The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Parallel Maximum Likelihood Algorithm for Robot Mapping

Dario Lodi Rizzini, Stefano Caselli
RIMLab - Robotics and Intelligent Machines Laboratory
Dipartimento di Ingegneria dell'Informazione
University of Parma, Italy
E-mail {dlr,caselli}@ce.unipr.it

***Abstract*— Several recent algorithms address simultaneous localization and mapping as a maximum likelihood problem. While many proposed methods focus on efficiency or on online computation, less interest has been devoted to investigate a parallel or distributed organization of such algorithms in the perspective of multi-robot exploration.**

In this paper, we propose a parallel algorithm for map estimation based on Gauss-Seidel relaxation. The map is given in the form of a constraints network and is partioned into clusters of nodes by applying a node-tearing technique. The identified clusters of nodes can be processed independently as tasks assigned to different processors. The graph decomposition induces also a hierarchical organization of nodes that could be exploited for more sophisticated relaxation techniques. Results illustrate the potential and flexibility of the new approach.

I. INTRODUCTION

A representation of the environment is required to achieve specific robotic tasks in several applications. Indeed, autonomously building maps of the environment has become a major problem in the robotics community. For this task the robot has at its disposal its motion information and its sensor observations, which are both affected by uncertainty. In literature, map building and localization is often referred to as *Simultaneous Localization and Mapping (SLAM)*.

Maximum Likelihood (ML) is one of the approaches developed to address this problem. According to ML, SLAM is formulated as a network of constraints. Relations among robot poses and observations are represented by constraints among nodes in a graphical model. The solution of the problem corresponds to the configuration of the graph that maximizes its likelihood or, equivalently, minimizes the least square error.

Early algorithms exploiting ML formulation of SLAM were only suitable for an offline computation of the solution. Offline methods require that all data be available at the beginning of computation. In particular, Lu and Milios [1] pioneered the maximum likelihood approach proposing a brute force technique to align range scans, once all the scans have been acquired. Gutman and Konolige [2] improved the construction of the network by introducing map patches instead of single scans and provided an effective loop detection method based on correlation. Duckett *et al.* [3] introduced Gauss-Seidel relaxation to compute the optimal solution, although in their formulation angular terms were not considered.

Recent research has focused on making these algorithms more efficient and, eventually, incremental in order to make online optimization possible. Multi-level relaxation (MLR) [4] improves the simple Gauss-Seidel relaxation by solving the network at different levels of resolution. The Treemap algorithm [5] performs efficient updates with a tree-based subdivision of the map into weakly-correlated components. The smoothing and mapping (SAM) algorithm [6] relies on a QR factorization of information matrix that allows an efficient estimation of the poses of the network nodes with an efficient back-substitution. The original algorithm has been modified to allow hierachical decomposition into submaps [7] and online update of the map when new observations are available [8]. A stochastic gradient descent (SGD) technique has been proposed [9] to compute the configuration minimizing least-square error by using a representation which enables efficient updates. An incremental variant of the algorithm relies on several improvements such as tree parameterization, adaptive learning rate and selective update rules [10].

Several improvements of previous ML methods depend on a decomposition of the graph in clusters of nodes. This decomposition can be hierarchical or concerns the portion of graph involved by the addition of new measurements, but it is allowed by the sparsity of information matrix in ML approaches that avoids repeated marginalization. Given such decomposition, the update of network configuration focuses only on a portion of the map.

Decomposition of the network allows also the introduction of *parallelism* in algorithms for map estimation. There are several advantages in a parallel design. First, map estimation is sped-up by computing solutions for different graph clusters on different threads, when a proper hardware is available. The decomposition of the problem into tasks for different processors requires a distributed implementation. Recently, the diffusion of multi-core processors has also stimulated multi-threaded single process implementations. In any case, the impact of non-parallelizable portions of the algorithms and the overhead due to data sharing and synchronization should be carefully considered. Moreover, a parallel design of SLAM algorithms makes easier their extension to a multi-robot context. When only a robot is involved, data acquisition and map-building are intrinsically serial operations. However, graph decomposition is required when the map is concurrently estimated by two or more robots. In multi-robot

contexts, map decomposition is sometimes determined by the activity of each robot, as shown in [11].

In this paper, we present an offline parallel algorithm that solves mapping problems where a set of constraints is provided as input. Our algorithm combines Gauss-Seidel relaxation and a reordering of variables that transforms the information matrix in block-bordered-diagonal form. The graph associated to the information matrix is partitioned into clusters of connected nodes. A special cluster contains all nodes separating pairs of clusters. Hence, the Gauss-Seidel update of the value of a pose in a cluster depends either on another pose in the same cluster or on a separator node. Gauss-Seidel update depends on variables whose value could have been already updated in current iteration or not, and the order of variables matters. Thus, each cluster can be computed independently if the separator set is computed after the other clusters.

