A new approach to distributed data fusion

de Waard, H.W.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 6

Parallel processing model

For complex scenarios, where a large number of targets in dense clutter have to be tracked, there is a significant possibility that even fast heuristics will not solve the general multidimensional assignment (MDA) problem according to the realtime requirement introduced in section 1.5. A promising way to accelerate MDA applications is to partition the application in a number of processes, which are mapped onto an available processor network. Cluster-based methods for computation time reduction have been introduced in chapter 4. Historically, accelerating an MDA application using clusters was not considered possible for loosely coupled multiprocessors. In this chapter a new partition approach, based on the concept of distributed partial clusters, is introduced which solves the problem of using clusters to accelerate the solution process for a MDA problem and which can also handle very large cluster-related tasks which consume an excessive amount of processing power.

6.1 Introduction

The data association problem for a number of data sets \( \geq 3 \), coming from single or multiple sensors is NP-hard ([51], [112]). A data set from a sensor corresponds with the data collected during a full scan. The set of established track hypotheses outside the sliding window is treated as measurements and provides an extra data set to the sliding window (see section 3.1). Normally, the sliding window has a size \( w_s \geq 3 \), which means that the MDA problem defined by the number of data sets within the window is already NP-hard. Due to appropriate complexity reduction measures, the number of track hypotheses stabilizes after some time (section 3.10, chapter 3).

In section 6.2 it is shown that even the use of fast heuristics will not guarantee realtime behavior when large numbers of targets have to be tracked. A promising approach to accelerating data association applications is to partition the application in a number of processes, which are mapped onto an available processor network. An overview of different partition approaches, found in literature, is given in section 6.3. In section 6.4 the problem of an optimal mapping of tasks determined for a MDA application onto a processor network is dis-
CHAPTER 6. PARALLEL PROCESSING MODEL

cussed. This problem is also NP-hard [66]. Section 6.5 discusses the scheduling of tasks onto a loosely coupled multiprocessor and the estimation of cost functions for cluster-related tasks, which instantiate the considered MDA problem, and the strong component related tasks, which determine one or more solutions for the considered MDA subproblem. In section 6.6, a number of important practical problems is discussed related with the scheduling of very large clusters onto a loosely coupled multiprocessor and the corresponding distribution of large amounts of cluster track hypotheses using a communication network with a limited bandwidth. To solve both problems, the concept of partial clusters is introduced in section 6.6.3.

6.2 Limitations of fast heuristics

The process to solve a MDA problem is divided in a problem instantiation phase, a problem decomposition phase and the actual solution phase. During the problem instantiation phase, all relevant track hypotheses have to be created to determine the decision variables and the track formation costs (chapter 3). Assuming that the number of track hypotheses stabilizes after processing data set \( w_m \), the worst case upper bound for the total number of established track hypotheses is given by \( T \approx N^{w_m} \), where \( N \) is the maximum number of measurements in any one of the data sets (eq. 3.21, chapter 3). When the complexity of the considered scenarios increases, the upper bound also increases. Normally, it is sufficient that heuristic algorithms are used to determine one or more good solutions for the MDA problem. Using a typical fast heuristic, it will be shown that it is still possible that the realtime requirement is not met for solving the MDA problem (chapter 1).

For the example the low order polynomial algorithm SGTS, developed by Capponi [32], is chosen which is very fast. The upper bound to the number of operations to sort a list of \( T \) track hypotheses is \( \frac{T^2}{2} + \frac{3T}{2} - 1 \) [4], which means that the complexity of the algorithm is \( O(T^2) \). Assuming that the size of the set of track hypotheses has stabilized to \( T \) track hypotheses and that a new data set has been added to the sliding window, containing maximally \( N \) measurements, the worst case upper bound for the number of track hypotheses is given by \( T \times N \). Substituting this number, the final complexity for the SGTS algorithm is \( O(N^2 \times (w_m + 1)) \) which is clearly exponential.

SGTS is a very fast algorithm. The exponential complexity of the algorithm is due to the exponential upper bound for the number of track hypotheses in the set of track hypotheses, which has to be generated during the problem instantiation phase. Due to the fact that the set of hypotheses is input to all (heuristic) algorithms which solve the MDA problem, no guarantee can be given that an algorithm will meet the realtime requirement for complex scenarios. Even after applying the computation-time reduction methods proposed in chapter 4 and 5, there still will be scenarios which require additional methods to reduce computation time to meet the realtime requirement.

\footnote{In [32] the average performance of Quicksort has been used instead of the worst case performance [4].}
6.3 Literature overview

A promising way to accelerate data association applications is to partition the application in a number of processes which are mapped onto an available processor network. The objective of the mapping is to minimize the risk that stated realtime requirements are not fulfilled. A complicating factor is that the mapping problem is NP-hard [66]. A limited number of partitioning approaches can be identified in literature. In this section an overview of the most important approaches is given.

6.3.1 Distribution of hypotheses

One approach to parallelize multiple hypothesis tracking (MHT) is to exploit the fact that creation, evaluation and extension of individual hypotheses are completely independent of each other [125]. This means, theoretically, that multiple processors can start with a set of hypotheses and can do all the computations on their local hypotheses (and the hypotheses’ subsequent offspring hypotheses) without the need for any communication or synchronization during the processing of a data set. An implementation was tested on several loosely coupled computer architectures [125]. For all architectures the same problem was encountered: often one of the processors ran into processing problems because one of the hypotheses trees grew much faster than average. The result was that the specific processor had to carry out many more correlations than the other processors in the network. Load balancing by sending hypotheses to other nodes which were lightly loaded was not successful, because it was impossible to predict accurately what the number of offspring hypotheses would be for the different involved hypotheses. Since only 10 frames of data were processed, there was not enough information to make accurate statistical estimates. Furthermore, it was also not analyzed whether the problems were due to a large number of targets which where operating close together, or whether a large number of measurements was due to clutter. No follow-up paper has been published, suggesting that no solutions were found to remedy the reported problems.

Also, once clusters are formed, hypothesis formation and pruning can be performed in parallel for all clusters [25]. To facilitate that each processor has at any given time efficient access to all track data such as Kalman filter and track score information, a parallel computer with globally shared memory is required. The globally shared memory allows convenient access to cluster information so that, upon cluster merging, all relevant cluster information can be transferred without any significant communication delay from one processor to another.

6.3.2 Observation space partitioning

Another possible approach is to divide the observation space in segments and to use a separate processor to process the data in a segment [25]. Each processor solves the data association problem corresponding with the segment and produces a corresponding air picture. To guarantee that continuous track hypotheses can be maintained on targets that cross segment boundaries, the different segments must partially overlap with adjacent segments. In cases where adjacent segments produce a track hypothesis for the same object, the track hypotheses
are fused using the track-to-track association method proposed by Bar-Shalom [11]. Using the air pictures of the different segments, an overall air picture is created. No implementations or tests have been reported in literature.

An alternative heuristic approach is to partition the observation space, based on the number of measurements to be processed [118]. The objective is to minimize computation time by distribution of the processing load on the available computer architecture with the requirement that the resulting communication load does not exceed pre-set communication load thresholds. The approach has not been implemented or tested.

