On Mapping a Tracking Algorithm Onto Parallel Processors

KRISHNA R. PATTIPATI, Member, IEEE

THOMAS KURIEN, Member, IEEE
Alphateh, Inc.
Burlington, MA

RONG-TAY LEE, Member, IEEE
Valid Logic
San Jose, CA

PETER B. LUH, Member, IEEE
University of Connecticut

The problem of mapping the tasks of a multi-target tracking algorithm onto parallel computing architectures to maximize speedup is considered. An asymptotically optimal mapping algorithm is developed and applied to study the effects of task granularity and processor architectures on the speedup. From the simulation results, it is concluded that task granularity, and the parallelization of clustering and global hypotheses formation stages of the tracking algorithm are major determinants of speedup.

Manuscript received March 1, 1989.

IEEE Log No. 38445.

This work was supported by RADC/COTC, Griffis Air Force Base, Rome, NY. The work of K. R. Pattipati was also partially supported by the Office of Naval Research under Contract N00014-87-K-0057.

Authors' addresses: K. R. Pattipati and P. B. Luh, Dep't. of Electrical and Systems Engineering, University of Connecticut, Storrs, CT 06269-3157; T. Kurien, Alphatech, Inc., 111 Middlesex Turnpike, Burlington, MA 01803; R.-T. Lee, Valid Logic, San Jose, CA 95134.

0018-9251/90/0900-0774 $1.00

I. INTRODUCTION

A. Motivation

Ballistic missile defense and airborne surveillance require identification and tracking of several hundred targets in real time. Multitarget tracking algorithms designed for these problems demand large computational resources, which generally cannot be satisfied with conventional von Neumann-type processors. Fortunately, since multitarget tracking involves the simultaneous tracking of many targets, the algorithm consists of many computational tasks (i.e., units of computation that may be assigned to a processor), which may be run in parallel.

With the advent of multiprocessor architectures, adapting multitarget tracking algorithms to these new parallel computing architectures poses a challenging problem. Past studies in this area [3, 4, 17] have two features in common. First, they consider "operational" multitarget tracking algorithms using simple rules for associating target tracks with returns, and $\alpha - \beta$ filters for target state estimation. Second, they examine the adaptation of the tracking algorithms onto single instruction, multiple data stream (SIMD) array processors, which have many processing elements executing the same set of instructions broadcast by a central control unit.

During the past ten years, several multitarget tracking algorithms have been developed [1, 8, 15, 19, 24]. These algorithms permit the evaluation of a mathematically optimal set of target tracks based on models of target motion and of the environment. Evaluation of the optimal solution requires substantially larger computational resources than the operational tracking algorithms; however, by specifying a set of heuristics, the computational requirements of these model-based tracking algorithms can be reduced to levels close to those of operational algorithms. The advantage of designing an algorithm using the model-based approach is that it provides a systematic method for initiating new target tracks and eliminating unlikely ones. This approach is markedly different from operational tracking algorithms, and its implementation on parallel computing architectures has not been previously studied in detail.

We are concerned here with the mapping of a multitarget tracking algorithm onto multiple instruction, multiple data stream (MIMD) multiprocessors, wherein processors may be homogeneous (i.e., identical) or heterogeneous (i.e., nonidentical). There are four important factors that contribute to the completion time (or equivalently, speedup) of an algorithm on a MIMD multiprocessor: 1) task partitioning, 2) task allocation and sequencing, 3) the interconnection topology of processors and the capacity of each communication link in the interconnection network, and 4) the speed of

each processor. The partitioning problem refers to the selection of the level of granularity used to represent the tasks of an application algorithm. Task partitioning determines the amount of computation required by each task, the precedence relations among the tasks of the algorithm, and the amount of data transmitted between each pair of tasks. The allocation and sequencing strategy determines the assignment of the set of tasks to member processors. The inter-task communication is constrained by the interconnection topology of processors. The precedence relations among the tasks of the algorithm impose synchronization requirements, i.e., a task cannot begin execution until all the tasks preceding it have been completed. Finally, the instruction execution rates of processors determine the processing times for tasks, and have a direct bearing on the completion time of the algorithm.

B. Problem Complexity and Previous Mapping Approaches

The mapping problem is NP-hard, i.e., the computational requirements of an optimal solution increase exponentially with the number of tasks and the number of processors [18]. Therefore, all practical mapping algorithms incorporate heuristics. Bokhari [5] formulated the mapping problem as one of maximizing the cardinality of mapping, i.e., the number of edges of the task graph that fall on the links of the processor graph. The heuristic algorithm proposed by Bokhari involves an iterative interchange of mapped edges to increase the cardinality of mapping at each iteration. It was assumed that all the processors and links were identical, and that the computation was regular, i.e., all tasks had identical processing times and the amount of data transferred between any pair of tasks was the same. In addition, synchronization requirements were neglected.

Kasahara and Narita [7] proposed a heuristic method, termed critical path/most immediate successors first (CP/MISF), to (locally) minimize the completion time of the application algorithm. In their approach, the critical path method is first employed to form a priority list of tasks in nonincreasing order of task levels and the number of immediate successor tasks, wherein the level of a task i is defined as the longest path length from the terminal node (denoting the termination of the algorithm) to node i representing task i. Once a priority list is formed, the tasks are assigned sequentially to processors to locally minimize the completion time. The CP/MISF method considers the precedence relations among the tasks of an application algorithm. However, data communication requirements were neglected.

More recently, we have formulated and solved a general mapping problem of allocating the tasks of an algorithm onto parallel computing architectures to minimize the completion time of the algorithm [10]. We view the task allocation problem as one of assigning the nodes of a finite, directed, acyclic task graph representing the logical and data dependencies among the tasks of an algorithm on to the nodes of a finite, undirected processor graph denoting the parallel computing architecture. The key features of our formulation include: 1) explicit consideration of precedence restrictions among tasks and, hence, the synchronization delays, 2) sequencing of data messages to account for queueing delays at communication links, and 3) the incorporation of storage, security, and fault-tolerance requirements. Four algorithms have been developed in [10, 13, 20] to solve the mapping problem. The first algorithm is a two-stage heuristic algorithm that determines the order of task allocation based on the critical path method (CPM), and then employs one-step optimization method to determine task allocation that (locally) minimizes the completion time. The second algorithm employs the idea of pair-wise exchange on task allocation order to improve the performance of the two-stage heuristic algorithm, and is a practical heuristic for large mapping problems. The third algorithm is an optimal mapping strategy (in the class of permutation schedules) using the A^* algorithm [14], wherein a lower bound is obtained from the CPM. Finally, we developed an ϵ-optimal algorithm, which guarantees that the solution is within $1+\epsilon$ of the optimal solution. In addition, we established the asymptotic optimality of the two-stage heuristic algorithm for tasks with fork-join (i.e., series-parallel) structures, such as those that arise in multitarget tracking, assuming that the computation time is much larger than the communication time [10]. Extensive computational experiments on hundreds of random task graphs have shown that the heuristic algorithm provides optimal task allocation when the ratio of computation time over communication time is very large or very small, and that the pairwise exchange algorithm provides uniformly good allocations for all values of this ratio [13, 20].

We apply the two-stage heuristic mapping algorithm to the multitarget tracking problem. The primary objective is to investigate the effects of task granularity and processor architectures on the achievable speedup of the tracking algorithm. The directed graph representation of the tracking algorithm showed that the tasks have fork-join structure and that the computation time of tasks dominates the communication time. Consequently, the results of heuristic and pair-wise exchange mapping algorithms were found to be identical. From the simulation results, it is concluded that the granularity of tasks is a major determinant of speedup. In addition, the speedup can be improved if clustering and global hypotheses formation stages of the tracking algorithm are parallelized.

C. Organization of the Paper

This paper is organized as follows. In Section II, we present a mathematical formulation of the mapping problem. In Section III, we discuss a heuristic mapping algorithm, and illustrate it by means of a simple example. In Section IV, we discuss a multitarget tracking algorithm, and characterize it as an acyclic task graph. In Section V, we present the results of applying the mapping algorithm to the tracking problem. Finally, concluding remarks and future research issues are provided in Section VI.

II. MAPPING PROBLEM FORMULATION

A. Issues of Task Granularity

Parallelism in an application algorithm can be analyzed and exploited only if the flow of data, the data dependencies among various tasks, and the timing requirements of the algorithm are clearly understood. The data flow and the data dependencies of an algorithm may be represented in the form of a directed graph. The nodes represent tasks in the algorithm, the input arcs to a node represent the data required by the task, and the output arcs from a node represent the data produced by the task.

We measure the granularity of an algorithm in terms of the number of elements of computation (or operations such as additions, multiplications, comparisons, etc.). An algorithm generally exhibits parallelism at various levels of granularity. The instruction level can be thought of as the finest level of granularity. By aggregating operations at each level, coarser levels may be constructed. If the algorithm can be mapped on to a multiprocessor architecture that exploits parallelism down to the finest level of granularity, then it would appear that the maximum possible speedup can be achieved. Unfortunately, the finer the granularity of operations, the larger are the communication and synchronization requirements. On the other hand, the coarser the level of granularity, the greater is the computational requirements at a single processor. Clearly, there exists an optimum choice for the granularity of operations at which the tasks of an algorithm should be defined.