Graph clustering is performed with a node-tearing heuristic algorithm. The efficiency of the proposed algorithm depends on the sparsity of the information matrix that determines the result of decomposition. Gauss-Seidel relaxation is chosen because it is easy to parallelize, even though faster algorithms exist.

The paper is organized as follows. Section II provides the general formulation of the ML paradigm and the original version of Gauss-Seidel relaxation. Section III illustrates the steps of the proposed parallel algorithm, including clustering technique, variable reordering, and threaded implementation. Section IV reports experiments to evaluate convergence and performance of the algorithm on known datasets. Finally, section V gives conclusion remarks.

II. GAUSS-SEIDEL RELAXATION FOR MAXIMUM LIKELIHOOD MAPPING

This section discusses the general formulation of the maximum likelihood (ML) approach and a solution algorithm based on Gauss-Seidel relaxation to solve the resulting equations. The SLAM problem is formulated as a graph, whose nodes correspond to the variables of the map and whose edges represent the constraints between pairs of these variables.

Graph-based SLAM can be expressed according to *feature based* or *delayed-state* representations [12]. In the following, the Gauss-Seidel relaxation is applied to the solution of the map in the delayed-state form. According to this formulation the map consists only of robot poses obtained by matching observations anchored to a local frame [1] or by marginalizing feature-based maps [13].

Then let $\mathbf{x} = (x_1 \ \cdots \ x_n)^T$ be the vector of robot poses $x_i = (x_{i_x}, x_{i_y}, x_{i_\theta})$. Let δ_{ji} and Ω_{ji} be respectively the mean and the information matrix of an observation of node j seen from node i. Let $f_{ji}(\mathbf{x})$ be a function that computes a zero noise observation according to the current configuration of the nodes j and i

$$f_{ij}(\mathbf{x}) = \begin{pmatrix} (x_{j_x} - x_{i_x}) \ \cos x_{i_\theta} + (x_{j_y} - x_{i_y}) \ \sin x_{i_\theta} \\ -(x_{j_x} - x_{i_x}) \ \cos x_{i_\theta} + (x_{j_y} - x_{i_y}) \ \cos x_{i_\theta} \\ x_{j_\theta} - x_{i_\theta} \end{pmatrix} \quad (1)$$

Thus, the error on constraint $\langle j, i \rangle$ is given by

$$e_{ji}(\mathbf{x}) \quad = \quad f_{ji}(\mathbf{x}) - \delta_{ji} \quad (2)$$

Let $\mathcal{C} = \{\langle j_1, i_1 \rangle, \ldots, \langle j_M, i_M \rangle\}$ be the set of pairs of indices for which a constraint $\delta_{j_m i_m}$ exists. Then the aim of ML approach is to minimize the negative log-likelihood or error function on the observation

$$\chi^2(\mathbf{x}) \quad = \quad \sum_{\langle j,i \rangle \in \mathcal{C}} e_{ji}^T(\mathbf{x}) \ \Omega_{ji} \ e_{ji}(\mathbf{x}) \quad (3)$$

Several numeric techniques have been proposed in order to find the minimum of $\chi^2(\mathbf{x})$. In this section, we illustrate part of the relaxation algorithm proposed by Frese *et al.* [4]. The algorithm consists of two steps. First, the observation functions $f_{ij}(\mathbf{x})$ are linearized around the current value of the network configuration $\hat{\mathbf{x}}$

$$e_{ij}(\mathbf{x}) \approx f_{ij}(\hat{\mathbf{x}}) - \delta_{ji} + J_{ij}^i(x_i - \hat{x}_i) + J_{ij}^j(x_j - \hat{x}_j) \quad (4)$$

where J_{ij}^i and J_{ij}^j are the Jacobians of the observation function with respect to x_i and x_j evaluated in point $\hat{x}_i$ and $\hat{x}_j$. Since Eq. (1) only depends on poses i and j, there are no additional terms.