6.3.3 Functional partitioning

Track prediction, gating and track update are independent from track hypothesis to track hypothesis. Pattipati ([93],[92]) and Kurien [78] have used this fact to map a multitarget tracking algorithm onto a number of parallel processor architectures. They have done a number of parametric analyzes to determine the theoretical acceleration for various computer architectures and found that the key determinants of acceleration are the number of track hypotheses each task has to handle and the processor architecture. The theoretical results showed that when the possibility to carry out tasks in parallel is properly used, the theoretical achievable acceleration on a hypercube architecture is very close to the maximum achievable acceleration, and is much better than the theoretically maximum achievable acceleration on a shared memory architecture. The proposed future research work would involve the implementation of the multitarget tracking algorithm on both the N−cube hypercube and the Sequent shared-memory multiprocessors.

Tower [122] describes the hardware architecture of the systolic cellular array processor (SCAP) and discussed the mapping of a multiple hypothesis tracking algorithm onto the architecture with the objective to accelerate the gating of tracks, the clustering of tracks and the formation of clusters. The SCAP is an extremely high speed simultaneous instruction, multiple data (SIMD) processor, containing 64 equal processing elements which were interconnected in a rectangular 2−D grid. For example, to correlate measurements with track hypotheses, 8 identical columns with track information are created with different track hypotheses in each row. Different measurements sets are floating down each column, which means that it was possible to carry out 64 correlations at the same time. No follow-up paper has been published, suggesting that the MHT algorithm has not been implemented onto the SCAP.

Popp [106] compared a coarse grained dynamic parallel implementation with a fine-grained parallel implementation of a 2−D assignment algorithm, which used an interactive multiple model estimator with a number of filter models. For the different experiments real sensor data was used, measured by two L-band sensors situated at different locations. Using a standard 2−D assignment algorithm and varying the number of filters in the IMM model, Popp obtained the workload distribution given in percentages in table 6.1 for a sequential implementation. Using the fact that most of the workload was dedicated to the calculation of the numerous correlations, in the coarse grained parallel implementation multithreading was used to calculate the correlations. For each track hypothesis a work thread was created, which determined the likely candidate associations. In the fine grained implementation, the numerically expensive
computations, like for example coordinate transformations, state vector and covariance matrix estimates, linear algebraic operations such as matrix multiplication, Cholesky factorization and inner products, were parallelized for each IMM-module and mapped to worker threads. The computing environment consisted of several shared memory MIMD multiprocessor systems: a 2–processor and 4–processor SPARCstation 20, and a 12–processor SPARCcenter 2000. For scheduling of the application threads, an available SunOS 5.4.2 Unix threads scheduler was used. The final result was that the coarse grained parallel implementation was much more efficient than the fine grained parallel implementation. Using the 12–processor SPARCcenter 2000 multiprocessor, for the coarse grained parallel implementation the acceleration in processor time, with respect to the sequential implementation, varied between 5 (2 models) and 12 (12 models) as function of the number of filter models used in the IMM-model.

In [103] a number of $k$–best 2–D assignment algorithms is compared. To mitigate complexity issues, also parallel versions of the different assignment algorithms have been made, which were implemented on a 4-processor SPARCstation 20. Using multithreading, a coarse-grained shared-memory parallelization was made. Each created worker thread processed a specified number of candidate associations across mutually exclusive tracks and measurement data. The processing of candidate associations primarily consisted of computing numerous independent gating tests, state estimates, covariance calculations, and likelihood function evaluations to determine cost coefficients. Dynamic scheduling was used to schedule threads. Furthermore, also the $k$–best 2–D assignment problem was partitioned into a number of independent subproblems. For each of the subproblems the optimal solution was determined. A number of worker threads was created, each of which processed a number of assignment problems, which were also scheduled using dynamic scheduling. The $k$-best 2–D assignment algorithm configured with the auction algorithm outperformed the $k$-best 2–D assignment algorithm configured with Jonker-Volgenant–Castanon (JVC) algorithm in terms of execution time by a factor of $\approx 1.5$ using 2 processors and a factor of $\approx 3$ using 4 processors using a sparse scenario. It must be noted that the JVC is especially suited for dense scenarios and the auction algorithm is especially suited for sparse scenarios.

The commonly proposed approach to parallel processing for MHT is based on functional partitioning. This approach utilizes the fact that track prediction, gating and track update are independent from track hypothesis to track hypothesis ([78], [92], [5], [122]). This means that the corresponding tasks can be performed in parallel.

### Table 6.1: Workload distribution as function of the number of IMM filters.

<table>
<thead>
<tr>
<th>Activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain measurements</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>Compute correlations</td>
<td>94.3</td>
<td>94.7</td>
<td>95.2</td>
<td>96.0</td>
</tr>
<tr>
<td>Solve 2-D problem</td>
<td>1.1</td>
<td>0.8</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>Track formation/updates</td>
<td>3.8</td>
<td>3.9</td>
<td>3.7</td>
<td>3.5</td>
</tr>
</tbody>
</table>
CHAPTER 6. PARALLEL PROCESSING MODEL

6.4 The optimal mapping problem

In this section the problem is discussed of finding an optimal mapping of the tasks determined for a MDA application onto a processor network. In section 6.4.1 a general approach is introduced to develop a directed acyclic graph for an algorithm, which is used to map the identified tasks onto a processor network, represented by an undirected graph. Next, in section 6.4.2 a directed acyclic task graph for an MDA application is developed. Finally, in section 6.4.3 the optimization problem corresponding with the optimal mapping problem is formulated.

6.4.1 Mapping tasks onto a processor network

The data flow and the data dependencies of an algorithm can be analyzed by representing the algorithm in the form of a directed, acyclic graph [93]. 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. A cycle is a closed path in a graph, where each node and arc is different, except the start and end node [20, 131]. A graph containing no cycles is called acyclic. The mapping problem is to assign the nodes of a task graph onto the nodes of a processor graph in such a way that the mapping with the earliest completion time is obtained. It is assumed that in each node of the processor graph processing and communication are carried out in parallel. An example is shown in Fig. 6.1. The task graph is a directed, acyclic graph,

Figure 6.1: The problem is to map a task graph onto a processor network

\[ G_t = (V_t, E_t) \], where \( V_t \) is the set of nodes denoting the tasks which have to be carried out, and \( E_t \) represents the set of directed arcs. Each arc between two nodes represents the required communication between the two nodes. The direction of an arc between two nodes indicates the direction in which the data flows, and thus implies a precedence relation on the processing in those nodes. For example, the arc directed from node \( i \) to node \( j \) indicates the processing in node \( i \) must proceed that of node \( j \). Each graph includes a start node.
6.4. THE OPTIMAL MAPPING PROBLEM

and a terminal node. The number of tasks in the graph is \( n = |V_t| \) and the number of nodes in the processor network is given by \( p \). The processor graph is an undirected graph, \( G_p = (V_p, E_p) \), where \( V_p \) is the set vertices denoting processors, and \( E_p \) is the set of undirected arcs depicting the communication links among processors. Each node of the processor graph is labelled by the service rate of the processor in terms of millions of instructions per second and each arc by the link capacity, measured in bits per second.