B. Problem Formulation

Given the level of task granularity, the mapping problem is one of assigning the nodes of a task graph onto the nodes of a processor graph, such that the completion time is minimized. The task graph is a directed, acyclic graph, $G_t = (V_t, E_t)$, where $V_t = \{i : i = 1,2,\ldots,L\}$ is the set of vertices (nodes) denoting the tasks of the application algorithm, and $E_t = \{\langle i,j\rangle : i,j = 1,2,\ldots,L;\ i \neq j\}$ is the set of *directed* edges (arcs, links) representing the inter-task communication between pairs of tasks i and j, and the partially ordered constraint that task i must precede task j. Each node i of the task graph is parameterized by s_i, the service demand of task i measured in terms of millions of instructions. Each directed edge $\langle i,j\rangle$ of the task graph is parameterized by v_{ij}, the amount of data transmitted between tasks i and j measured in terms of bits. We assume, without loss of generality, that the task graph G_t consists of a start (source) node and a terminal (sink) node, i.e., the task graph G_t is such that each node can be reached if we go forward from the start node, or go backward from the terminal node. If the task graph does not have a start node and/or a terminal node, a dummy start node/terminal node can always be added to the task graph such that the number of instructions of the dummy node and the data transmitted from the dummy node to other nodes of the task graph (and vice versa) are zero. For notational convenience, we assume that the terminal task corresponds to node L of the task graph, G_t.

In the same vein, the processor graph is an undirected graph, $G_p = (V_p, E_p)$, where $V_p = \{p : p = 1,2,\ldots,M\}$ is the set of vertices denoting the processors, and $E_p = \{(p,q) : p,q = 1,2,\ldots,M;\ p \neq q\}$ is the set of *undirected* edges depicting the communication links among processors. Each node p of the processor graph is parameterized by μ_p, the service rate of processor p measured in terms of millions of instructions per second. Each edge (p,q) of the processor graph G_p is parameterized by the link capacity, c_{pq} measured in bits per second. We assume that the data communication between a pair of processors (k_0,q) follows the shortest path $(k_0,k_1,k_2,\ldots,k_n,q)$, where $k_0,k_1,k_2,\ldots,k_n,q$ are the nodes on the shortest path, and that task sequencing at each processor is nonpreemptive. That is, once processing begins on a task, it is processed to completion without interruption. Furthermore, we assume that each node of the processor graph contains an execution processor and a communication processor, so that task execution and data communication can be serviced simultaneously at the same node.

Formally, a mapping is a partition of the task set $V_t = \{1,2,\ldots,L\}$ into M disjoint ordered sets $T_1,T_2,\ldots,T_M$:

$$T_q = \{q_1,q_2,\ldots,q_{k_q}\}, \qquad k_q \geq 0 \tag{1}$$

and

$$\bigcup_{1\leq q\leq M} T_q = \{1,2,\ldots,L\} \tag{2}$$

that satisfy the precedence relationships of the task graph. In (1), T_q is the ordered set of tasks assigned to processor q, q_i is the ith task executed on processor q, and $k_q = 0$ implies that the corresponding ordered set T_q is a null set. Let β_i denote the set of immediate

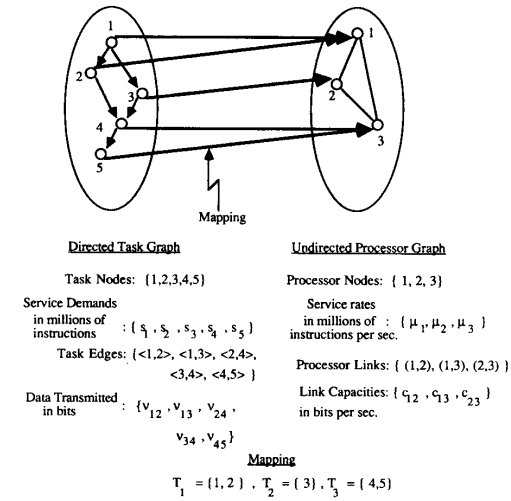


Fig. 1. Algorithm-architecture mapping problem.

parents of task i in the task graph, G_t. Then, the mapping problem can be stated succinctly as follows:

$$\min_{\{T_1,T_2,\ldots,T_M\}} C_L(T_1,T_2,\ldots,T_M) \tag{3}$$

where $C_L(T_1,T_2,\ldots,T_M)$ is the completion time of the terminal task L under the mapping $(T_1,T_2,\ldots,T_M)$. Note that the mapping must satisfy the precedence constraints, viz., task i cannot begin execution on processor q until the data from each parent task in β_i are available at processor q, $i \in V_t$. The algorithm-architecture mapping problem is illustrated in Fig. 1.

C. Performance Measures

The key performance measures that summarize the effectiveness of implementing an algorithm on a multiprocessor system are the speedup that is achievable and the efficiency of processor utilization. Both these measures will obviously depend on how the algorithm is mapped onto a multiprocessor architecture. Once the mapping $(T_1,T_2,\ldots,T_M)$ is known, the speedup, λ, can be obtained via

$$\lambda = \frac{\text{Execution Time on Fastest Processor}}{\text{Execution Time on Multi-processor}} = \left[\sum_{i\in V_t} s_i\right] \Big/ [\mu_f C_L(T_1,T_2,\ldots,T_M)] \tag{4}$$

where $\mu_f = \max_{q\in V_p} \mu_q$ is the service rate of the fastest processor in the multiprocessor system. Note that we have generalized the classical definition of speedup that assumes identical processors [21]. The processor utilization, U_q, is obtained from

$$U_q = \frac{\text{Busy Time of } q\text{th Processor}}{\text{Execution Time on Multi-processor}} = \left[\sum_{i\in T_q} \frac{s_i}{\mu_q}\right] \Big/ C_L(T_1,T_2,\ldots,T_M). \tag{5}$$

The utilization of links can be derived in a similar manner. The efficiency η is defined as

$$\eta = \frac{\text{Actual Speedup}}{\text{Number of Processors}} = \frac{\lambda}{M}. \tag{6a}$$

When the processors are identical, the efficiency η is related to utilization U_q via:

$$\eta = \frac{\sum_{q\in V_p} U_q}{M} \tag{6b}$$

For a homogeneous M-processor system (i.e., service rates $\mu_p = \mu$ for $p = 1,2,\ldots,M$) executing an algorithm with a series-parallel task graph structure (e.g., a multitarget tracking algorithm, optimization algorithms based on primal-dual methods [22]), we can derive an upper bound on the speedup of the algorithm as follows. Let the L tasks in the series-parallel task graph be divided into K sequential steps, wherein step i consists of w_i parallel (and identical) tasks such that $\sum_{i=1}^{K} w_i = L$. Further, let $t_i = s_i/\mu$ be the average execution time on a processor for each task of step i. Since w_i may not be perfectly divisible by M, some processors are forced to be idle, and as a result, the maximum achievable efficiency of the multiprocessor system will be less than 1. Consequently, from (6a), the maximum achievable speedup $\overline{\lambda}$ will also be less than M. Indeed, the maximum achievable speedup, $\overline{\lambda}$ is obtained (by neglecting communication delays) via:

$$\overline{\lambda} = \frac{\sum_{i=1}^{K} w_i t_i}{\sum_{i=1}^{K} \left\lceil \frac{w_i}{M} \right\rceil t_i} \tag{7}$$

where $\lceil x \rceil$ denotes the smallest integer greater than x. In Section V, we compare the actual achievable speedup λ of (4) with the upper bound $\overline{\lambda}$ of (7) as a function of the number of processors M for a multitarget tracking algorithm mapped onto homogeneous hypercube and shared memory multiprocessors.

III. HEURISTIC MAPPING ALGORITHM

A. Key Timing Equation

For the general mapping problem discussed earlier, the completion time of a task i on an assigned processor q is a function of: 1) the service demand of task i and the service rate of processor q, 2) the

time at which the data from immediate predecessor (or parent) tasks of task i are available at processor q, and 3) the time when processor q becomes available. The delay due to data dependency accounts for synchronization requirements and communication delays. The consideration of processor availability accounts for queueing delays due to previously assigned tasks. In the following, we derive an explicit equation for the evolution of completion time as a function of mapping.

Let $(T_1,T_2,\ldots,T_q,\ldots,T_M)$ denote a partial mapping and let $C_j(T_1,T_2,\ldots,T_M)$ be the corresponding completion time of task j, $j \in V_t$. Suppose we want to assign a task i to processor q so that the new mapping is $(T_1,T_2,\ldots,T_q \cup i,\ldots,T_M)$. We would like to derive an expression for the completion time of task i on processor q, $C_i(T_1,T_2,\ldots,T_q \cup i,\ldots,T_M)$, given the partial mapping and the corresponding completion times of prior tasks.

Since task i cannot begin execution on processor q until the data from predecessors of task i are available at processor q, and also until processor q completes its execution of the last task in the ordered set T_q, the completion time of task i is given by:

$$C_i(T_1,T_2,\ldots,T_q \cup i,\ldots,T_M) = \max[D_i^q, C_l(T_1,T_2,\ldots,T_q,\ldots,T_M)] + \frac{s_i}{\mu_q} \quad (8)$$

where D_i^q denotes the earliest time at which data from all the parents of task i become available at processor q, and l is the last task in the ordered set T_q. The earliest time at which data from all the parents of task i become available at processor q is:

$$D_i^q = \max_{j=1,2,\ldots,|\beta_i|} \beta_{i[j]}^q = B_{i[n]}^q \quad (9)$$

where $B_{i[j]}^q$ is the time at which data from the jth earliest completed parent task of task i became available at processor q, β_i is the set of parents of task i, and $|\beta_i|$ denotes the numbers of elements in the set β_i. To compute $B_{i[j]}^q$, we make the reasonable assumption that data communication from parent tasks takes place in the order in which they are completed, i.e., the first completed parent task sends its data first. The computation implied by (8) and (9) is illustrated in Figs. 2(a)–(b), where the max term in (8) is the start time of task i and s_i/μ_q is the service time of task i on processor q. An algorithm for the computation of $B_{i[j]}^q$ needed in the evaluation of D_i^q is included as Algorithm 1 in Appendix A.

