously correlated with a MSI track and passes the 3 gate test it remains in the confirmly correlated bin. The next step is to enlarge the gate and repeat the test for the remaining unconfirmed uncorrelated sensor-2 and MSI tracks. A likelihood of correlation estimate is then performed for all sensor-2 and MSI tracks which pass the second gate test. The sensor-2 and MSI tracks that do not pass this gate test are declared either a new detection or a false alarm, based upon the probability of detection for sensor-1 (assumed to have the highest probability of detection for all sensors) and the probability of a false alarm for sensor-2. Before declaring that a sensor-2 track is correlated with a MSI track, the confidence in this hypothesis is computed by comparing the likelihood of correlation estimate to the likelihood estimate that the correlation is a false alarm or a new detection. A sort is performed to obtain the highest scoring set of correlations. Now, all the sensor-2 and MSI tracks have been grouped into the four bins mentioned above. The state vectors of the newly correlated MSI tracks are updated using again a Kalman Filter, thus providing an estimate of the state of the environment based upon the data from sensor-1 and sensor-2. This procedure is repeated sequentially until all sensor information has been processed and we are left with an updated estimate of the state of the environment based upon all sensor information.

### 3.3. Stages to concurrency

The transformation of the sequential MSI code to a concurrent form was based on the following philosophy:

- Do not alter the numerical results of the test cases;
- Leave the MSI algorithm intact;
- Convert the serial MSI code to a concurrent form in incremental stages.

It was decided not to implement changes in the MSI code that would cause a change in the numerical accuracy because of the resulting difficulties in determining the validity of the alterations against the test cases. There are fundamental changes in the MSI algorithm that could be made which would lead to a completely parallel algorithm. These changes are discussed in Section 5, but are not made

here because they would affect the accuracy of the calculations. The conversion of the serial MSI code into a parallel code was done in incremental stages. This reduced the time spent debugging the changes and allowed us to document how each change effected the execution time. The actual changes in the conversion of the serial MSI code to concurrent form are illustrated in Figure 2 and are discussed below.

Stage - I

As shown in Figure 2, the likelihood of correlation estimate was the first computational task to be assigned to the node processors. The likelihood computations involve a number of time consuming matrix operations for calculating the residual nearest-neighbor distance (score) for each feasible combination of sensor and MSI tracks. It was decided to assign one feasible sensor-MSI track combination per node until all combinations were distributed over the hypercube. Each node would calculate the likelihood of correlation score for a particular sensor-MSI track pair.

Stage - II

As shown in Figure 2, the MSI state vector update estimate was the second computational task to be assigned to the node processors. This routine uses a Kalman filter to calculate an updated estimate of the state vector and the associated covariance for all MSI tracks.

The state vector of a MSI track is composed of kinematic (position, velocity, and acceleration) and attribute (Missile, RADAR or IR; fighter, friend or foe; etc.) variables. Kinematic variables were updated in parallel by assigning different kinematic variables to each of the nodes in the hypercube; each node then performed a Kalman Filter calculation on the kinematic variables assigned to it. The results of these calculations were sent back to the host processor for attribute variable updating and subsequentially for processing of the next sensors information.

Stage - III

As shown in Figure 2, the common referencing of sensor tracks was the third computational task to be assigned to the node processors. This routine propagates all sensor tracks to the same space-time point in a common reference frame. All sensor tracks, whether from the same sensor or different sensors, have different time stamps associated with them; since the new measurements occur asynchronously in time then sensor tracks are formed asynchronously at different times. In order to fuse sensor tracks, all sensor tracks must be projected to the same time point by modeling the state vector dynamics for each sensor track. The model used by this program assumes the target can maneuver with a constant acceleration, and a Kalman Filter is used in conjunction with this model to propagate all sensor tracks to a common time point. In addition, corrections for misalignment between sensors are applied to the sensor tracks and the tracks are transformed into a common space reference frame.

It was decided to assign one sensor track per node until all tracks were distributed over the hypercube. Each node would then perform the common referencing calculation for a given sensor track. Once Stage III is implemented approximately half of the MSI code is programmed in concurrent form on the nodes.