It is assumed that the original data association problem is partitioned into a number of tasks, which have to be mapped on the available processor network. An example is shown in Fig. 6.2. Four tasks have to be scheduled on the available processor network containing the processor nodes \( P_1, \cdots, P_4 \). If the four tasks are scheduled on node \( P_1 \), the four problems are solved at clock time \( T_2 \). Scheduling the four problems on four different nodes possibly has the advantage that all problems are solved at clock time \( T_1 \). Due to the fact that three tasks have to be relocated from node \( P_1 \) to the other three nodes, the time necessary to transfer the relevant task data between processors has to be accounted for. From this example the conclusion can be drawn, that significant communication times already occur because a large amount of track hypotheses, decision variables and track formation cost data are transmitted through the communication network. Consider the slightly changed example in Fig. 6.3. In this case relocation of the fourth task to node \( P_4 \) is a bad idea, because a very large amount of data has to be transmitted from node \( P_1 \) to node \( P_4 \). Clearly not all tasks are finished at time \( T_2 \).

6.4.2 MDA task graph

Consider the general MDA introduced in chapter 3. It is assumed that the sliding window technique is used to consider only the last \( w_s \) data sets. Furthermore, it is assumed that a number of \( S \) sensors contribute each data sets to the sliding window. A data set from a sensor corresponds with the data collected during a full scan. When a new data set is retrieved and the window has been shifted one position to the right (Fig. 6.4), the different tasks, shown in Fig. 6.5, are executed. Without applying one of the methods proposed in this thesis to reduce the prediction computation time (see section 4.1), each time a new data set is processed all track hypotheses have to be predicted to the time of each measurement in the new data set. Earlier in section 6.3.3 it has been stated...
that track prediction, gating and track update are independent from track hypothesis to track hypothesis. This means that the corresponding tasks can be performed in parallel. This produces the directed, acyclic task graph, given in Fig. 6.6, for the tasks specified in Fig. 6.5. A box around tasks indicates that those tasks sequentially process a set of track hypotheses corresponding with a cluster. When information like a set of track hypotheses is distributed, a standard header with a fixed number of bytes is attached to the information. This means that when the track hypotheses are separately distributed instead of a single set, more bytes of data are necessary which increases the load on the communication links. To fulfil the requirement that a start and end node are included in the graph, a task to retrieve a new data set and a task to update the single integrated air picture (SIAP) (chapter 1) are added to graph.

In this thesis it is assumed that each participant has a loosely coupled multiprocessor available to process the received information where each processor has its own local memory (Fig. 6.7)[44]. Information exchange between processors is carried out by sending messages through the communication network. Communication delays may occur when a large number of messages has to be exchanged.
6.4. THE OPTIMAL MAPPING PROBLEM

6.4.3 Optimal mapping

The problem to find an optimal mapping or schedule can be formulated as a nonlinear integer optimization problem, which is known to be NP-hard [66]. The number of \( n \) tasks has to be assigned to \( p \) processor nodes. If task \( i \) is assigned to processor node \( j \), the assignment variable \( x_{ij} = 1 \), otherwise \( x_{ij} = 0 \). The cost to carry out task \( i \) on processor node \( j \) is given by

\[
c_{ij} = T_{ij}^p + T_{i,kj}^c
\]  

(6.1)

where \( T_{ij}^p \) is the cpu-time necessary to perform the task and \( T_{i,kj}^c \) is the time to relocate task \( i \) from processor node \( k \) to processor node \( j \). The set of tasks assigned to processor node \( j \) is given by \( T_j \) and the number of tasks in the set is given by \( n_j = |T_j| \). The total cost to carry out the \( n_j \) assigned tasks on processor node \( j \) is given by

\[
V_j = \sum_{i \in V_j} c_{ij} \times x_{ij}
\]

where \( n_j \) assignment variables \( x_{ij} \) are set to 1. The nonlinear integer optimization problem to determine the
optimal schedule is given by:

\[
\text{Minimize} \quad V(\vec{x}) = \max_{i_1=1}^{p} \sum_{i_0 \in T_{i_1}} c_{i_0,i_1} \times x_{i_0,i_1} \tag{6.2}
\]

Subject to

\[
\sum_{i_1=1}^{p} x_{i_0,i_1} = 1, \quad i_0 = 1, ..., n
\]

\[
x_{i_0,i_1} \in \{0, 1\} \quad \text{for all } i_0, i_1
\]

taking into account the different precedence relations specified by the directed, acyclic task graph which means that it is possible that one or more tasks have to be finished before a certain task can be carried out.

### 6.5 Scheduling of tasks

The obvious approach to finding the optimal solution for eq. 6.2 is to first generate all possible execution lists specifying the order in which the different tasks have to be executed, taking into account the different precedence relations specified in the directed, acyclic task graph. Worst case, the total number of lists generated is \(n!\), where \(n\) is the number of tasks which have to be carried out. For each of the generated lists, the partial enumeration branch-and-bound algorithm ([49, 90]) is used to generate the different task processor mappings. When all relevant mappings have been generated, the mapping with the shortest completion time is chosen and carried out. Worst case the number of different mappings which have to be generated by the branch-and-bound algorithm to determine the mapping with the shortest completion time is given by \(n! \times p^n\).

This means that the complexity to determine the optimal solution for eq. 6.2 is given by

\[
O(n! \times p^n) \tag{6.3}
\]

Given the fact that the complexity of the computation time to determine the optimal mapping for the \(n!\) tasks is strongly non-linear, it is necessary to use a heuristic mapping algorithm for larger \(n\) and \(p\) values. Pattipati [93] has developed a heuristic mapping algorithm, which produces very good results and which provides optimal solutions as the numbers of tasks in each parallel path tend to infinity and the computation times dominate the involved communication times. The heuristic mapping algorithm consists of two stages. The first stage employs the concept of critical path to determine the order of task allocation and creates an allocation list with ordered tasks. A critical path is a
6.5. SCHEDULING OF TASKS

longest path through the directed, acyclic graph, corresponding to the longest
time to perform an ordered sequence of tasks [37]. To determine the order of
task allocation, the level of a task \( i \) is defined as

\[
l_i = \max_k \sum_{j \in \pi_k} \frac{s_j}{\mu_f}
\]

where \( \mu_f \) is the service rate of the fastest processor expressed in operations per
second (e.g. Mflops), \( s_j \) is the service demand of task \( j \) expressed in operations
to be performed (e.g. Mflop), and \( \pi_k \) denotes the \( k^{th} \) path from task \( i \) to the
end task. That is, \( l_i \) is a lower bound for the length of the critical path from
task \( i \) to the end task. The algorithm is based on the premise that tasks with
larger levels should be executed earlier in the sequence to minimize completion
time. If several tasks have the same level, then the task with the greater number
of successors should be allocated first. Thus, the allocation order is ranked in
decreasing level order. Furthermore, tasks with the same level are ranked in
decreasing order of the number of successors. The second stage sequentially
allocates the tasks from the ordered list to processors, so that the completion
time for the different tasks is minimized.