In order to illustrate the timing equations (8) and (9), consider the simple example in Fig. 1. The timing equations for the mapping shown in Fig. 1 are given by Table I where, for notational clarity, we denoted processor i by p_i. The mapping is illustrated by a Gantt chart in Fig. 2(c), assuming $C_2 + \nu_{24}/c_{13} > C_3 + \nu_{34}/c_{23}$.

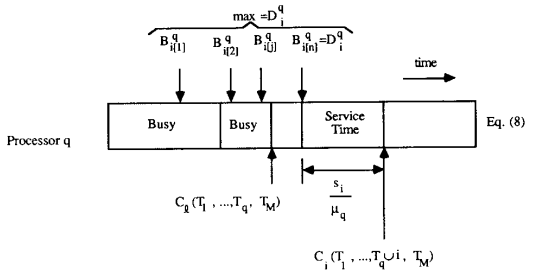


Fig. 2(a). Graphical illustration of mapping equations (8), (9).

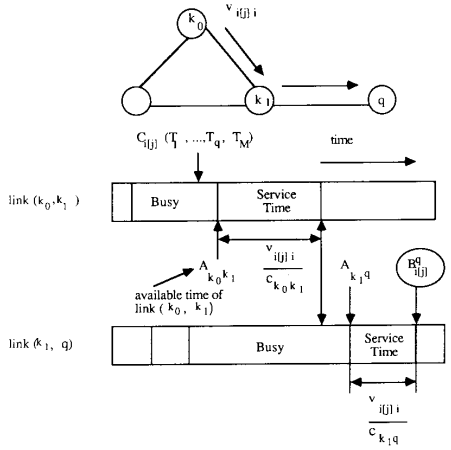


Fig. 2(b). Computation of $B_{i[j]}^q$ of (9).

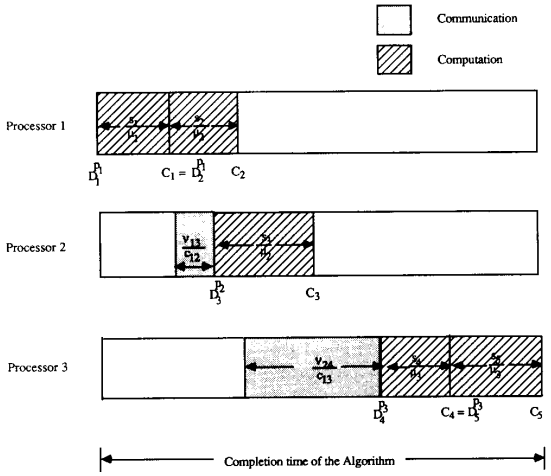


Fig. 2(c). Illustration of timing equations for example in Fig. 1.

B. Heuristic Mapping Algorithm

The heuristic algorithm consists of two stages. The first stage employs the concept of critical path to determine the order of task allocation, while the second stage sequentially allocates the tasks from the

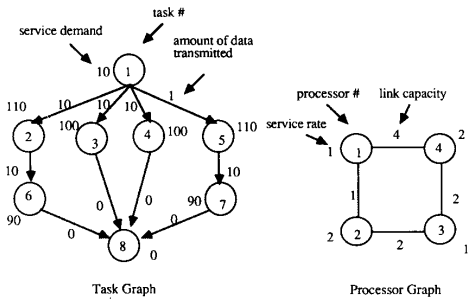


Fig. 3. Illustrative example.

TABLE I
Illustration of Timing Equations

Processor 1	Processor 2	Processor 3
$D_1^{p_1} = 0$	$D_3^{p_1} = C_1 + \frac{\nu_{13}}{c_{12}}$	$D_4^{p_3} = \max\left\{C_2 + \frac{\nu_{24}}{c_{13}}, C_3 + \frac{\nu_{34}}{c_{23}}\right\}$
$C_1 = \frac{s_1}{\mu_1}$	$C_3 = C_1 + \frac{\nu_{13}}{c_{12}} + \frac{s_3}{\mu_2}$	$C_4 = D_4^{p_3} + \frac{s_4}{\mu_3}$
$D_2^{p_1} = C_1$		$D_5^{p_3} = C_4$
$C_2 = C_1 + \frac{s_2}{\mu_2}$		$C_5 = C_4 + \frac{s_5}{\mu_3}$

ordered list to processors, so that the completion time of the tasks is a minimum. To determine the order of task allocation, we define the level of a task i as

$$l_i = \max_k \sum_{j \in \pi_k} \frac{s_j}{\mu_f} \tag{10}$$

where μ_f is the service rate of the fastest processor, s_j is the service demand of task j, and π_k denotes the kth path from task i to the terminal task. That is, l_i is the length of the critical path from task i to the terminal task. By construction, l_i is a lower bound on the completion time of a task graph rooted at task i. Following the level algorithm of Hu [6] and Sethi [16], the heuristic algorithm is based on the premise that tasks with larger levels should be executed earlier in the sequence. If several tasks have the same level, then the task with the greater number of successors should be allocated first. Thus, we construct the allocation order according to nonincreasing levels first, and nonincreasing successors next for tasks with the same level.

Once the priority list is constructed, we sequentially allocate tasks to processors to minimize the completion time (termed the greedy heuristic method), C_i, $i \in V_t$. In the case without constraints, task i is assigned to processor q^*, where

$$q^* = \arg\min_q C_i(T_1, T_2, \ldots, T_q \cup i, \ldots, T_M) \tag{11}$$

where arg denotes argument. The heuristic mapping algorithm is included as algorithm 2 in Appendix A.

The complexity of the heuristic mapping algorithm is $O(|E_t| + \gamma ML)$, where $\gamma \leq$ maximum number of parents of any task $\times$ number of links on the longest path between a pair of processors. For series-parallel task graphs, the two-stage heuristic mapping algorithm has been proven to be optimal, as the numbers of tasks in each parallel path tends to infinity and when computation times dominate the communication times [10].

C. Illustrative Example

Consider task and processor graphs as shown in Fig. 3. The level of the start task is the sum of the service demands on the critical path, 210, divided by the fastest service rate, 2, which is 105 in this case. The levels of other tasks can be derived from (10). The priority list of tasks is constructed according to nonincreasing levels first, and nonincreasing successors next for tasks with the same levels. The allocation order then is $O = (1,2,5,4,3,6,7,8)$ and the corresponding levels are $\mathbf{l} = (105,100,100,50,50,45,45,0)$. Let the completion time of task i be denoted by C_i and processor j be denoted by p_j. Then the highest level task, task 1, is assigned to one of the fastest processors, p_2, with $C_1 = 5$. The next task to be assigned is task 2. We assume that data communication takes place along a set of specified shortest paths. For example, suppose that we send one unit of data from p_2 to p_4. If the routing path is $\pi = (p_2, p_3, p_4)$, the communication time is $1/c_{23} + 1/c_{34} = \frac{1}{2} + \frac{1}{2} = 1$, where (p_i, p_j, p_k) represents the routing path from p_i to p_k, and c_{ij} is the link capacity between processors i and j. Instead, if the routing path is $\pi = (p_2, p_1, p_4)$, the communication time would be $1/c_{21} + 1/c_{14} = \frac{1}{1} + \frac{1}{4} = 1.25$. Hence, the shortest routing path is $\pi = (p_2, p_3, p_4)$. If task 2 is assigned to p_1, C_2 will be equal to C_1 plus the communication delay from p_2 to p_1 plus the service time of task 2 at p_1, i.e., $5 + \frac{10}{1} + \frac{110}{1} = 125$. In the case that task 2 is assigned to p_2, the completion time of task 2, C_2 at p_2 is $t + \frac{110}{2} = 60$. The completion time of task 2 at p_3 and p_4 can be derived in a similar manner. The best choice for task 2 is p_2 with $C_2 = 60$. The next task to be allocated is task 5. Now C_5 at p_2 is $C_5 = A_2 + s_5/\mu_2 = 115$, where A_j is the available time of processor j. We find that assigning task 5 to p_4 will provide the earliest C_5, so p_4 will be our selection. The results of the mapping algorithms are shown in Table II. The completion time of the mapping provided by the heuristic algorithm is 153.5. The pair-wise exchange algorithm, discussed in [13, 20], provides a mapping with a completion time of 115, which is an improvement of over 30 percent for this example. The A^* (optimal) algorithm discussed in [13, 20] provides a mapping with a completion time of 110.5. Intuitively, the optimal mapping should balance the work load of processors, if communication time is much less

TABLE II
Computational Results for Different Mapping Algorithms

	Heuristic Algorithm			Pair-wise Exchange Algorithm			A* Algorithm		
Priority List	task no.	proc no.	completion time	task no.	proc no.	completion time	task no.	proc no.	completion time
1	1	2	5.0	1	2	5.0	1	4	5.0
2	2	2	60.0	2	2	60.0	4	1	107.5
3	5	4	61.0	5	4	61.0	5	2	61.0
4	4	2	110.0	6	2	105.0	2	4	60.0
5	3	3	110.5	7	4	106.0	7	2	106.0
6	6	4	115.0	4	3	110.5	6	4	105.0
7	7	1	153.5	3	1	115.0	3	3	110.5
8	8	1	153.5	8	1	115.0	8	4	110.5

than the computation time. For example, allocation of tasks 2 and 6 to p_4, task 3 to p_1 (or p_3), task 4 to p_3 (or p_1), and tasks 5 and 7 to p_2 will balance the processing times at different processors after task 1 has been completed. Although the heuristic and pair-wise exchange algorithms do not provide optimal mapping for this example, both of these algorithms yield identical results for the multitarget tracking problem discussed in the next two sections.