Generally speaking, each node in the concurrent version of the MSI code performs a computational task on a track. As the number of tracks increases so will the number of nodes needed in the calculations. If the number of tracks is greater than the number of nodes then some nodes will perform calculations on more than one track. Quantitative measurements of the execution times are presented in Section 4.

### 3.4. Rationale for the coarse grain

The coarse grain approach adopted towards the conversion of the serial MSI code in Stages I-III was chosen in order to accommodate the possibility of re-dimensioning the code to handle a larger number of targets. The current MSI code is dimensioned to handle up to 20 targets and 4 sensors. An additional consideration in determining grain size is the NCUBE's message passing overhead. To pass a message between two processors requires about the same amount of time as a single processor to perform 120 floating point operations (flops). Thus, a grain size of a 120 flops was the minimum to be considered for parallel execution.

## 4. PERFORMANCE RESULTS

In this section we present a description of the test case and a quantitative discussion of the hypercube performance results.

### 4.1. Description of test case scenario

In this report, an Air-to-Air Offensive Sweep test case will be discussed. In the Offensive Sweep scenario two friend players engage six foe players in a pincer attack. At the start of the scenario the two friend players and six foe players are flying toward one another separated by a distance of approximately 300 km. The total playing time (MSI time) is 300 s. At the 160 s mark, the six foe players break to engage the friend players: two foe players roll upwards and to the right while two other foe players roll downwards and to the left; these four foe players perform sweeping banks designed to intercept the friend players trajectories from above and below; the remaining two foe players close on straight line trajectories towards the friend players. The four banking foe players fly out of the friend players' sensors field-of-view and are reacquired when the two friend players launch missiles on center foes and break left and right to engage the banking foes. Each friend player has RADAR (two modes), IRST (two modes), and ESM sensors onboard. The information from these sensors must be fused by the MSI program in order to present the pilots a coherent view of the current state of the environment.

### 4.2. Quantitative timing and efficiency results

Shown in Figure 3 is the total cumulative execution time (denoted CPU time), for all three stages of concurrency implemented together on the nodes of the hypercube and is plotted as a function of the scenario time (MSI time). The curves in the figure represent the execution time for different sized hypercube configurations. The top most curve of the figure represents the sequential execution time of the MSI algorithm by the host processor. The VAX 11/785 time is provided as a point of reference and is not particularly relevant to the discussion that follows.

The 1-node calculation is the one by which to judge the effects of concurrency. This calculation represents a sequential MSI program in which all three stages were executed in serial on one node processor, while the remainder of the program was executed on the slower host processor. The 1-node calculation and the other concurrent calculations using more nodes have the host processor performing exactly the same number of operations. Hence, any difference in execution time between the calculations for 1-node and the other hypercube configurations is due to the concurrency in the program.

Table I summarizes the results displayed in Figure 3. In Table I, the execution time for 1-node calculation was normalized to 1.00, and the remaining numbers are relative to it. The most striking feature from Table I is that the 4-node hypercube configuration provides approximately a 1/3 reduction in execution time, while the reduction for the 8-node configuration is not much better. This indicates that the Offensive Sweep scenario does not sufficiently exercise a hypercube with 8 or more nodes.

Table I

| Processor | Speedup Factors |
|---|---|
| Host | 3.07 |
| VAX-11/785 | 1.76 |
| 1 - Node | 1.00 |
| 2 - Nodes | 0.77 |
| 4 - Nodes | 0.66 |
| 8 - Nodes | 0.60 |

These results can be understood by recalling the discussions of Stages I-III. In this concurrent implementation of the MSI program the number of nodes in use is proportional to the number of sensor and MSI tracks, and consequentially proportional to the number of targets in the scenario.

Figures 4-6 display the number of tracks per MSI time step for Stages I-III. Since the calculations for one track are handled by a single node, these figures indicate the maximum number of nodes that could be active per MSI time step in Stages I-III.

Figure 4 illustrates for Stage-I that at early times in the scenario many likelihood of correlation estimates are being performed. As track fusion takes place this routine is not called very often unless maneuvers occur or a new target is acquired.