Then, the resulting negative log-likelihood $\chi^2(\mathbf{x})$ is approximated by a quadratic function [4]

$$\chi^2(\mathbf{x}) \quad \approx \quad x^T \ A \ x + 2bx^T \ b + c \quad (5)$$

The minimum of the linearized function is easily found by solving the linear system $A \ \mathbf{x} = b$. The method proposed in [4] to perform this final step is Gauss-Seidel relaxation. The value of each pose x_i is computed individually by solving the single block-row equation i with fixed value of x_j $(j \neq i)$. Let A_{ij} be the block of matrix A corresponding to block-row i and block-column j; let b_i be the values for block-row i. Respectively, we have

$$A_{ij} \quad = \quad \sum_{\langle j,i \rangle \in \mathcal{C}} {J_{ij}^i}^T \ \Omega_{ij} \ J_{ij}^j \quad (6)$$

$$b_i \quad = \quad \sum_{\langle j,i \rangle \in \mathcal{C}} {J_{ij}^i}^T \ \Omega_{ij} \ (J_{ij}^i \hat{x}_i + J_{ij}^j \hat{x}_j) \quad (7)$$

The relaxed solution of equation i at step k is

$$x_i^{(k+1)} = A_{ii}^{-1} \left(b_i - \sum_{j<i} A_{ij} x_j^{(k+1)} - \sum_{j>i} A_{ij} x_j^{(k)} \right) \quad (8)$$

The estimated value of x_i is determined by the neighbor poses, either already updated $(j < i)$ or not $(j > i)$. These procedure is performed iteratively until solution is reached with enough precision. Since A is a symmetric positive defined matrix, the convergence of the algorithm is guaranteed.

Gauss-Seidel relaxation is only the basic step of a multi-level relaxation (MLR) algorithm. MLR defines a hierarchy between nodes to solve the problem at different levels of resolution in order to speed up the convergence. However, the Gauss-Seidel algorithm can be conveniently decomposed in separate tasks that are performed independently. In the next section, we describe a parallel version of Gauss-Seidel relaxation.

III. A PARALLEL LINEAR-EQUATION SOLVER

Identification of the parts of the algorithm that can be executed independently is the first step in making an algorithm parallel. The block-row update Eq. (8) of Gauss-Seidel depends on nodes x_j that are connected to the current node, i.e. the nodes with $A_{ij} \neq 0_{3\times 3}$. The structure of the linearized information matrix A depends on the connectivity of the constraints network. Since the graph is built incrementally by one or more robots adding each pose in a trajectory or eventually closing loops, the resulting matrix is naturally sparse and locally connected. Hence, a decomposition of the network into clusters follows directly from the structure of the problem. Still the clusters are connected to each other: the computation of border nodes requires data from another cluster and the single algorithm tasks cannot be executed in parallel when such dependency exists.

An important remark concerns the role of order for node variables in Eq. (8). While the order does not change the value to which the algorithm converges, it determines the dependencies among variables. In particular, at a given iteration k the updated value of pose $x_i^{(k+1)}$ depends both on already updated poses, whose index is $j < i$, and on the poses yet to be updated ($j > i$). The value of the last ones is known before starting a new iteration, so their values and the value of pose i can be computed independently. Thus, in order to compute the nodes of a cluster independently from the nodes connected to the cluster and not belonging to it, the *contour nodes* have to be computed at the end.

The suggested reordering leads to the so called *block-bordered-diagonal form* (BBD) of a matrix [14]. Figure 1 shows how it is possible to reorder variables based on a cluster decomposition. Each simple cluster (labeled with $B1, B2, B3$) has no direct connection to other clusters, except for contour nodes (labeled with a, b, c) cluster. By reordering nodes so that contour nodes are the last ones, the resulting information matrix assumes a block diagonal form with a border due to contour nodes. Each cluster can be solved independently by using the value of contour nodes at the previous step. It is therefore convenient that the contour partition be as small as possible in order to limit computation of sequential parts. In the following, we discuss how to find the clusters and the permutation that achieves BBD form.

A. Clustering nodes and reordering variables

A clustering algorithm is needed to reorder the matrix into BBD form. There are two main requirements for this algorithm. First, the number of contour nodes should be limited as much as possible. This property is achieved when

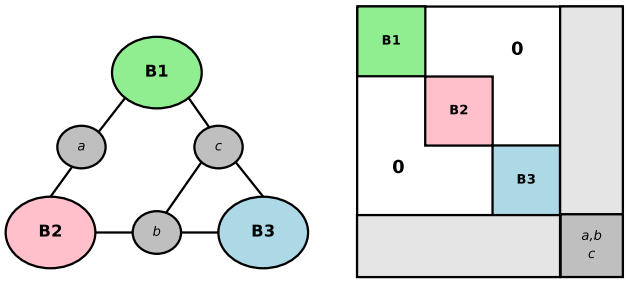

Fig. 1. Decomposition of a graph in clusters of nodes (left) and information matrix in block-bordered-diagonal form after reordering (right). Note the position of contour nodes (a, b, c) in the matrix.