Assume that number of nodes in the task graph is \( L \) and that the number of
processors in the multiprocessor is \( p \). Furthermore, assume that the maximum
number of parents for a task in the task graph is \( n_p \) and that the maximum
number of arcs between two processors in the processor graph is \( n_{arc} \). The
complexity of the heuristic mapping algorithm proposed by Pattipati [93] is
determined by the following contributions:

- Using eq. 6.4, for each task the level is determined. This means that
every arc in the task graph has to be traversed once. The number of arcs
is given by \( |E_t| \). The complexity for this step is given by:

\[
O(|E_t|)
\]  

- The next step is to construct an allocation list by sorting the tasks in
decreasing order of task level, taking into account that in case of equal
level ranking occurs in decreasing number of successors. Here it is as-
sumed that Quicksort is used, which has worst case complexity \( O(L^2) \) [4].
Furthermore, assuming that worst case every task has the same level all
elements have to be sorted in order of decreasing number of successors
which amounts to complexity \( O(L^2) \). Using the earlier introduced rule for
summation [68], the complexity for this step is:

\[
O(L^2) + O(L^2) = O(L^2)
\]  

- The last step is to assign the different tasks in the allocation list to proces-
sors in such a way that the completion time of each task is minimized,
taking into account the time that each predecessor is finished and the pro-
duced information is available. This means checks have to be carried out
to determine that each predecessor task completes in time and that the
produced data is available for the task to be allocated, taking into account
the capacity of the used communication links. Assuming worst case that
each task can be allocated to each processor, that each task has maximally $n_p$ predecessors and that the number of arcs between the relevant processors is at most $n_{arc}$, the upper bound for the number of operations is given by $n_p \times n_{arc} \times p \times L$.

The complexity for this step is given by

$$O(n_p \times n_{arc} \times p \times L) \quad (6.7)$$

Using the rule for summation [68] on the contributions given by eq. 6.5-6.7, the complexity for the heuristic mapping algorithm is

$$O(|E_t| + L^2 + n_p \times n_{arc} \times p \times L) \quad (6.8)$$

To use the heuristic mapping algorithm, it is necessary to obtain estimates for the cost $c_{ij}$ to carry out task $i$ on processor $j$. The cost is defined by $c_{ij} = T_{pj}^i + T_{kj}^i$, where $T_{pj}^i$ is the cpu-time necessary to carry out the task on processor $j$ and $T_{kj}^i$ is the time necessary to relocate task $i$ from processor $k$ to processor $j$. It is assumed that the task sequencing at each processor is non-preemptive: once processing begins on a task, it is completed without interruption. The execution of task $i$ requires the service $s_i$, expressed in the number of operations to be performed, and processor $j$ provides the service rate $\mu_j$, expressed in operations per second. An arc between processor $j$ and $k$ in the undirected processor graph (Fig. 6.1) is labelled by the link capacity $d_{jk}$. For 10-Gigabit Ethernet, $d_{jk} = 10$ Gbps [8].

In Fig. 6.6 the proposed parallel model for the general multi-dimensional assignment (MDA) problem is shown. Earlier it has been stated that when track hypotheses are separately distributed instead of in one set, more bytes of data are necessary which increases the load on the communication links (section 6.4.2). In this thesis clusters are used to reduce computation time (see section 4.1) and to reduce the load on the communication links. In our case the box around a number of tasks indicates that the tasks process sequentially a set of track hypotheses corresponding with a cluster. In section 5.2 it has been shown that both the complexity to create the set of track hypotheses and the complexity to determine one or more good solutions for the MDA problem using a fast heuristic are exponential in the number of data sets used by the sliding window. For each cluster the following tasks are defined:

- **mean track and cluster ellipsoid determination**

  The clustering approach is based on the cluster ellipsoid approach, proposed in section 4.5.2.2, and the mean track approach, proposed in section 5.5. It is assumed that it is sufficient to (re-)calculate the different cluster ellipsoids and mean tracks once each time interval $\Delta T$ (e.g. $\Delta T = 5$ sec.) during processing of a measurement. Each cluster ellipsoid is predicted to the time of the other measurements falling in this time interval.

- **prediction, correlation and track maintenance**

  If a measurement correlates with a certain cluster, all mean tracks in that cluster are predicted to the time of that measurement. If a measurement
correlates with a mean track, it is correlated with all the track hypotheses in the related target tree. Finally, for each found (measurement-track hypothesis) pair, track maintenance is carried out. Furthermore, for each measurement also a new target tree is created.

The task to decompose the MDA problem into smaller, independent MDA problems is based on the decomposition algorithm described in section 5.6. Each of the found independent MDA problems is solved in parallel by a separate task, using for example the SGTS algorithm [32]. In section 6.5.1 and section 6.5.2, reasonable estimates for the costs to carry out the cluster-related tasks and the task to determine one or more solutions for an MDA problem will be given, which will be then used to map tasks onto the undirected processor graph.

6.5.1 Clusters

$S$ sensors contribute data sets to the sliding window. Each cluster corresponds with a significant number of real, but unknown targets and the number of measurements in data sets produced by a certain sensor is reasonably constant. If a cluster exists during a significantly long time interval, it is reasonable to assume that the cluster contains mainly mean tracks with similar dynamical behavior, operating closely together. If at a certain moment one of the mean tracks shows deviating dynamic behavior, there is a significant probability that the mean track (and the corresponding target tree) will be removed from the cluster.

By pruning all unlikely track hypotheses, the number of previously established track hypotheses stabilizes after a certain time interval to an average number $t = \sum_{i=1}^{n_c} t_i$ (section 6.2), where $n_c$ is the number of clusters at the time the new data set has to be processed. The average number of previously established track hypotheses, which is assigned to cluster $i$, is $t_i$. The average number of measurements assigned to cluster $i$ from a data set collected by a specific sensor is $N_i$. $N_i$ is the maximum number of measurements in any one of the processed data sets assigned to cluster $i$. The average total number of target trees at the time of the last measurement in a data set is given by $T_i = t_i + w_s \times N_i - 1$. The maximum number of target trees is given by $T_i = t_i + w_s \times N_i - 1$. Only the relevant contributions to the computation time are considered in this section. It is also assumed that cluster $i$ has been assigned to processor $j$ and that the number of clusters is constant during the processing of the retrieved data set. Each time, when mean tracks are (re-)calculated or correlations with track hypotheses in a target tree are carried out, the number of track hypotheses in the related target trees is determined and used to determine the average number of track hypotheses for the specific trees.

- Mean track and cluster ellipsoid determination

For each time interval $\Delta T$, the different mean tracks and the cluster ellipsoids are recalculated once for cluster $i$. The average number of track hypotheses for tree $h$ is given by $n_h$. If during processing of the new data set the time interval $\Delta T$ is exceeded, the different mean tracks and
the cluster ellipsoid have to be recalculated. The computation time to construct the mean tracks and the cluster ellipsoid is given by

\[
f_{pl}^j = \left( \sum_{l=1}^{T_i} \bar{n}_i^l - 1 \right) + (\bar{T}_i - 1) \times (n + n^2) \times c_s + \left( \sum_{l=1}^{T_i} \bar{n}_i^l + \bar{T}_i \right) \times c_p
\]

where \( n \) is the dimension of the state vector, \( c_s \) is the computation time necessary for a summation on processor \( j \) and \( c_p \) is the cpu time to predict a state vector and error covariance matrix. If the time interval \( \Delta T \) is not exceeded during processing of the data set, it is only necessary to predict the cluster ellipsoid and the different mean tracks to the time of the different measurements. This means that

\[
f_{pl}^j = \bar{N}_i \times [\bar{T}_i + 1] \times c_p
\]