Figure 5 shows for Stage-II the number of times a kinematic variable (position, velocity, and acceleration) for various number of tracks must be updated. For this scenario a maximum of 12 kinematic variable tracks could be updated per pass through Stage II. This corresponds to 2 kinematic variables per target for 6 targets.

Figure 6 displays for Stage-III the number of tracks per sensor requesting propagation to a time point in the common referencing calculation. At early MSI times each sensor has six tracks to be propagated. This can be done in two passes using 2 or 4 nodes of the hypercube or in one pass using 8 nodes with two nodes sitting idle.

From Figures 4-6, the fraction of the total MSI time for which calculations in each stage require 1, 2, 4, 8, or more nodes can be determined by placing thresholds in Figures 4-6 at the 1, 2, 4, and 8 track levels, respectively. This fraction (percentage) is displayed in Figure 7 for each of the three stages. The 1-node percentage represents the fraction of MSI time that a given stage is called. The 2-node percentage represents the fraction of MSI time 2 or more nodes could be used in the concurrent calculations, and likewise for the 4 and 8 node percentages.

The fraction of the total MSI program that requires 1, 2, 4, or 8 nodes in the calculations is obtained by averaging the percentages in Figure 7 over Stages I-III. This percentage, shown in Figure 8, shows that approximately 70%, 65% and 57% of the offensive sweep scenario can, respectively, utilize 2, 4, 8 or more nodes in the calculations. By considering these the percentages for the maximum node usage possible, it is easy to see that the speedup factors in Table I represents a very efficient use for a 2- and 4-node hypercube, and the efficiency declines for the 8-node hypercube.

The preceding analysis shows that a reduction of 1/3 could be obtained for this test case by using a 4-node hypercube and that larger sized hypercubes do not add significantly in the reduction of execution time. A more complex scenario with more players, more sensors per player, and more complex maneuvers will better demonstrate the utility and performance advantages of a larger dimensioned hypercube computer.

## 5. RECOMMENDATIONS FOR MSI ALGORITHMS AND PROGRAMS ON HYPERCUBE COMPUTERS

The MSI problem has a great deal of concurrency which can be exploited in a very efficient manner on a hypercube computer. Computer architecture must be taken into account from the beginning at the algorithm design stage. Algorithm design and its implementation in a program are not separate issues, they are intimately related through the architecture of the computer. Almost all algorithms designs are dictated by the architecture in which they will be executed.

This section presents some basic principles about hypercube computers which must be considered during the design of any concurrent algorithm and program. The section then concludes with recommendations for concurrent MSI algorithms.

### 5.1. Sequential MSI algorithm recast in concurrent form

This MSI algorithm has a great deal of concurrency which can be exploited, but only at the expense of completely rewriting the programs. It is shown in this section how the MSI algorithms can be arranged in a synchronized, completely concurrent form for a hypercube computer.

The MSI algorithm consists of four main computational procedures (see Figure 9):

1. Propagation of all MSI and sensor tracks according to the current state estimates (common referencing).
2. Association of all sensor tracks with MSI tracks (estimate likelihood of correlation).
3. Selection of the highest scoring associations (sorting and hypothesis selection).
4. Provide an updated estimate of the current state based upon the new

associations (updating MSI tracks).

Steps 1, 2, and 4 are ideally suited for decomposition into a concurrent form; these tasks for any one track are completely independent from any other tracks. The selection of the highest scoring associations, step 3, implies some global communications.

This MSI algorithm suggests an obvious decomposition for the hypercube architecture:

A. Distribute approximately equal number of MSI (or sensor) tracks to each node processor.
B. Distribute all sensor (or MSI) tracks across the nodes of the hypercube.
C. Each node performs steps 1 and 2, discussed above.
D. Selection of the highest scoring correlations, discussed in step 3, can be performed by a parallel optimization algorithms.[2]
E. Distribute approximately equal number of MSI tracks to each node of the hypercube and perform the updated state estimation, step 4.