IV. APPLICATION TO MULTITARGET TRACKING

A. Overview of Multitarget Tracking Algorithm [8, 19]

Among the various model-based multitarget tracking algorithms proposed in the literature, we confine our attention to the one proposed in [8, 19]. We have chosen this multitarget tracking algorithm because it has a firm mathematical basis and has been demonstrated with real data. This algorithm, termed *multitracker*, uses a hybrid state estimation approach for the joint problem of associating sensor reports with targets and estimating the kinematic state of each target. The general hybrid model consists of continuous and discrete-valued states. Using measurements related to the hybrid state, it is possible to compute an optimal (in the sense of minimum mean-square-error or maximum a-posteriori) estimate of the hybrid state. Variables in the multitarget tracking model can be identified with the generic hybrid model as follows. The kinematic states (generally position and velocity) of all existing targets constitute the continuous-valued states; indicators for target dynamic model status (constant velocity model versus a maneuver model) and sensor report status (associated with target, clutter) constitute the discrete-valued states; the noisy range, azimuth, and range rate measurements from targets and clutter at every scan constitute the measurements.

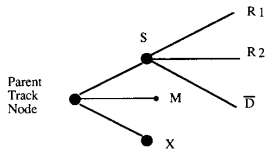


Fig. 4. Track splitting to account for different target dynamics and different measurement associations.

The hybrid state estimation approach dictates the form of the optimal solution for the multitarget tracking problem; however, the enumeration of all possible values of the discrete-valued state (which we refer to as the *global hypotheses*) and the computation of their likelihoods poses a difficult combinatorial problem.

Multitracker uses a track-oriented approach for generating the global hypotheses and computing their likelihoods [8, 19]. This approach maintains a set of target trees and a list of global hypotheses. The root of each target tree represents the birth of the target. The branches represent the different dynamic models assumed by the target and the different reports the target can be associated with in subsequent scans. A trace of the successive branches from the root of the tree to a leaf represents a potential track for the target. The leaf of each such trace in unique, and it is referred to as a track hypothesis (or a track) for the target.

Track hypotheses are first constructed recursively from one scan to the next. At each scan, new branches for existing tracks in each target tree are formed in two steps: first new branches are formed corresponding to different dynamic models for the target; each of these branches is then split to account for feasible associations with sensor reports in the scan. This is illustrated in Fig. 4, where we have first split the target track to account for a nominally constant velocity model (S), a maneuver model (M), and a target termination (X); we have then split the nominally constant velocity model track to account for a missed detection ($\overline{D}$) and associations with report R1 or report R2. Each sensor report is also used to initiate a new target track since it could potentially have originated from a new target.

Global hypotheses are formed by combining tracks from different target trees. Each global hypothesis is maintained as a list of pointers which point to nodes in the target trees as shown in Fig. 5. These nodes represent the combination of target tracks postulated by the global hypothesis. Assuming that there is at most one report per target in each scan, the collection of pointers in any one global hypothesis cannot point to more than one track hypothesis within the same target tree. The dotted line across the target trees in Fig. 5 indicates the points at which new branches have been grown for existing target track hypotheses using

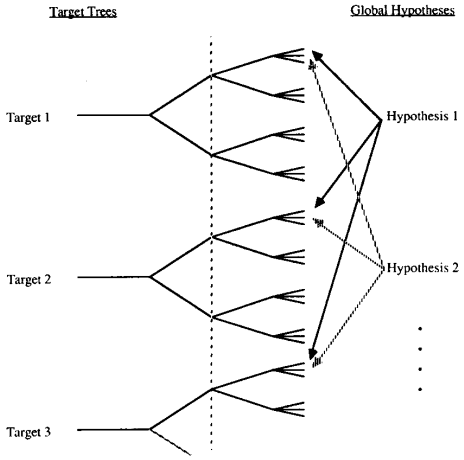


Fig. 5. Representation of global hypotheses.

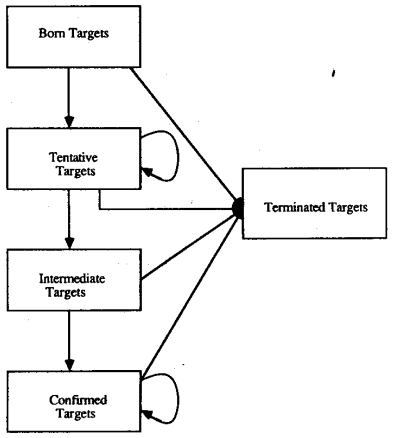


Fig. 6. Classification of targets.

different dynamic models for the target and sensor reports in the most recent scan. Probabilities of the global hypotheses are evaluated from the probabilities of the track hypotheses. The recursive procedures for evaluating these probabilities have been provided in [8, 19].

The two-step procedure used by the track-oriented approach provides a systematic method for postulating and evaluating the probabilities of global hypotheses. It also enables efficient data structures to be defined for postulating track and global hypotheses. These data structures provide an effective means for postulating pertinent hypotheses such as target track initiation, target maneuvers, and missed detections in multitarget tracking. Specific advantages resulting from the use of the track-oriented approach compared to the target-oriented approach recommended by Reid [15] are provided in [19].

The track-oriented approach represents a systematic method for constructing the global hypotheses using a hybrid system model of the multitarget tracking problem. It can be used to construct the optimal solution to the multitarget tracking problem, if computational resources can support the postulation of the exponentially increasing number of global hypotheses [1, 2, 12, 15]. In order to construct a practical algorithm, all the unlikely global hypotheses have to be eliminated at every scan. Eliminations can be enforced either by screening (prior to generation of hypotheses) or by pruning (after the generation of hypotheses). We discuss four key elimination techniques included in multitracker: gating, classification, clustering, and N-scan approximation. The first three represent screening techniques which limit the number of reports associated with a target track, and the last one represents a pruning technique which limits the number of tracks postulated for a target.

Gating: Gating is a screening technique that eliminates the unlikely associations of reports with targets. It proves to be effective in reducing the number of unlikely tracks, and it has been used in most tracking algorithms designed in the past [2, 15]. The gating procedure consists of constructing a region around the predicted target position, and choosing only those reports which lie within this region for associating with the target track.

Classification of Targets: As indicated earlier, the number of global hypotheses is a combinatorial function of the number of track hypotheses. Consequently, multitracker confines attention to track hypotheses of only a selected set of targets while forming global hypotheses. This results in a dramatic reduction in the number of global hypotheses. The selection of targets is based on the criteria that the probability of the most likely track of the target is greater than a specified threshold, and that the age of the target is $\geq \delta$. Targets which satisfy these criteria are referred to as *confirmed* targets. The remaining targets are divided into three categories. Targets having an age $\geq (\delta - 1)$ are referred to as *intermediate* targets. It is from this group that confirmed targets are selected. Targets with an age ≥ 1 are referred to as *born* targets, and targets with an age ≥ 2 and which have not reached the intermediate category are referred to as *tentative* targets (Fig. 6).

Precise conditions for promoting targets from one category to the other will be application dependent and are based on the surveillance system requirements. If confirmed targets were the only targets to be declared or displayed to the human surveillance operator, then these conditions for promoting targets define the target track initiation procedure. Furthermore, this grouping scheme allows different criteria to be applied for screening and pruning targets belonging to different categories. These include the following two: 1) The imposition of stringent pruning

requirements for targets in lower categories. For example, a born target is allowed to postulate fewer missed detections compared with that allowed for a confirmed target; 2) The imposition of restrictions on the number of branches allowed for targets in lower categories. For example, a born target is allowed fewer branches compared with that allowed for a confirmed target; only confirmed targets are allowed to postulate maneuver branches.

Clustering: Since the purpose of forming global hypotheses is to resolve ambiguities in the association of reports to targets, another form of grouping is possible. If the track hypotheses for a set of targets do not share any common reports, it is not necessary to form global hypotheses for those targets. This motivates the idea of partitioning targets into separate clusters for the purpose of forming global hypotheses. Targets within each cluster share common reports, whereas targets in different clusters do not share any common reports. By forming such clusters, global hypotheses can be formed and resolved independently for each cluster; consequently, the combinatorial problem associated with forming global hypotheses is reduced significantly.

Targets may be clustered by determining those targets which share the same reports. The clustering algorithm included in multitracker [19] utilizes information about report-to-target assignments during the gating operation and forms clusters in two steps: first it forms subclusters which represent the targets which are associated with the same report; it then forms clusters by recursively combining subclusters which share common targets.

N-Scan Approximation: The optimal multitarget tracking algorithm postulates all possible report-to-target associations for all available scans of sensor reports prior to evaluating the most likely set of associations in each scan. An algorithm that formulates all possible associations in $NSCAN$ subsequent scans prior to evaluating the most likely association in the current scan is referred to as an N-scan approximation algorithm. For example, the Probabilistic Data Association (PDA) algorithm [2] corresponds to an N-scan algorithm with $NSCAN = 0$. Multitracker defines $NSCAN$ as a parameter whose value can be set based on the requirements of the surveillance system.

B. Parameters that Control Computational and Communication Requirements of Multitracker [9]

Computational requirements of an algorithm may be specified in terms of the number of instructions required to exercise the algorithm on a digital computer. In the same vein, the communication requirements of an algorithm may be specified in terms of the number of bits transferred among the various constituent tasks of the algorithm. For an iterative algorithm, the computational and communication requirements derived for a single iteration are representative for the algorithm provided that the algorithm reaches a steady-state operational condition. The multitarget tracking algorithm may be viewed as an itertive algorithm, wherein the same set of steps (postulation of track hypotheses, postulation of global hypotheses, etc.) are repeated at each scan. A steady-state operational condition is certainly not achieved by the optimal multitarget tracking algorithm, since its computational requirements grow exponentially with each scan. However, it is reasonable to assume that multitracker will reach steady-state operational conditions after the first few scans, since it incorporates all the screening and pruning techniques discussed above.