The complexity to determine a mean track for each target tree is given by \( O(T) \) (eq. 5.15), where \( T \) is the number of track hypotheses in the set of track hypotheses. Furthermore, the complexity to determine a cluster ellipsoid from the different mean tracks is given by \( O(T_{mt}) \) (eq. 4.48), where \( T_{mt} \) is the number of mean tracks. The total complexity for this step is \( O(T) + O(T_{mt}) = O(T) \), because \( T_{mt} \ll T \). Two different upper bounds for \( T \) are considered. First the number of track hypotheses has stabilized and the upper bound is given by \( T_{up} = N_{w_m}^{w_m} + 1 \) (section 6.2), where \( w_m \) is the data set during which the stabilization in the number of track hypotheses occurred. Setting \( T = c \times T_{up} \) \((0 \leq c \leq 1)\), the complexity for this step is given by \[68\]

\[
O(N_{i}^{w_m} + 1)
\]

If stabilization has not yet occurred, the upper bound for the number of track hypotheses is \( T_{up} = t_i \times N_{i}^{w_s} \) (appendix D) and the complexity is given by

\[
O(t_i \times N_{i}^{w_s})
\]

- **Correlation, prediction and track maintenance**

Let for the specific processor the computation time for a correlation be \( c_s \). On average \( \bar{N}_i \) measurements from the considered data set are assigned to cluster \( i \). This means that the estimated number of correlations is given by \( \bar{N}_i \times \bar{T}_i + 1 \approx \bar{N}_i \times \bar{T}_i \). Furthermore, if we assume that a measurement correlates with \( T \) mean tracks, this means that the estimated number of correlations with track hypotheses is given by \( \bar{T}_i \times (\sum_{l=1}^{T_i} \bar{n}_i^l) \). When a fraction \( a \) \((0 \leq a \leq 1)\) of the correlations with track hypotheses is successful, the estimated number of track maintenance operations is given by

---

\(^4\)The summation of vectors and matrices are taken into account.
6.5. SCHEDULING OF TASKS

\[ a \times \frac{T}{T_i} \times (\sum_{i=1}^{T} \bar{n}_i^t). \] The estimated computation time is given by

\[ f_{pj}^j = \bar{N}_i \times [ \bar{T}_i \times c_s + \frac{T}{T_i} \times (\sum_{i=1}^{T} \bar{n}_i^t) \times (c_s + a \times c_m)] \quad (6.13) \]

where \( c_m \) is the computation time necessary to carry out track maintenance for a track hypothesis.

The upper bound on the number of operations contains the following contributions:

1. Assuming a maximum of \( N_i \) measurements, the maximum number of correlations with mean tracks is given by \( N_i \times T_i \approx w_s \times N_i \times N_t \), assuming that \( t_i \ll w_s \times N_t \). The complexity of this step is \( O(w_s \times N_i^2) \);

2. If we assume a maximum of \( T \) successful correlations with mean tracks and that the track hypotheses are equally distributed over the created trees, the upper bound on the number of correlations with track hypotheses is given by

\[ \begin{align*}
    N_s &= N_i \times \frac{T}{T_i} \times N_i^{w_m+1} \approx \frac{T}{w_s} \times N_i^{w_m+1} & \text{in case of stabilization} \\
    N_{ns} &= N_i \times \frac{T}{T_i} \times t_i \times N_i^{w_s} \approx \frac{T}{w_s} \times t_i \times N_i^{w_s} & \text{if the number of established track hypotheses has not yet stabilized, where the number of track hypotheses in the } T_i \text{ trees is given by } t_i \times N_i^{w_s} \text{ (appendix D). The complexity is } O(\frac{T}{w_s} \times N_i^{w_m+1});
\end{align*} \]

3. If a fraction \( a \ (0 \leq a \leq 1) \) of the correlations with track hypotheses is successful, the upper bound on the number of track maintenance operations is given by \( a \times N_s \) in case of stabilization in the number of track hypotheses, otherwise by \( a \times N_{ns} \). In case of stabilization the complexity is \( O(\frac{T}{w_s} \times N_i^{w_m+1}), \) otherwise \( O(\frac{T}{w_s} \times t_i \times N_i^{w_s}). \)

After applying the rule for summation on the different complexity contributions \([68]\), in case of stabilization of the number of established track hypotheses the final complexity of this step is given by

\[ O(w_s \times N_i^2) + O\left(\frac{T}{w_s} \times N_i^{w_m+1}\right) + O\left(\frac{T}{w_s} \times N_i^{w_m+1}\right) = O\left(\frac{T}{w_s} \times N_i^{w_m+1}\right) \quad (6.14) \]

If stabilization has not yet occurred, the final complexity is given by

\[ O(w_s \times N_i^2) + O\left(\frac{T}{w_s} \times t_i \times N_i^{w_s}\right) + O\left(\frac{T}{w_s} \times t_i \times N_i^{w_s}\right) = O\left(\frac{T}{w_s} \times t_i \times N_i^{w_s}\right) \quad (6.15) \]

The estimated total computation time for processor \( j \) to carry out the significant operations for cluster \( i \) during the problem initiation phase is given by (section 6.4)

\[ T_{ij}^p = f_{p1}^j + f_{p2}^j \quad (6.16) \]
CHAPTER 6. PARALLEL PROCESSING MODEL

The time $T_{i,kj}$ to relocate cluster $i$ from processor $k$ to processor $j$ can be estimated using the estimated number of track hypotheses $\sum_{l=1}^{T_l} \tilde{n}_l$ contained in the average number of target trees $\tilde{T}_i$ contained in cluster $i$ and applying eq. 6.28, given in section 6.6.2. In case of stabilization of the number of established track hypotheses, the full complexity for both task contributions for cluster $i$ is determined by

$$\max(O(N_{w,m}^{w_{m+1}}), O(\frac{T}{w_s} \times N_{w,m}^{w_{m+1}}))$$ (6.17)

otherwise by

$$\max(O(t_i \times N_{w,m}^{w_{m}}), O(\frac{T}{w_s} \times t_i \times N_{w,m}^{w_{m}}))$$ (6.18)

The possible acceleration in computation time by carrying out the cluster-oriented calculations in parallel is defined as

$$\lambda = \frac{\min_{j=1}^{p} \sum_{i=1}^{n_c} \tilde{c}_{ij}}{\max_{j=1}^{p} \sum_{i \in \{task_1, \ldots, task_{n_c}\}} \tilde{c}_{ij}^*}$$ (6.19)

where $p$ is the number of processors in the multiprocessor, $\tilde{c}_{ij} = \tilde{T}_{ij}^p$, $\tilde{c}_{ij}^* = \tilde{c}_{ij} + \tilde{T}_{i,kj}$, $\tilde{T}_{i,kj}^p$ is the estimated cost to relocate cluster $i$ from processor $k$ to processor $j$ and $n_c$ is the number of clusters. The numerator is determined by processing sequentially all tasks on the fastest processor. The denominator is determined by the last processor of the multiprocessors, which finishes its allotted tasks assuming that all tasks are executed in parallel.

![Simulated multiprocessor](image)

Figure 6.8: The simulated multiprocessor.