In this version of the MSI algorithm, specifically step A, if more MSI tracks than sensor tracks are expected on average for a given scenario, the MSI tracks should be distributed across the nodes of the hypercube. In step B each node should receive all sensor track data. The reverse situation will apply if more sensor tracks are expected on an average in the scenario. Assigning the largest number of tracks to the nodes will lead to greater concurrency and throughput. Since all nodes perform the same calculations, the computational load on the hypercube will be balanced if all nodes are assigned the same number of tracks to process. For the case where the number of tracks is not a multiple of the number of nodes, a load imbalance will occur only for the amount of time it takes the nodes to process one extra track. This version of a concurrent MSI algorithm is well-suited to a dynamic environment where tracks are continuously being created and annihilated, because track information can be added and deleted easily from the nodes.

## 5.2. Consideration of the hypercube computer architecture

The basic concept in the design of concurrent algorithms and programs for a hypercube computer is that any time a processor (be it node or host) must wait for the results from another processor, a reduction in throughput will occur. When an algorithm is designed such that the calculation of step k cannot start until it has the results from step k-1 the algorithm is called a synchronous algorithm. An asynchronous algorithm is one in which the calculation for step k will be computed based on the results from step j, where $j < k$.

To further illustrate the concepts of synchronous and asynchronous algorithms, consider the following general iterative function

$$x_{k+1} = Q(x_k, x_{k-1}, \ldots, x_{k-e+1}),$$

where e is the memory of the iterative method and the vector $x_{k+1}$ has n dimensions. Now suppose we perform a series expansion of Q, then the iterative equation would have form

$$x_{k+1} = F(x_k, x_{k-1}, \ldots, x_{k-e+1}) + G(x_k, x_{k-1}, \ldots, x_{k-e+1}), \quad (1)$$

where F could be a linear function and G could be a function of the derivatives of Q. Further, assume that the execution time to calculate G is greater than that for F.

In order to facilitate a parallel computation, one strategy is to exploit the parallelism of the individual components $x_i$ (i=1,n), by assigning the computation of each component $x_{i,k+1}$ to a different node processor. For each iteration step each node processor updates its components according to Eq. (1). The processors are synchronized at the end of each iteration and the next iteration begins after all processors are finished updating. This is the method used in the current parallel version of the MSI code.

An alternative method for a parallel computation would be to recognize that G is more costly to calculate than F. Rather than waiting for the computation of $G(x_k, \ldots, x_{k-e+1})$, one might choose to estimate $x_{k+1}$ based on the last estimate of G (i.e., $G(x_j, \ldots, x_{j-e+1})$, where $j < k$). This represents an asynchronous algorithm.

In the two concurrent algorithms just discussed, Levin[1] as shown that for certain classes of nonlinear equations the asynchronous algorithm's order of convergence to a satisfactory approximation of the correct solution is the same as that for the synchronous algorithm. However, the execution time required for each asynchronous iteration can be drastically reduced. Both types of concurrent iterative algorithms have serious drawbacks; the synchronized algorithms may have processes blocked for a long period of time due to synchronization restrictions while the details of behavior of asynchronous algorithms are difficult to analyze. This leads to a hybrid approach called semi- synchronized algorithms which accentuates the positive features of both algorithms without being limited by the drawbacks. It appears that some iterative subtasks in the MSI problem could be accomplished in such a form.

In addition, it should be noted that neural network algorithms can be formulated as asynchronous algorithms. This feature naturally makes neural network algorithms prime candidates for concurrent computations.

## 6. SUMMARY

This report showed that a large sequential MSI program could be successfully transformed into a concurrent form and implemented on a NCUBE hypercube computer through a coarse grain decomposition of the sequential program. The essential idea behind the coarse grain approach is to assign each sensor track to a hypercube node such that the nodes perform the same computational tasks simultaneously on different sensor tracks. For the Offensive Sweep test scenario examined here, this method yielded a 1/3 reduction in the total execution time for a 4-node hypercube when compared with the total sequential execution time, and it demonstrated that an 8-node hypercube did not significantly reduce the total execution time further. However, more complicated test cases would exercise hypercube cubes of larger dimension and the total parallel execution time should be proportional to the sequential execution scaled by the ratio of the number of sensor tracks to the number of nodes.

This study also demonstrated that the current MSI algorithm could be recast in a completely concurrent form. The completely concurrent form exploits the fact that track propagation (common referencing), track association, scoring, selection, and updating calculations can be performed entirely on the nodes, but only at the expense of completely rewriting the MSI program.