TABLE III
Definition of Multi-target Tracking Variables

Term	Meaning
N_c	number of confirmed targets
N_i	number of intermediate targets
N_t	number of tentative targets
N_b	number of born targets
B_c	number of tracks per confirmed target
B_i	number of tracks per intermediate target
B_t	number of branches per tentative target
B_b	number of branches per Born target
R_c	average number of reports in the gate of each Confirmed track
R_i	average number of reports in the gate of each Intermediate track
R_t	average number of reports in the gate of each Tentative track
R_b	average number of reports in the gate of each Born track
N_r	number of returns per scan
NS	number of subclusters
NTS	number of targets in each subcluster
NC	number of connected clusters
NTC	number of targets in each cluster
$NSCAN$	number of further scans to be processed prior to resolving reports in the current scan
NGH	maximum number of global hypotheses formed per cluster

Parameters that control the screening and pruning of hypotheses in multitracker are referred to as algorithm parameters. These parameters, listed in Table III are chosen based on the requirements of the surveillance system. For example, 1) the number of confirmed targets that the algorithm should be able to accommodate should correspond to the maximum number of targets anticipated within the surveillance region, 2) the number of targets postulated for intermediate, tentative, and born targets should be based on the expected target birth and death distributions, as well as clutter distribution, and 3) the number of tracks permitted for each target and the number of targets in each cluster should take into account the clutter density, the target density, and the

probability of target detection.

Once these parameters are selected in multitracker, they restrict the number of targets, the number of track hypotheses per target, and the number of global hypotheses retained at each scan. Consequently, they have a bounding influence on the number of instructions per scan (and the data structures) required by the various tasks of the multitracker, as well as the data transferred among the tasks. Specifically, the target-branch pairs $\{(N_x,B_x),\ x = c,i,t,b\}$ bound the number of branches and targets postulated by the algorithm, NTC bounds the number of targets in each cluster, $NSCANS$ limits the number of scans over which target tracks are examined for resolving global hypotheses, and NGH limits the number of global hypotheses that are postulated for each cluster.

C. Determination of Computational and Communication Requirements and Task Graph of Multitracker

Here we evaluate the computational requirements in terms of the number of arithmetic operations required by multitracker at each scan. For this analysis, we confine our attention to the arithmetic operations of floating point multiplications, floating point additions, and integer number comparisons. All floating point operations involve real numbers; subtractions are treated as additions and divisions are treated as multiplications. The communication requirements are evaluated in terms of the number of bits transferred among the tasks of multitracker.

The processing performed by multitracker during each scan may be partitioned into five functional steps: 1) prediction of all tracks postulated in the previous scan, 2) gating of all reports for each predicted track, 3) update of all predicted tracks with reports which lie within the gate of the track, 4) clustering of updated tracks, and 5) the formation of global hypotheses and the computation of their likelihoods. Prediction, gating, and update of each track involve precise equations, and so we can calculate the exact requirements to perform these steps for each track; however, since the number of tracks postulated at each scan is scenario dependent, we are able to evaluate only an upper bound on the total requirements for these three functional steps using the algorithm parameters. The requirements of the fourth and fifth steps are scenario dependent, since they are a function of the number of tracks postulated for each target; here again, we are able to evaluate only an upper bound on the requirements using the algorithm parameters. An outline of the evaluation of these upper bounds on the computational and communication requirements for each of the steps is provided below.

1) *Track Prediction*: The (Kalman) prediction equations for each target track are given by

$$\hat{x}(k+1 \mid k) = \Phi(k)\hat{x}(k \mid k) \tag{12}$$

$$P(k+1 \mid k) = \Phi(k)P(k \mid k)\Phi^T(k) + Q(k) \tag{13}$$

where $\hat{x}$ is a n-dimensional vector which represents the track state estimate, Φ is the state transition matrix associated with the target dynamics, P is the error covariance matrix, Q is the covariance matrix of the process noise in the dynamic model, k is the time index, and (j/i) represents an estimate at time j based on information up to and including time i. The number of operations required to evaluate these equations for a single track are

$$\text{Multiplications}: \quad \tfrac{3}{2}(n^3 + n^2)$$

$$\text{Additions}: \quad \tfrac{3}{2}(n^3 - n).$$

The amount of data to be transmitted to the track prediction function per track are shown in Table IV. The target identification (ID) index is an 8 bit integer, the track ID is a 4 bit integer, the track state estimate $\hat{x}$ is assumed to be n real words with 16 bits per word, and the symmetric covariance matrices P and Q are assumed to be $n(n+1)/2$ double precision real words per matrix, with 32 bits per word. Thus, the amount of data to be transmitted to the track prediction function for a single track is $[16n(2n+3)+12]$ bits. Upper bounds on the computational and communication requirements for predicting tracks in a given category are obtained by multiplying the corresponding operations and the data transferred by the factor N_xB_x, where $x = c,i,t,b$.

TABLE IV
Data to be Transmitted to Track Prediction Function per Track

Variable	Words	Bits/Word
target ID	1	8 bits
track ID	1	4 bits
x	n	16 bits
P	$0.5n(n+1)$	32 bits
Q	$0.5n(n+1)$	32 bits

2) *Gating*: This function selects, for each track, reports whose square of the innovations (ν) are less than their corresponding variances, i.e.,

$$\nu_i^2(k) < \gamma[H(k)P(k \mid k-1)H(k)^T + R(k)]_{ii}, \quad i = 1,2,\ldots,m \tag{14}$$

$$\nu_i(k) = y_i(k) - [H(k)\hat{x}(k \mid k-1)]_i, \quad i = 1,2,\ldots,m \tag{15}$$

where γ is an algorithm tuning parameter (typically $\gamma = 3$), H is the measurement matrix, R is the covariance matrix of the measurement noise in the measurement model, y is an m-dimensional vector of measurements in a sensor report, $[a]_i$ denotes

the ith element of the vector a, $[A]_{ii}$ denotes the ith diagonal element of the square matrix A, and m is the dimension of the measurement vector y. The number of operations to evaluate these equations for a single track are

$$\text{Multiplications:} \quad m(n^2+2n)+m(N_r)$$
$$\text{Additions:} \quad m(n^2+n-1)+m(N_r).$$

Similarly, the data to be transmitted to the gating function are obtained as follows. The data to be transferred include the target ID, the track ID, the state estimate, the symmetric error covariance matrix, the measurements in a scan, and the diagonal measurement noise covariance matrix. The number of bits to be transferred for a single track is $[16n(n+2)+m(16N_r+32)+12]$ bits, where N_r is the number of reports per scan. An upper bound on the computational requirements for the gating of all tracks in a category, and the concomitant communication requirements to the gating function are obtained by multiplying the above operations and the data transferred by the factor N_xB_x, where $x = c,i,t,b$.

3) *Track Update*: The (Kalman) update equations for each track are given by

$$\hat{x}(k \mid k) = \hat{x}(k \mid k-1) + K(k) \times [y(k) - H(k)\hat{x}(k \mid k-1)] \tag{16}$$

$$P(k \mid k) = [I - K(k)H(k)]P(k \mid k-1) \tag{17}$$

$$K(k) = P(k \mid k-1)H^T(k) \times [H(k)P(k \mid k-1)H^T(k) + R(k)]^{-1} \tag{18}$$

where $K(k)$ denotes the Kalman gain at time k. Assuming that the operations performed during gating are repeated here, the number of operations to evaluate these equations and the probability for a single track are

$$\text{Multiplications:} \quad m\left(3\frac{n^2}{2}+9\frac{n}{2}+4\right)$$
$$\text{Additions:} \quad \frac{m}{2}(3n^2+5n+1).$$

The data to be transferred to the track update function include the target ID, the track ID, the state estimate, the symmetric error covariance matrix, the report ID, the measurements in a scan, and the diagonal measurement noise covariance matrix. The number of bits transferred per track is $[16n(n+2)+28+48m]$ bits. An upper bound for the computational requirements for updating all the tracks in a given category and the corresponding communication requirements are obtained by multiplying the above operations and the data transferred by the factor $N_xB_xR_x$, where $x = c,i,t,b$.

4) *Clustering*: This function compares the indices of targets in each subcluster with target indices in other subclusters and merges them when there is a match. An upper bound for the computational requirements for this function are

$$\text{Comparisons:} \quad \frac{NTS[NS(NS-1)]}{2}.$$

Similarly, the data to be transferred to the clustering function includes the report IDs over the current and the past $NSCAN$ scans, and the target IDs of confirmed targets. Since the report IDs are 16 bit integers and the target IDs are 8 bit integers, the communication requirements are $[16N_cB_c(NSCAN+1)+8N_cB_c]$ bits.

5) *Formation of Global Hypotheses*: This function forms combinations of confirmed targets where the report indices are unique for all targets in each combination. Upper bounds for the number of operations to perform this function for each cluster are

$$\text{Comparisons:} \quad [\min\{(BC+1)^{NTC}, NGH\}] \times \left[\frac{NTC(NTC+1)(2NTC+1)NSCAN}{6}\right]$$
$$\text{Additions:} \quad [\min\{(BC+1)^{NTC}, NGH\}][NTC-1].$$

The upper bound for the number of comparisons is evaluated as follows. We consider the case where we arbitrarily pick one track for a target, and compare its report indices over the past $NSCAN$ scans with a track from the next target. If the report indices are unique, then the tracks are "matched" in the sense that they can be included in a combination. This procedure is repeated for the third target and so on. Assuming that a match is found on the second try for the second target, the third try for the third target, and so on, the number of comparisons made to form a single global hypotheses is

$$[(1)*(2)+(2)*(3+(3)*(4)+\cdots+(NTC-1)*(NTC)] *NSCAN \le \left(\sum_{k=1}^{NTC} k^2\right) NSCAN$$

where NTC is assumed to be $< B_c$, the number of branches per confirmed target. Thus, the left-hand side of the above equation is less than $(NTC)(NTC+1)(2NTC+1)/6(NSCAN)$.