Consider the following example. The heuristic mapping algorithm is used to map forty tasks on a simulated multiprocessor consisting of 3 equal processors (Fig. 6.8). At the start of each simulation run, each task has earlier been allocated to a processor by drawing random numbers from a uniform distribution function. When the next retrieved data set has to be processed, the algorithm possibly reallocates tasks to different processors. Between each pair of processors...
6.5. SCHEDULING OF TASKS

Figure 6.9: Forty tasks have been mapped on a loosely coupled multiprocessor consisting of three equal processors \( w_s = 5 \), using 3 different intervals to determine estimates for the acceleration in computation time \( \lambda \).

is a communication link with a capacity of 10 Gigabit per second (Gbps). The number of correlations for cluster \( i \) was calculated with

\[
N_{i_{cor}} = n_i \times t_i \times N_{i_{ws}}
\]

(6.20)

where \( n_i \) is the number of measurements assigned to cluster \( i \), \( t_i \) is the number of previously established track hypotheses and \( N_i \) is the maximum number of measurements in any one of the data sets within the sliding window. Only polar correlations have been taken into account. The cost of a single correlation was determined for a Dell Notebook with a 1.7 GHz Pentium 4 processor, using \( 10^4 \) samples. To determine the number of track hypotheses in cluster \( i \), the equation

\[
N_i = t_i \times N_{i_{ws}}
\]

(6.21)

is used, which has been derived in appendix D. Three experiments with different interval sizes were carried out, corresponding with tasks to carry out correlations with an increasing variation in the number of measurements with clusters with an increasing variation in the number of hypotheses. In each Monte Carlo run the number of hypotheses in a cluster and the number of measurements to be correlated was determined by choosing randomly the parameters \( N_i \), \( n_i \), and \( t_i \) for cluster (task) \( i \), using a uniform random number generator. The parameter \( N_i \) was determined from the interval \([n_{min}, n_{max}]\), which can be read from Fig. 6.9. The parameter \( n_i \) was determined from the interval \([n_{min}, N_i]\), and the parameter \( t_i \) from the interval \([t_{min}, t_{max}]\), which can be also be read from Fig. 6.9. If \( t_i = 8 \) and \( N_i = 12 \), the number of track hypotheses in cluster \( i \) is 1,990,656 and the number of correlations is 23,887,872 corresponding with a computation time of 525.53 seconds. In experiment three the largest cluster is determined by \( t_i = 18 \), \( n_i = 20 \) and \( N_i = 20 \). For that case the number of
track hypotheses is 57,600,000 and the number of correlations is 1,152,000,000 corresponding with a computation time of 25,574 seconds. For an operational system, depending on the scenario the number of track hypotheses is normally considerably smaller and the computation time to instantiate the complete MDA problem and to determine one or more solutions is only a few seconds.

In Fig. 6.9, for each experiment a histogram is presented which shows the number of times a specific speedup ($\lambda$) is obtained. The results for the first experiment are shown in the first histogram, for the second experiment in the second histogram and for the third experiment in the third histogram. The speedup indicated by $\lambda$ is indicated along the x-axis. During each simulation run, a total number of 40,000 Monte Carlo runs has been made. When the size of the chosen intervals for $N_i$, $n_i$ and $t_i$ increases, the position of the peak shifts to lower speedup values and the width of the resulting peak increases due to the larger variation in the computation time necessary for the different tasks. In each run a speedup $\lambda \geq 2.4$ has been obtained, which shows that the heuristic mapping algorithm succeeds rather well in balancing the load and performs well in the considered cases.

### 6.5.2 Independent MDA problems

In the track-oriented approach target trees are constructed, which postulate the existence of a target, which is the possible origin of the assigned correlating measurements in the different branches. The likelihood of the assignment hypothesis, corresponding which a branch, is expressed by a weighting factor. A strong component contains one or more target trees, representing a group of postulated targets operating closely together. It is assumed that each found strong component corresponds with a group containing a significant number of real, but unknown targets and that the expected number of measurements per sensor scan, originating from the group, is reasonably constant. If the group operates together during a certain time interval and it is observed by one or more sensors, it is reasonable to assume that a matching strong component will be maintained during the same time interval, containing target trees with track hypotheses with similar dynamic behavior (section 4.5.2.2). However, if at a certain moment one of the postulated targets shows deviating dynamical behavior, there is a significant probability that the corresponding target tree will disappear very soon from the strong component.

Assume that the processing time on processor $k$ has to be estimated for determining one or more solutions for strong component $i$. During a time interval, a set $T_i$ containing $n_i = |T_i|$ processing times $T_{ij}^p$ ($j \in [1, p]$) for carrying out this task on different processors has been collected. Using the collected processing times $T_{ij}^p$, an average processing time for processor $k$ is calculated, using

$$T_{ik}^p = \frac{1}{n} \times \sum_{j \in T_i} \frac{T_{ij}^p \times \mu_j}{\mu_k} \tag{6.22}$$

where $\mu_k$ is the service rate of processor $k$. Neglecting the time to decompose the graph (see section 5.7), the possible acceleration in calculation time by

---

5It is possible to collect sufficient statistical information on processor $k$, but it takes much more time.
carrying out the tasks to determine one or more solutions for the MDA problems, corresponding with \( e \) strong components, is defined as

\[
\lambda = \frac{\min_{j=1}^{p} \sum_{i=1}^{e} \bar{c}_{ij}}{\max_{j=1}^{p} \sum_{i \in \{\text{task}_1, \ldots, \text{task}_k\}} \bar{c}^*_{ij}}
\]  

(6.23)

where \( e \) is the number of strong components, \( \bar{c}^*_{ij} = \bar{c}_{ij} + \bar{T}_{i,lj} \), \( \bar{c}_{ij} \) is the cost to relocate task \( i \) from processor \( l \) to processor \( j \). The cost \( \bar{T}_{i,lj} \) to relocate a strong component can be estimated in the same way as the cost to relocate a cluster (see section 6.5.1).

### 6.5.2.1 Complexity

It is assumed that each of the independent MDA subproblems is solved with the SGTS algorithm [32]. The complexity of this algorithm is determined by sorting the set of generated track hypotheses containing \( T \) track hypotheses. In the original version of the algorithm, the Quicksort algorithm is used to sort the track hypotheses, which means that the complexity of the SGTS algorithm is \( O(T^2) \) (section 6.2). When the Heapsort algorithm is used instead, the complexity of SGTS is given by [4]

\[
O(T \times \log(T))
\]  

(6.24)

A maximum of \( a \times T \) track hypotheses is contained in the target trees assigned to strong component \( i \), where \( 0 \leq a \leq 1 \) and \( T \) is the upper bound on the number of track hypotheses. Assuming that stabilization in the number of already established track hypotheses has occurred during the processing of data set \( w_m \) the upper bound on the number of track hypotheses is \( T = N^{w_m+1} \), otherwise the number of track hypotheses is given by \( T = t \times N^{w_s} \), where \( N \) is the maximum number of measurements in any one of the data sets within the sliding window and \( t \) is the number of already established track hypotheses (section 6.5.1). The complexity for SGTS is given by eq. 6.24. In case of stabilization the complexity to determine one or more solutions is

\[
O(N^{w_m+1} \times \log(N^{w_m+1}))
\]  

(6.25)

otherwise

\[
O(t \times N^{w_s} \times \log(t \times N^{w_s}))
\]  

(6.26)