Finally, this study addressed our current area of research in concurrent MSI algorithms, whether the track propagation (common referencing) and track update calculations could be performed asynchronously on the nodes of the hypercube. The advantage of this method is that the convergence of the estimation to the true value requires approximately the same number of iterations as the current synchronous Kalman Filter method but the execution time of each iteration is drastically reduced.

## 7. DISCLAIMER

Portions of this paper have appeared in the Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, Pasadena, California [G. Fox, editor] (January 1988).

## 8. REFERENCES


1. M. Levin, 1984, M. Sc. Dissertation Project, Center for Computing and Computer Science, Univ. of Birmingham, Birmingham, England.

2. Culioli, J. C., Glover, C. W., Jones, J. P., and Roe, C., "NCUBE Implementation of Some Heuristics and an Optimal Algorithm for Large-Scale Assignment Problem," Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications, Monterey, California (1989) [in press].

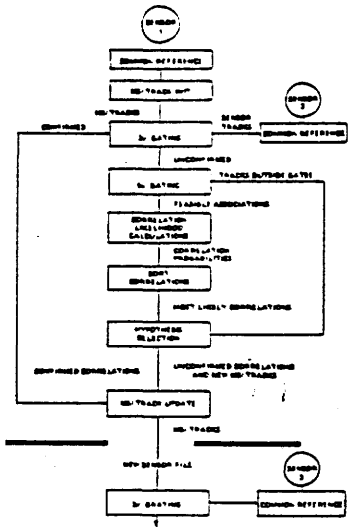


Fig. 1. Schematic illustration of the sequential MSI program.

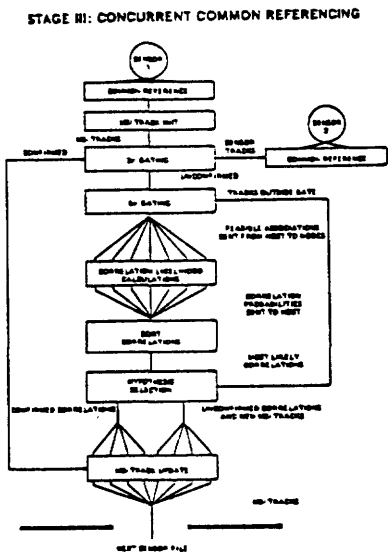


Fig. 2. Schematic illustration of the concurrent MSI program. The fan-out lines represent those tasks assigned to the hypercube nodes; fan-in lines represent information passed back to the host processor from the nodes.