The maximum number of combinations that can be formed is

$$\min[(B_c+1)^{NTC}, NGH].$$

(We add 1 to Bc to account for the fact that some combinations may not include any branch of a specific target). Combining these two expression provides the upper bound for the number of comparisons.

The upper bound for the number of additions results from the fact that the logarithm of the probability of each global hypotheses is computed as a sum of the logarithms of the probabilities of the track hypotheses included in it.

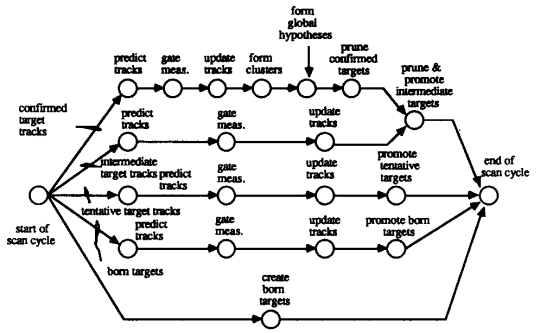


Fig. 7. Directed flow graph of operations in one scan of tracking algorithm.

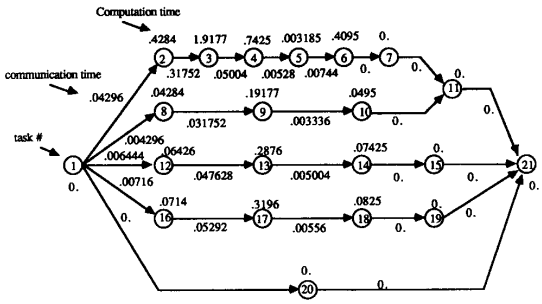


Fig. 8. Task graph of tracking algorithm corresponding to Fig. 7.

The data to be transferred to the global hypotheses formation function includes the report IDs, target IDs and the track probabilities. The track probabilities are assumed to be computed using double precision arithmetic. Since the target IDs and the track probabilities are assumed to be transmitted only for those targets in a cluster, the amount of data transferred per cluster is $[16(NTC)B_c(NSCAN+1)+12(NTC)B_c+32(NTC)B_c]$ bits.

Three remarks need to be made here. First, we have neglected the operations associated with pruning of targets, promotion of targets to the next higher category, and the creation of born targets. Second, the pruning and promotion steps result in an R_x : 1 transformation in the number of tracks of each category $x = c,i,t,b$ to satisfy the steady-state assumption. Finally, the promotion of intermediate targets is conditioned upon the available storage in the confirmed target data structure, and can be performed only after the pruning of confirmed targets; however, the promotion of the tentative and born targets is independent of the processing of the higher category, and can thus be performed at any time during the scan cycle.

A directed task graph summarizing the functional steps executed in one scan of multitracker is shown in Fig. 7. The execution times for each step are evaluated in Appendix B. If each node in the graph of Fig. 7 is defined as a task, then the task graph for multitracker with the execution and communication times can be constructed as shown in Fig. 8. Appendix C shows that, for a particular scenario, the maximum achievable speedup for multitracker (assuming zero communication delays) on multiprocessor architectures with 4, 8, 16 and 32 processors are 3.99, 7.95, 15.90, and 31.72, respectively. In the next section, we compare these maximum values with the actual speedup obtained via the heuristic mapping algorithm that explicitly considers communication and synchronization delays.

V. MAPPING RESULTS

In this section, we present three computational experiments to investigate the effects of task granularity and parallel processor architectures on the speedup in (4) of a multitarget tracking algorithm. In the first experiment, we compute the speedup as a function of task granularity on the critical path, i.e., the path with the largest computational requirements in the task graph. For the multitarget tracking problem, the critical path is the one involving confirmed target tracks and global hypotheses formation. In the first experiment, we vary the number of target tracks per task on the confirmed target path, and compare the speedup for various task sizes. In the second experiment, we vary the task sizes on all paths, and compare the corresponding speedups for hypercube and shared memory multiprocessor architectures. Finally, we also parallelize global hypotheses formation, and compare the resulting speedup with the maximum achievable speedup for different architectures. In all the experiments, we assume that the link (bus) bandwidth is 10 Mbits/s for hypercube (shared memory) architecture. The instruction speeds of processors are as described in Appendix B. All the experiments were run on a SUN 3/160 work station.

A. Experiment 1: Effects of Task Granularity on Critical Path

If the task graph of the multitarget tracking algorithm in Fig. 8 is mapped onto a 2-cube multiprocessor, the completion time for one scan of the tracking algorithm is 3.5013 s, with a concomitant speedup of 1.3381 (the completion time on a uniprocessor system is 4.684935 s. See Appendix C). The mapping obtained from the heuristic algorithm is as follows.

$T_1 = \{1,2,3,4,5,6,7\}$ on processor 1

$T_2 = \{16,17,18,19,21\}$ on processor 2

$T_3 = \{8,9,10,11,15,20\}$ on processor 3

$T_4 = \{12,13,14\}$ on processor 4.

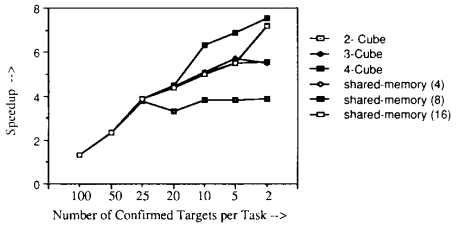


Fig. 9. Speedup versus task granularity on confirmed target path.

In this case, the mapping is such that processor 1 is dedicated to executing the tasks on the confirmed target path. Indeed, the completion time of the heuristic algorithm is equal to the level of the start task, i.e., the cumulative execution time of the tasks on the confirmed target path. Since the level of the start task is a lower bound on the completion time of any mapping algorithm, the mapping is optimal [13, 20]. In addition, since the nodes of the prediction, gating, and updating steps can be decomposed into parallel subtasks, we can vary the granularity of a task (i.e., the number of targets per task) to decrease the level of the start task and obtain a better speedup. In order to understand the best task size for prediction, gating, and track updating on the confirmed path, we varied the number of confirmed targets per task from 2 to 100, and computed the speedup for the resulting task graphs. Specifically, if the number of confirmed targets per task is t, the confirmed target path of the task graph is split into $(100/t)$ parallel paths that join at the clustering and global hypothesis formation stage. For 50 targets per task, we find that the speedup increases to 2.37. When 25 targets are defined as a task, the speedup is 3.76. If we continue partitioning the task size into smaller sizes (i.e., decrease the number of targets per task), we find that the speedup is limited to approximately 3.87 for 2 targets per task. The results are summarized in Fig. 9.

Since the efficiency of the 4 processor architecture is 0.96, it appears that the speedup may be limited by the architecture. In order to verify this conjecture, we consider 3-cube, and 4-cube hypercube architectures. For these architectures, the speedup curve levels off at around 5.53 and 7.55, respectively. When we replace the hypercube architecture with a shared memory structure, the maximum speedups for 4, 8, and 16 processors were 3.87, 5.53, and 7.16, respectively. As can be seen from Fig. 9, the speedup of the tracking algorithm on 2-cube and 4-processor shared memory architectures are virtually identical. However, for a larger number of processors, the hypercube architecture is uniformly better than the shared memory architecture. In addition, when a task on the confirmed path contains 100 targets, all the architectures provide the same speedup. Since the maximum speedup is limited by the critical path (in this case, the confirmed target path), the selection of a computer architecture is not as important as the decomposition of a task. In order to get a better speedup, tasks on the critical path should be decomposed. The speedup levels off when the computational requirement of each task is approximately equivalent to 2 confirmed target tracks.

B. Experiment 2: Effects of Task Granularity on Noncritical Paths

In this experiment, we decompose the track prediction, gating, and track updating on all paths, i.e., confirmed, intermediate, tentative, and born paths, to investigate the corresponding speedup. Experimental results (not shown) reveal that the decompositions on the intermediate, tentative and born paths have little or no effect on speedup as compared with the results of experiment 1 (less than 4 percent improvement). This is because the completion time of the multitarget tracking algorithm is dominated mainly by the critical path (confirmed target path). However, if we decompose a task to a very small size such that the confirmed path is no longer the critical path, then further decomposition of other paths becomes important. In the next experiment, we decompose all parallelizable tasks (including global hypotheses formation) and obtain an almost linear speedup with the number of processors.

C. Experiment 3: Achievable Speedup for Shared Memory and Hypercube Architectures

The final experiment involves the partitioning of all parallelizable tasks (including global hypotheses formation, but excluding clustering).[1] For the 2-cube architecture, the speedup is 3.99 when the task granularity is selected as 2 targets per task in the confirmed path, 2 targets per task in the intermediate path, 3 targets per task in the tentative path, 4 targets per task in the born path, and 25 parallel tasks for global hypotheses formation. For the 3-cube architecture, the achievable speedup is 7.95. In both cases, the actual speedups are equal to the upper bound on the speedup derived in Appendix B. For the 4-cube architecture, the achievable speedup is 15.86, which is comparable to the upper bound on speedup of 15.90.