### 6.6 Further improvement

The objective of parallelizing an MDA algorithm is to minimize the risk that for certain complex multitarget scenarios the realtime requirement is not met. For clusters and strong components, equations for the possible acceleration in computation time have been derived in section 6.5.1 and 6.5.2. In section 6.6.1 the important subject of large clusters will be discussed. Furthermore, in section 6.6.2 the influence of a limited communication bandwidth will be discussed. To solve both problems, the concept of partial clusters is introduced in section 6.6.3.
6.6.1 Large clusters

The Monte Carlo experiments, presented in 6.5.1, have been repeated with the possibility that one of the 40 clusters grows rather strongly in the number of previously established track hypotheses \( t \) and the number of measurements \( N \). The parameters are determined in the same way as discussed earlier, but for the first cluster they are multiplied by 3. At the start of the simulation, the task corresponding with the large cluster has earlier been allocated to processor 3. The simulation results are shown in Fig. 6.10. The results for the first experiment are shown in the first histogram, for the second experiment in the second histogram and for the third experiment in the third histogram. The speedup indicated by \( \lambda \) is given along the \( x \)-axis. For none of the three experiments

\[
\lambda \approx \frac{\frac{n_i}{\mu_j}}{\mu_k} = \frac{\mu_k}{\mu_j} \leq 1 \quad (6.27)
\]

Figure 6.10: Calculation acceleration for one extremely large task and 39 significantly smaller tasks have been mapped on a loosely coupled multiprocessor consisting of three processors. For this example, equal processors are assumed and \( \lambda \) is defined by eq. 6.19.

a significant acceleration was obtained and most of the time two of the three processors were idle. This result is also supported by equation 6.19 and 6.23: if one of the tasks needs an excessive amount of computation time, a speedup in the neighborhood of 1 results and load balancing on the three processors is not possible anymore. For the second and third experiment the width of the peak increases due to the increasing probability that more larger clusters appear, increasing the probability on a speedup larger than 1.

Consider eq. 6.19 and assume that cluster \( i \) needs an excessive amount of computation time. Processor \( j \) is one of the processors with the highest service rate. Furthermore, the completion time on the multiprocessor is determined by cluster \( i \), which is running on processor \( k \). In that case eq. 6.19 is reduced to
6.6. FURTHER IMPROVEMENT

which means there is no acceleration. The acceleration is less than 1, when cluster \( i \) is processed on a processor with a lower service rate than processor \( j \).

6.6.2 Limited communication bandwidth

The loosely coupled multiprocessor has a communication network, which possibly has a bandwidth \( \geq 1 \) Gbps when fiber optics is used [21]. Assume that the different clusters have been successfully scheduled onto the available processor network. When the next retrieved data set has to be processed, it may be necessary to relocate tasks to different processors. In this section, the necessary communication time will be estimated.

Two different kinds of tasks have been defined: cluster-related tasks (section 6.5.1) and strong component-related tasks (section 6.5.2). For both kinds a set of track hypotheses has to be relocated. The set contains \( T \) track hypotheses and is relocated from processor \( j \) to processor \( k \). The maximum number of communication links between the two processors is given by \( n_{\text{max}} \). The bandwidth for a communication channel is \( d_{\text{bw}} \). It is assumed that the contribution of contention can be neglected. Each package of data is labelled with a standard header with a length of \( N_{h} \) bits. If the information is distributed over \( n_{d} \) data packages, the communication time necessary to relocate the set of track hypotheses is determined by

\[
\frac{T \times N_{\text{bits}} + n_{d} \times N_{h}}{d_{\text{bw}}}
\]

(6.28)

where \( N_{\text{bits}} \) is the number of bits necessary to distribute a state vector and a (partial) residual error covariance matrix or the Kalman gain for a track hypothesis.

Assume that the upper bound for \( T \) is given by \( T_{\text{up}} \). Setting \( T = c \times T_{\text{up}} \) (\( 0 \leq c \leq 1 \)), the complexity for the communication time is given by [68]

\[
O(T_{\text{up}})
\]

(6.29)

Due to the fact that the complexity of the communication time depends on the upper bound for the relevant track hypotheses set, the complexity of the communication time is equal to the complexity to determine mean tracks and a cluster ellipsoid when a MDA problem contains only a single cluster (section 6.5.1). First it is assumed that the number of track hypotheses has stabilized. In that case the upper bound is given by \( T_{\text{up}} = N_{w}^{m+1} \) (section 6.2) and the resulting complexity is given by

\[
O(N_{w}^{m+1})
\]

(6.30)

where \( N \) is the maximum number of measurements in any one of the \( w_{s} \) data sets contributing to the sliding window and \( w_{m} \) is the data set during which the stabilization in the number of track hypotheses occurred. If stabilization has not yet occurred, the upper bound for the number of track hypotheses is \( T_{\text{up}} = t \times N_{w}^{m} \) (appendix D) and the complexity is given by

\[
O(t \times N_{w}^{m})
\]

(6.31)

where \( t \) is the number of already established track hypotheses. The matter becomes sufficiently worse, if it is necessary to relocate \( n_{c} > 1 \) clusters simultaneously. Due to the limited bandwidth, the relocation of the different clusters...
has to be scheduled and large delays are possible. Using eq. 6.28, a scheduling algorithm can schedule the relocation of sets of track hypotheses and prevent contention. Depending on the obtained completion time, the scheduling algorithm decides if tasks are relocated or not. However, such scheduling cannot mitigate the risk that the realtime requirement is not met.

For the experiments which have been discussed in section 6.6.1, the average time the different communication links have been used during the different experiments is shown in Fig. 6.11. Only communication link 1 has been used for a significant amount of time. This is due to the fact that the task corresponding with the large cluster has been relocated from processor 3 to processor 1 in the different runs.

6.6.3 Distributed partial clusters

In this section a new cluster decomposition approach is proposed to minimize the risk, identified in section 6.6.1 and 6.6.2, that the realtime requirement is possibly not met.

Assume that cluster $i$ exists for a certain time interval and that the cluster has been assigned to processor $l$. Each of the $m$ contributing target trees contains a large number of likely track hypotheses. At a certain moment, the computation time necessary to instantiate the data association problem increases rapidly. At time $t_l$ processor $l$ exceeds the deadline prescribed by the realtime requirement.

The objective of the new approach is to obtain an earlier completion time for the original cluster by decomposing the cluster into partial clusters, which are independently scheduled. An earlier completion time means also that a higher speedup factor is obtained.
of track hypotheses will be decomposed in a number of smaller sets. When
the partial clusters are relocated to different processors, the data load will be
distributed, possibly in parallel, using more communication links and will be
spread out in time. The proposed solution is to divide the target trees in a
cluster in \( n \) equal parts or partial clusters. Each partial cluster contains one or
more target trees. If the obtained completion time still exceeds the deadline,
the solution is to split up the different target trees and assign the partial target
trees to partial clusters. For each cluster a list is created which contains an
overview of the different partial clusters which together form the cluster. For
each of the partial clusters, tasks are created which are scheduled by the heuristic
mapping algorithm, described in section 6.5. For each partial cluster an ellipsoid
is determined using the mean tracks corresponding with the target trees which
are a member of the partial cluster, by carrying out the calculation proposed
in section 4.5.2.2. To map the new tasks onto the processor network, it is
necessary to have an accurate estimate of the necessary processing time for each
task. Here the assumption is made that the track hypotheses within a cluster
(and the target trees) operate very closely together and have similar dynamic
behavior.\(^7\) A computation time estimate for each of the (partial) target trees
is determined by dividing the total computation time, given by eq. 6.16, by
the number of (partial) target trees. The computation time, necessary for each
of the partial clusters, is determined by the number of assigned target trees.
Each time interval \( \Delta t \) a new data set with measurements will be retrieved. It is
assumed that in each processor processing and communication are carried out
in parallel and that the partial clusters are distributed during the time interval
\( \Delta t \). Normally, each task has to be completed within this time interval. If the
bandwidth is not sufficient, it will take more time before completion of the
different tasks is obtained with an increased risk that the realtime requirement
is not met.