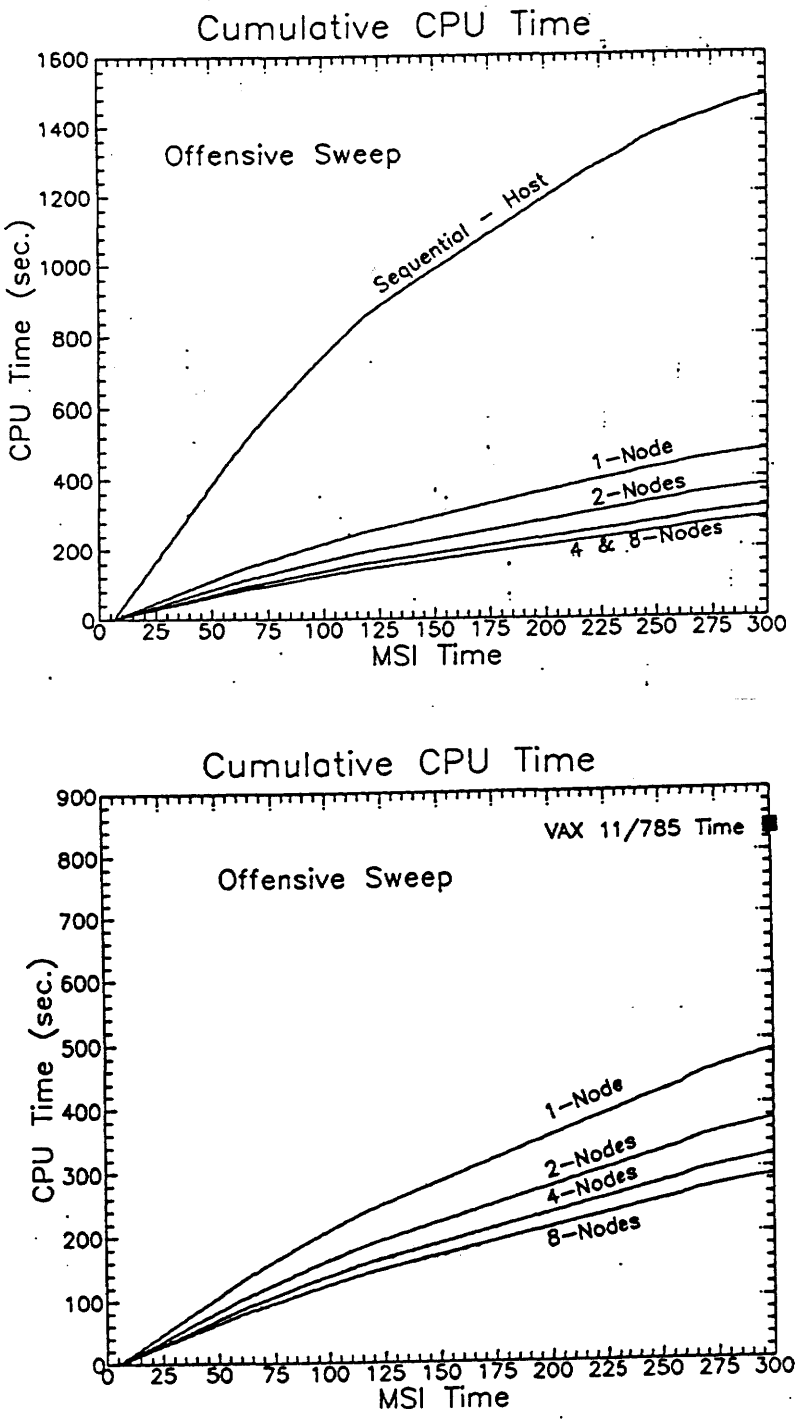


Fig. 3. The total cumulative execution time (denoted CPU time) plotted against the scenario time (denoted MSI time) for various hypercube configurations.

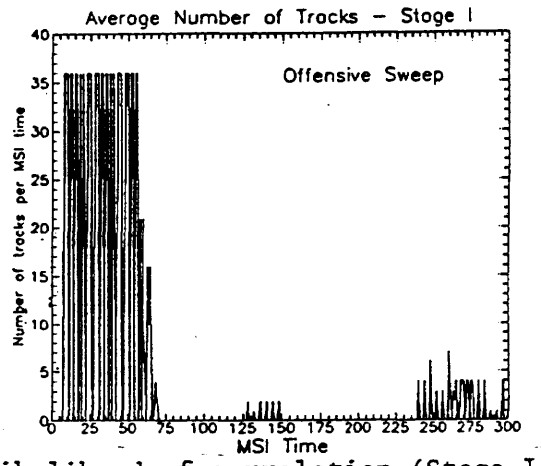


Fig. 4. The number of likelihood of correlation (Stage I) calculations to be performed at each MSI time. This represents the maximum number of tasks which could be assigned to the nodes, or the maximum number of nodes which could be active at a given MSI time.

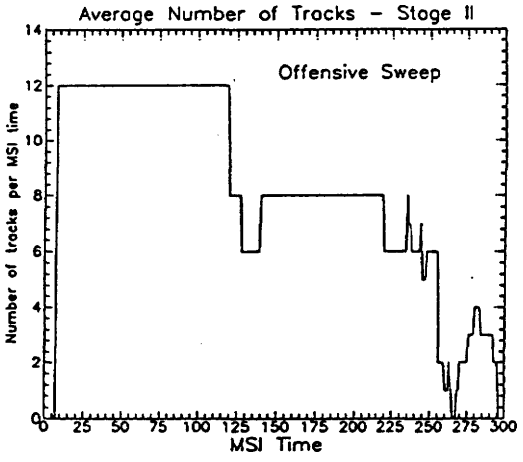


Fig. 5. Same as Fig. 4, except the number of kinematic variables to be updated per MSI time (Stage II) is displayed.