Next, we consider a shared memory multiprocessor and repeat the same experiment. For 4-processor shared memory architecture, the achievable speedup is 3.99. For 8-processor architecture, the achievable speedup is 7.66. For 16- and 32-processor architectures, the achievable speedups are 13.23 and 16.05, respectively. Fig. 10 shows the maximum achievable speedup and the actual achievable speedup for

[1]Parallelization of clustering function of multitracker on multiprocessor architectures is still a research issue. For recent results in this area see [23].

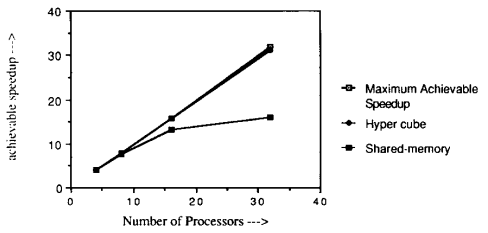


Fig. 10. Achievable speedup for various architectures.

hypercube and shared memory architectures. From Fig. 10, it is seen that the actual achievable speedup for hypercube multiprocessor is almost linear with respect to the number of processors, and is almost identical to the maximum achievable speedup. In addition, the speedup of multitracker on a hypercube architecture is uniformly better than that on a shared-memory architecture for $M \leq 32$. This is because the communication overhead of the shared memory system is higher than that of the hypercube architecture.

VI. CONCLUSIONS AND FUTURE WORK

We considered the problem of mapping a multitarget tracking algorithm onto parallel processor architecture. We evaluated and compared the maximum achievable and the actual speedups for various architectures. We find that the key determinants of speedup are the task granularity and the processor architecture. To increase the speedup, decomposition of tasks should start from those tasks lying on the critical path, since the level of the task graph is a lower bound on the completion time. The results show that when parallelizable tasks are properly partitioned, the achievable speedup of the multitarget tracking algorithm on a hypercube architecture is very close to the maximum achievable speedup, and is much better than that on a shared memory architecture. Furthermore, since the formation of global hypotheses has large computational requirements, decomposition of this task has substantial effect on speedup.

Future research work involves: 1) quasi-static task allocation, 2) dynamic task allocation, 3) automatic task partitioning, and 4) implementation of the multitarget tracking algorithm on the N-cube hypercube and the Sequent shared-memory multiprocessors.

APPENDIX A

Algorithm 1: Computation of $B^q_{i[j]}$: Given the completion times of parent tasks of a task i, $C_{i[j]}$, $1 \leq j \leq |\beta_i|$ arranged in a nondecreasing order and the link available times A_{pq}, the following algorithm computes $B^q_{i[j]}$.

For $j = 1$ to $|\beta_i|$ do

$$k_0 := \{p : i[j] \in T_p\}.$$

If $k_0 \neq q$ then

Find the shortest path $(k_0, k_1, \ldots, k_n, q)$ from processor k_0 to processor q

$$A_{k_0 k_1} := \max\{A_{k_0 k_1}, C_{i[j]}\} + \frac{\nu_{i[j]i}}{c_{k_0 k_1}}$$

$$k_{n+1} := q$$

For $m = 1$ to n do

$$A_{k_m k_{m+1}} := \max(A_{k_{m-1} k_m}, A_{k_m k_{m+1}}) + \frac{\nu_{m[j]m}}{C_{k_m k_{m+1}}}$$

end do

$$B^q_{i[j]} := A_{k_n k_{n+1}}$$

else

$B^q_{i[j]} := C_{i[j]}$[no delay time if in same node]

end if
end do.

Algorithm 2: The Heuristic Mapping Algorithm: Given a directed task graph $G_t = (V_t, E_t)$ with parameters (s_i, v_{ij}) and the processor graph $G_p = (V_p, E_p)$ with parameters (μ_p, c_{pq}), the heuristic algorithm computes the mapping $(T_1, T_2, \ldots, T_q, \ldots, T_M)$. Let α_i represent the set of immediate successors of task i and β_i represent the set of immediate predecessors of task i.

Step 1: Determine the level of each task

$$Z = \{L\}$$

Repeat until $Z = \phi$.

Select a task i of z such that no successors of task i appear in Z

Compute the level of task i, l_i via $l_i = \max_{j \in \alpha_i}(l_j) + s_i/\mu_f$

$Z = Z - \{i\} \cup \beta_i$ [add immediate predecessors of task i to set Z]

end

Step 2: Construct a priority list [1] [2]...[L] by sorting l_i in nonincreasing order.

Break ties on the basis of nonincreasing number of successors.

Step 3: For $j = 1$ to L do

$i = [j]$, [[j] = jth task in the priority list].

Find assignments q^* via (11)

$$T_{q^*} = T_{q^*} \cup i$$

Update the link available times as appropriate

end do.

APPENDIX B. COMPUTATION OF TASK PARAMETERS FOR MULTITARGET TRACKING ALGORITHM

We assume the following values for arithmetic operations:

time for 32 bit multiplication	4 μs = $4*10^{-6}$ s
time for 32 bit addition	2.6 μs = $2.6*10^{-6}$ s
time for 16 bit comparison	1.3 μs = $1.3*10^{-6}$ s.

Typical values for scenario parameters (see Table III) are

$$n = 4$$
$$m = 3$$
$$N_c = 100$$
$$B_c = 6$$
$$R_c = 1.5$$
$$N_i = 20$$
$$B_i = 3$$
$$R_i = 1$$
$$N_t = 30$$
$$B_t = 3$$
$$R_t = 1$$
$$N_b = 100$$
$$B_b = 1$$
$$R_b = 1.0$$
$$N_r = 100$$
$$NTS = 2$$
$$NS = 50$$
$$NTC = 4$$
$$NC = 25$$
$$NGH = 100$$
$$NSCAN = 4.$$

Using the computational requirements for various steps defined in Section IV, the task execution times may be evaluated as shown in Table V. The associated data to be transferred may be evaluated as shown in Table VI.

TABLE V
Task Execution Times

Task	Execution Time (in micro seconds)	Execution Time (in seconds)
predict confirmed tracks	$714N_cB_c$	0.428400
predict intermediate tracks	$714N_iB_i$	0.042840
predict tentative tracks	$714N_tB_t$	0.064260
predict born tracks	$714N_bB_b$	0.071400
gate confirmed tracks	$3196.2N_cB_c$	1.917720
gate intermediate tracks	$3196.2N_iB_i$	0.191772
gate tentative tracks	$3196.2N_tB_t$	0.287658
gate born tracks	$3196.2N_bB_b$	0.319620
update confirmed tracks	$825N_cB_cR_c$	0.742500
update intermediate tracks	$825N_iB_iR_i$	0.049500
update tentative tracks	$825N_tB_tR_t$	0.074250
update born tracks	$825N_bB_bR_b$	0.082500
clustering	$0.65NTS.NS(NS-1)$	0.003185
global hypotheses formation	$NC*16380$	0.409500

TABLE VI
Data Communication Requirements

Task	Value (in bits)	Value (in Kilobits)
predict confirmed targets	$712N_cB_c$	429.60
predict intermediate targets	$712N_iB_i$	42.96
predict tentative targets	$712N_tB_t$	64.44
predict born targets	$712N_bB_b$	71.60
gate confirmed targets	$5288N_cB_c$	3175.20
gate intermediate targets	$5288N_iB_i$	317.52
gate tentative targets	$5288N_cB_c$	476.28
gate born targets	$5288N_bB_b$	529.20
update confirmed targets	$552N_cB_cR_c$	500.40
update intermediate targets	$552N_iB_iR_i$	33.36
update tentative targets	$552N_tB_tR_t$	50.04
update born targets	$552N_bB_bR_b$	55.60
clustering		52.80
global hypotheses formation	$NC*2976$	74.40

APPENDIX C. MAXIMUM ACHIEVABLE SPEEDUP

As we have pointed out in Section IV, the processing of tentative and born targets are not dependent on the processing of confirmed targets, and hence, can be processed in parallel with confirmed targets. In order to take advantage of this parallelism, we have grouped the computations in each scan into the following five task groups (see Figs. 7 and 8).

Task group a. Prediction, and gating of confirmed targets. (corresponds to Tasks 2 and 3 of Fig. 8)

Task group b. Update of confirmed targets. (corresponds to Task 4 of Fig. 8)

Task group c. Clustering of confirmed targets. (corresponds to Task 5 of Fig. 8)

Task group d. Formation of global hypotheses. (corresponds to Task 6 of Fig. 8)

Task group e. Prediction, gating, and update of born, tentative, and intermediate targets. (corresponds to Tasks 8–20 of Fig. 8)

All the above task groups (with the exception of task group c) involve the processing of multiple subtasks, wherein each of the subtasks may be executed in parallel. The number of subtasks and the

TABLE VII
Computational Requirements

Task group	Number of Parallel Subtasks	Time to Perform Each Subtask (in seconds)	Total Task group Time (in seconds)
a	$N_c B_c = 600$	0.003910	2.346000
b	$N_c B_c R_c = 900$	0.000825	0.742500
c	1	0.003185	0.003185
d	$N_c = 25$	0.016380	0.409500
e	$N_i B_i R_i + N_t B_t R_t + N_b B_b R_b = 250$	0.004735	1.183750

computational requirements of the five task groups may be evaluated as shown in Table VII.

If multitracker were to be run on a single processor architecture, the completion time of the algorithm would be 4.684935 s. If multitracker were to be run on a four-processor MIMD architecture, the subtasks may be scheduled as follows to achieve maximal speedup (neglecting communication delays).