Assume again that cluster \( i \) needs an excessive amount of computation time
and that processor \( j \) is the processor with the highest service rate. Further-
more, the completion time on the multiprocessor containing \( p \) processors was
determined by cluster \( i \), running on processor \( k \). Now cluster \( i \) is decomposed
into \( N \) partial clusters, each of which still needs a large amount of computation
time. In this case eq. 6.19 is reduced to

\[
\lambda \approx \frac{\bar{c}_{ij}}{\max_{j=1}^{p} \sum_{i \in \{\text{partclus}_1, \ldots, \text{partclus}_N\}} \bar{c}^*_{ij}} \tag{6.32}
\]

where \( \bar{c}^*_{ij} \geq \bar{c}_{ij}/N \) due to the involved reallocation costs. The average costs are
defined by \( \bar{c}_{ij} = \bar{T}_{ij}^p \) and \( \bar{c}^*_{ij} = \bar{c}_{ij} + \bar{T}_{i,kj}^c \), where the average processing time
\( \bar{T}_{ij}^p \) is estimated using eq. 6.16 and the average communication time \( \bar{T}_{i,kj}^c \) to
reallocate task \( i \) from processor \( k \) to processor \( j \) is estimated using eq. 6.28. If
all processors are equal, the acceleration \( \lambda \) is approximated by

\[
\lambda \approx \frac{\bar{c}_{ij}}{N \times \bar{c}^*_{ij}} = p \quad (N \mod p = 0) \tag{6.33}
\]

\[
\lambda \approx \frac{\bar{c}_{ij}}{([N/p] + 1) \times \bar{c}^*_{ij}} = \frac{N}{[N/p] + 1} \quad (N \mod p \neq 0) \tag{6.34}
\]

\(^7\)The validity of this assumption is supported by the results presented in chapter 5.
where it is assumed that the bandwidth of the communication channels is sufficiently large that the communication time can be neglected.

Each available multiprocessor contains a number of identical processors. Let the probability of processor failure during a mission be given by $p_f$. To prevent performance degradation, the multiprocessor has to contain a number of backup processors, which can take over the function of one or more failed processors. Assuming that each multiprocessor contains $p$ processors, the probability of $x$ failed processors during a mission is given by the binomial distribution

$$P[x] = \binom{p}{x} p_f^x (1 - p_f)^{p-x}$$

where $x \in [0, p]$. The number of necessary backup processors is given by the expected number of failed processors $p \cdot p_f$. Assume that $p_f \in [0, 0.1]$. If $p_f \leq 0.05$, the number of necessary backup processors is 1, otherwise 2.

The example of section 6.6.1 is considered again using the loosely coupled multiprocessor with three equal processors introduced in Fig. 6.8. In each Monte Carlo run, the large cluster has earlier been allocated to processor 3. Furthermore, the large cluster is decomposed in 3 partial clusters, which have to be distributed over the three processors. Theoretically, the expected acceleration factor is 3 for this example (eq. 6.33). During the different runs, the tasks corresponding with the partial clusters have been carried out in parallel. Fig. 6.12 shows for three different experiments the number of times certain speedup values ($\lambda$) are obtained during the simulation runs. The speedup values are given along the x-axis. Again a total number of 40,000 Monte Carlo runs has been

![Graph showing speedup values for different experiments.](image)

Figure 6.12: In each experiment the large cluster is decomposed in three partial clusters.

8 $[x]$ rounds $x$ off to the nearest integer towards $-\infty$.

9 The probability of failure depends on the considered time interval. The probability that a processor fails during one day is significantly lower than during one month.
made for each experiment. Comparing this figure with the earlier obtained results given in Fig. 6.10 (section 6.5.1), we see a significant increase in speedup due to the application of partial clusters. All obtained speedup values are larger than 2.2. The broadening of the peaks is due to the occurrence of larger tasks in the different experiments. The new approach has not been applied to the other tasks.

Fig. 6.13 shows the average time the different communication links have been used during the different experiments. The two large peaks correspond with distributing in parallel the track hypotheses data corresponding with the two partial clusters. Processor 3 is connected with communication link 1 to processor 1 and with communication link 3 to processor 2 (see Fig. 6.8). Link 2 connects processor 1 with processor 2. Compared with the result presented for the large cluster in Fig. 6.11, the communication load has now been equally divided over communication links 1 and 3.

The conclusion is that the application of partial clusters in this example is very effective to accelerate the completion by carrying out the tasks for a very large cluster in parallel and by distributing in parallel the corresponding data using different communication links. Future research is necessary to determine if this conclusion is also valid for other, relevant examples.

6.7 Conclusions

In this chapter a parallel processing model has been introduced, which solves the problem to use clusters to accelerate the solution process for the general multidimensional assignment problem for loosely couple multiprocessors. Very large clusters are partitioned in a number of partial clusters, which can be scheduled like normal tasks.
It has been shown in section 6.4.1 and 6.4.2 that it is possible to define for the multidimensional assignment (MDA) problem a directed, acyclic task graph, where the nodes in the graph represent the different tasks which have to be carried out to solve the MDA problem. The input arcs to a node represent the data required by a task and the output arcs represent the data produced by a task. The direction of the arcs between the nodes defines the order in which the tasks have to be processed.

To map the different tasks found in the task graph onto a processor network, it has been shown in section 6.4.3, that the problem to find an optimal mapping or schedule for the tasks in the task graph can be formulated as a nonlinear integer optimization problem, which is known to be NP-hard. In section 6.5 the complexity to determine the optimal mapping has been derived. Due to the strong nonlinearity of the complexity function, the heuristic mapping algorithm developed by Pattipati [93] is used. In section 6.5.1 and 6.5.2 it has been shown that it is possible to estimate the different costs necessary to apply the heuristic mapping algorithm.

Two major problems applying clustering have been identified in section 6.6. Due to the possible occurrence of very large clusters which consume a very large amount of computation time, using clusters to accelerate the general multidimensional assignment problem was considered not possible for loosely coupled multiprocessors. Furthermore, the scheduling of cluster-related tasks may run into practical problems when a large amount of track hypotheses, corresponding with a very large cluster, has to be distributed using a communication network with a limited bandwidth. To solve both problems, the concept of partial clusters has been introduced in section 6.6.3. The main advantage of the new concept is that it is possible to decompose very large cluster-related tasks in smaller tasks which can be scheduled like normal tasks. A partial cluster corresponds with one or more (partial) target trees. Simulation results show that using partial clusters, it is possible to accelerate the completion by carrying out the tasks for a very large cluster in parallel and by distributing in parallel the corresponding data using different communication links.