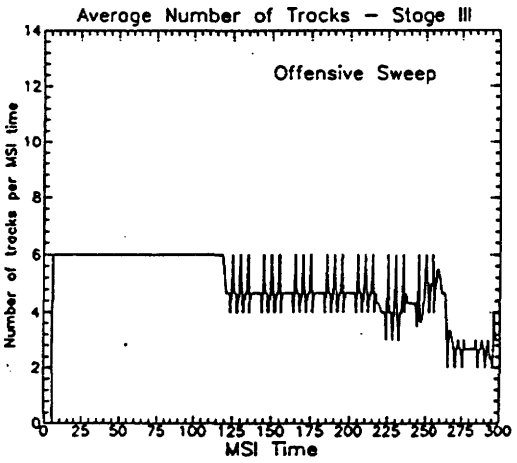


Fig. 6. Same as Fig. 4, except the number of sensor tracks requiring common referencing calculations (Stage III) per MSI time is displayed.

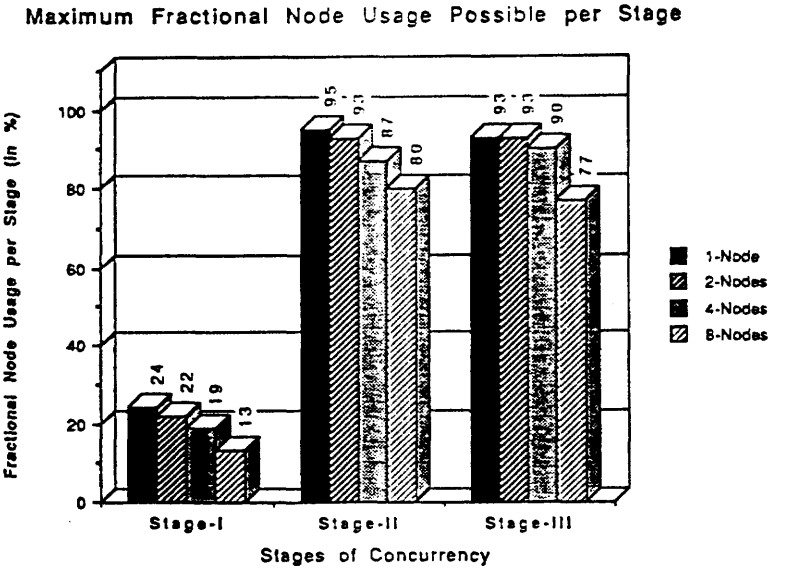


Fig. 7. The fraction of MSI time in which each stage could use 1, 2, 4, or 8 more nodes.

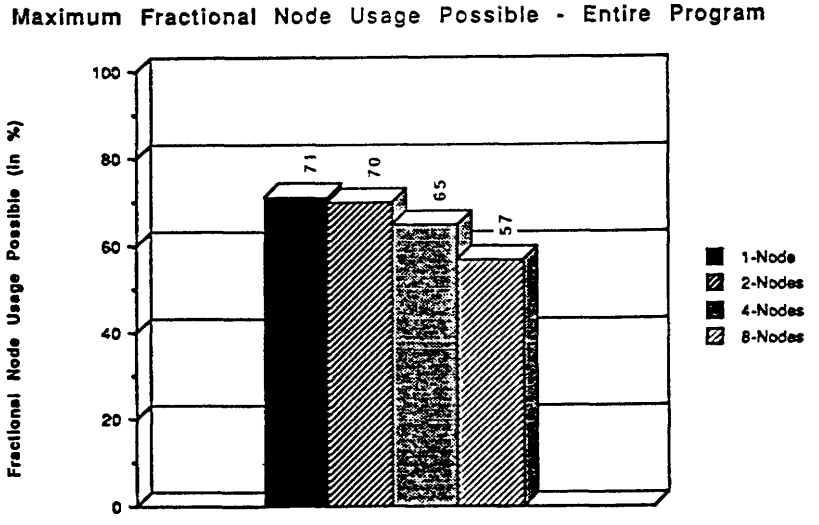


Fig. 8. The fraction of time the total MSI calculation could use 1, 2, 4, 8 or more nodes.