1. Subtasks in Task group a are processed in parallel on the four processors. The 600 subtasks will require 150 passes resulting in a net execution time of 0.5865 s.
2. Similarly, subtasks in Task group b are processed in parallel resulting in a net execution time of 0.185625 s.
3. The only subtask in Task group c is processed on one of the processors. At the same time, three subtasks from Task group e (i.e., three tracks from born or tentative targets) are processed on the remaining three processors resulting in a net execution time of .004735 s.
4. Subtasks in Task group d are processed in parallel on the four processors. The 25 subtasks will require seven passes resulting in a net execution time of 0.114660 s. On the seventh pass, Task group d will require only one processor and the remaining three processors are used to process nine subtasks of Task group e.
5. The remaining 238 subtasks in Task group e are processed in parallel on the four processors. The 238 subtasks will require 60 passes resulting in a net execution time of 0.2841 s.

Thus, the completion time on the 4-processor architecture (neglecting communication delays) will be 1.17562 s, resulting in a maximum achievable speedup of 3.99. Similarly, the maximum achievable speedups for the 8, 16, and 32 processor architectures can be evaluated to be 7.95, 15.90, and 31.71, respectively.

ACKNOWLEDGMENT

We are grateful to Mahesh Dontamsetty for his help in the preparation of this paper.

REFERENCES

[1] Bar-Shalom, Y., and Fortmann, T. E. (1988)
Tracking and Data Association.
New York: Academic Press, 1988.

[2] Bar-Shalom, Y. (1978)
Tracking methods in a multiobject environment.
IEEE Transactions on Automatic Control, **AC-23** (Aug. 1978), 618–626.

[3] Batcher, K. E. (1982)
Bit-serial processing systems.
IEEE Transactions on Computers, **C-31**, 5 (May 1982), 377–384.

[4] Bergland, G. D., and Hunnicutt, C. F. (1972)
Application of a highly parallel processor to radar data processing.
IEEE Transactions on Aerospace and Electronic Systems, **AES-8**, 2 (Mar. 1972), 161–162.

[5] Bokhari, S. H. (1981)
On the mapping problem.
IEEE Transactions on Computers, **C-30** (Mar. 1981), 207–214.

[6] Hu, T. C. (1961)
Parallel sequencing and assembly line problem.
Operations Research, **9** (Nov. 1961), 841–848.

[7] Kasahara, H., and Narita, S. (1984)
Practical multiprocessor scheduling algorithms for efficient parallel processing.
IEEE Transactions on Computers, **C-33**, 11 (Nov. 19184), 1023–1029.

[8] Kurien, T., and Washburn, R. B. (1985)
Multiobject tracking using passive sensors.
In *Proceedings of the 1985 American Control Conference*, Boston, MA, June 1985, 1032–1038.

[9] Kurien, T., Allen, T. G., and Washburn, R. B. (1986)
Parallelism in multi-target tracking and adaptation to multi-processor architectures.
In *Proceedings of the 9th MIT/ONR Workshop on C^3 Systems*, Dec. 1986, 45–51.

[10] Lee, R. T., Pattipati, K. R., and Luh, P. B. (1989)
On the asymptotic optimality of a heuristic mapping algorithm.
In *Proceedings of the 28th IEEE Conference on Decision and Control*, Tampa, FL, Dec. 1989, 853–859.

[11] Pattipati, K. R., Lee, R. T., Shah, S. A., and Luh, P. B. (1988)
A decision support system for the algorithm-architecture mapping problem.
In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Beijing, China, Aug. 1988, 223–226.

[12] Pattipati, K. R., and Sandell, N. R., Jr. (1983)
A unified view of multi-object tracking.
In *Proceedings of the American Control Conference*, San Francisco, CA, June 1983, 458–463.

[13] Pattipati, K. R., Luh, P. B., Lee, R. T., Shah, S. A., and Deb, S. (1989)
BM/C^3 algorithm mapping onto concurrent processors.
Technical report 88-147, RADC, Mar. 1989.

[14] Pearl, J. (1984)
Heuristics.
New York: Addison-Wesley, 1984.

[15] Reid, D. B. (1979)
An algorithm for tracking multiple targets.
IEEE Transactions on Automatic Control, **AC-24**, 6 (Dec. 1979), 843–854.

[16] Sethi, R. (1975)
Scheduling graphs on two processors.
SIAM Journal of Computing, **5** (1975), 73–82.

[17] Summers, M. W., and Trad, D. F. (1974)
The evolution of a parallel active tracking program.
In *Proceedings of the 1974 Sagamore Computer Conference*, August, 1974, 377–384.

[18] Garey, M. R., and Johnson, D. S. (1979)
Computers and Intractability: A Guide to the Theory of NP-Completeness.
San Francisco: W. H. Freeman & Company, 1979.

[19] Kurien, T. (1989)
Issues in the design of practical multitarget tracking algorithms.
In Y. Bar-Shalom, Ed., *Multitarget-Multisensor Tracking: Applications and Advances.*
Norwood, MA: Artech House, 1989.

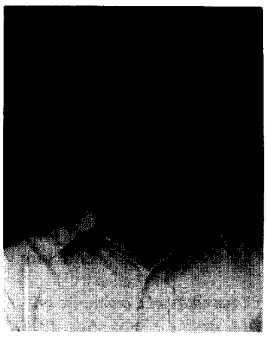
[20] Lee, R. T. (1988)
Fault-tolerant algorithm mapping onto parallel computing architectures.
M.S. thesis, Dept. of Electrical and Systems Engineering, Univ. of Connecticut, Storrs, 1988.

[21] Hwang, K., and Briggs, F. A. (1984)
Computer-Architecture and Parallel Processing.
New York: McGraw-Hill, 1984.

[22] Bertsekas, D. P., and Tsitsiklis, J. (1989)
Parallel and Distributed Computation.
Englewood Cliffs, NJ: Prentice Hall, 1989.

[23] Allen, T., Cybenko, G., Angelli, C. M., Polito, J. (1987)
Hypercube implementation of tracking algorithm.
In *Proceedings of the 1987 Command and Control Research Symposium*, National Defense University, Washington, DC, July 1987.

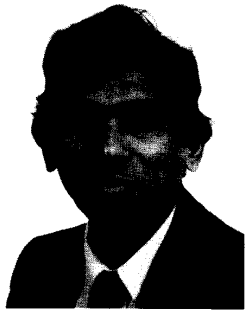
[24] Blackman, S. S. (1986)
Multi-target Tracking with Radar Applications.
Dedham, MA: Artech House, 1986.



Krishna R. Pattipati (M'79) received the B.Tech degree in electrical engineering with highest honors from the Indian Institute of Technology, Kharagpur, in 1975, and the M.S. and Ph.D. degrees in systems engineering from the University of Connecticut, Storrs, in 1977 and 1980, respectively.

He was employed by Alphatech, Inc., Burlington, MA from 1980 to 1986, where he supervised and performed research on human decision modeling, multitarget tracking, queuing networks, automated testing, and large-scale mixed-integer optimization. Since September 1986, he has been an Associate Professor in the Department of Electrical and Systems Engineering. He has also served as a consultant to Alphatech, Inc. and the I.B.M. Thomas J. Watson Research Center.

Dr. Pattipati was selected by the IEEE Systems, Man, and Cybernetics Society as the Outstanding Young Engineer of 1984, and received the Centennial Key to the Future award. He was a member of the administrative committee of the IEEE Systems, Man, and Cybernetics Society during 1986–1988. Dr. Pattipati served as the finance chairman of the International Conference on Control and Applications, Jerusalem, Israel, 1989 and as the Vice-Chairman for invited sessions of the IEEE International Conference on Systems, Man, and Cybernetics Conference, Boston, MA, 1989. He is listed in *Who's Who in the Frontiers of Science and Technology.*



Thomas Kurien (M'83) received his B.S. degree in electrical engineering from Osmania University, India, in 1973, and his M.S. and Ph.D. degrees in electrical engineering from the University of Connecticut, Storrs, in 1975 and 1983, respectively.

From 1979 to 1981, he worked as a navigation analyst for Intermetrics, Inc. Since 1981, he has been a member of the technical staff at ALPHATECH Inc. His research interests are in the application of statistical estimation and expert systems techniques to surveillance problems. These include the design of multisensor multitarget tracking, identification, and threat evaluation algorithms, the implementation of these algorithms on multiprocessor computer architectures, and the interface of these algorithms with human operators. He has published several papers in this area and is a coauthor of two books, *Science of Command and Control: Part II*, Eds. S. E. Johnson and A. H. Levis, and *Multitarget-Multisensor Tracking: Applications and Advances*, Ed. Y. Bar-Shalom.

Rong-Tay Lee received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China, in 1983, and the M.S. degree in electrical and systems engineering from the University of Connecticut, Storrs, in 1988.

Since graduation, he has been with Valid Logic Systems in the CAE division where he started as a software engineer working on logic and fault simulations. His interests include fault simulation, logic synthesis, testing, and distributed systems.

Mr. Lee is a member of IEEE.

Peter B. Luh received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China in 1973, the M.S. degree in aeronautics and astronautics engineering from M.I.T., Cambridge, MA in 1977, and Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA, in 1980.

From 1980 he has been with the University of Connecticut, Storrs, and currently is an Associate Professor in the Department of Electrical and Systems Engineering. His major research interests include hierarchical planning and control of large scale systems, schedule generation and reconfiguration for manufacturing systems, distributed decisionmaking, game theory, and multitarget tracking. He has been a principal investigator and consultant to many industry and government funded projects in the above areas.

He has published about 80 papers. He is an Associate Editor of the *IEEE Transactions on Automatic Control*, won the Best Paper Award of the 1987 Joint Command and Control Research Symposium, and has served on Program Committees and Operating Committees of several major national, international and intersociety conferences. He is a member of IEEE, Sigma Xi, and listed in *Who's Who in Engineering* and *Who's Who in the East*.