ORNL-DWG 87-17447

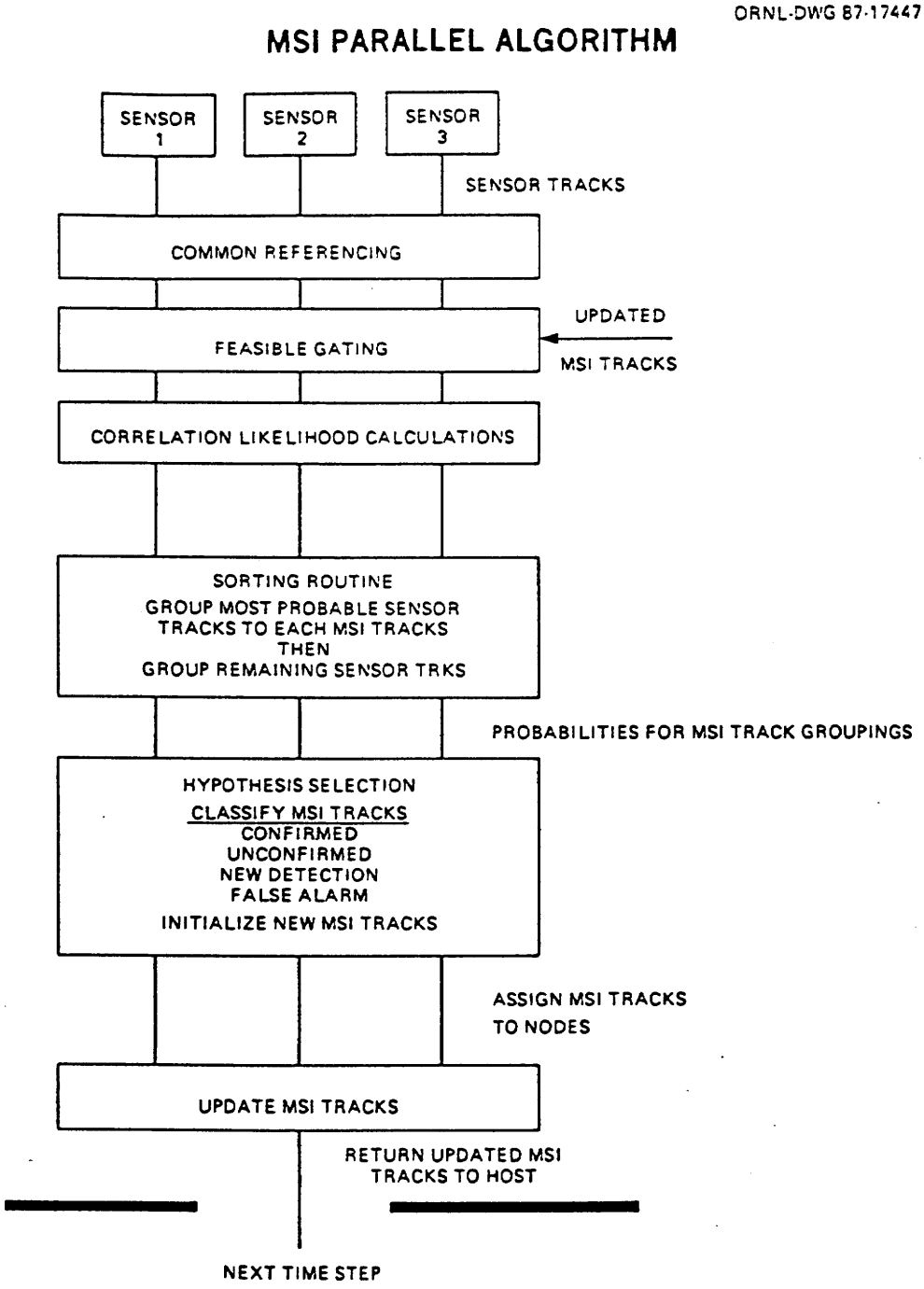


Fig. 9. Schematic of a completely concurrent MSI program.