the clustering algorithm picks *bottleneck* nodes. Bottlenecks nodes correspond to the minima in the size of the contour when partition consists of sets of connected nodes. Therefore the size of the clusters could be chosen to balance the computational load of each task. We propose to meet these requirements by adopting a heuristic algorithm to perform node-tearing using a contour set [15]. Node-tearing methods decompose a network into smaller subnetworks that can be solved separately. Starting from a weakly connected node, the contour set is expanded putting the new nodes in a contour set. Bottleneck is detected by searching a minimum in the size of contour set. Heuristic techniques avoid local minima and bound the size of a cluster to the interval $[perc\ n_{max}, n_{max}]$ $(0 < perc < 1)$, where $perc$ is the portion of n_{max} allowed as minimum cluster size.

Algorithm 1 illustrates how clustering is performed. Note that the condition for creating a new cluster is satisfied in two cases: when the contour is empty or when the candidate cluster set contains n_{max} nodes. Anyway, the final cluster set contains only the nodes found before the last detected bottleneck.

Each cluster is labeled with an integer, except for the contour nodes cluster that is labeled with ∞ (in real implementation, an integer larger than other labels is used). Thus, the numeric identifier of a cluster induces an order in the set of nodes. The identifiers of the simple clusters could be permuted without significant changes for parallel Gauss-Seidel relaxation. The order of the nodes belonging to the same cluster remains undefined and is chosen by implementation. Since the robot poses are usually numbered incrementally, it seems convenient to use their identifiers as second index of a lexicographic order criterion.

B. Parallel implementation

The clustering algorithm identifies the groups of nodes that can be solved independently. Thus, there is a set of tasks to be assigned to different threads or processes. A map estimator consisting of different processes would allow the execution of the algorithm on different hosts and could be applied in multi-robot contexts. The drawback of such a solution is the overhead due to the exchange of messages required at the end of each iteration. Nonetheless, the amount of exchanged data is limited to nodes of contour cluster.

Data: $\mathcal{N}$: set of nodes, n_{max}: maximum size of cluster, $perc$: portion of n_{max} allowed as minimum size of cluster.
Result: a label for each node
$\mathcal{I} = \{\}$; /* candidate cluster set */
$\mathcal{C} = \{\}$; /* current contour set */
$lastContourSize = 0$;
$iteratingSize = 0$;
foreach $n \in \mathcal{N}$ **do**
  $n.label = \{\}$;
end
while $\exists n \in \mathcal{N} : \ n.label = \{\}$ **do**
  pick n, node with minimum degree in $\mathcal{N}$);
  while $\exists n \in \mathcal{C} : \ n.label = \{\}$ **do**
    $\mathcal{I} = \mathcal{I} \cup \{n\}$;
    if $n \in \mathcal{C}$ **then**
      remove n from $\mathcal{C}$;
    end
    foreach *node* t *adjacent to* n *and* $t.label = \{\}$ **do**
      $\mathcal{C} = \mathcal{C} \cup \{t\}$
    end
    if $card(\mathcal{I}) > perc\ n_{max}$ *and* $card(\mathcal{C}) < lastSize$ **then**
      $\mathcal{T} = \mathcal{C}$; /* save cutting set */
      $iteratingSize = card(\mathcal{I})$;
    end
    if $\mathcal{C} = \{\}$ **then**
      $iteratingSize = card(\mathcal{I})$;
    else
    end
    if $=\{\}$ *or* $card(\mathcal{I}) > n_{max}$ **then**
      foreach t *in first* $iteratingSize$ *of* $\mathcal{I}$ **do**
        $t.label = newClusterLabel()$;
      end
      foreach *node* t *in cutting* $\mathcal{T}$ **do**
        $t.label = \infty$;
      end
      $\mathcal{C} = \{\}$;
    end
    $lastSize = card(\mathcal{C})$;
    pick n from $\mathcal{C}$, if exists;
  end
end

Algorithm 1: Partition of graph into clusters.

In this paper, the straightforward multi-threads solution has been chosen. A thread-pool, specifically a work crew, has been implemented in order to create and manage threads ready to accept tasks. This choice allows the creation of a correct number of threads depending on the available parallelism of the machine. After nodes reordering, the evaluation of the poses of a cluster is inserted into the queue of tasks ready to be executed by an available thread. When the execution of all tasks is completed, the poses of contour nodes are updated. The Gauss-Seidel procedure within linearization is then repeated for an appropriate number of iterations.

Synchronization is required at the end of each iteration to update data for all threads. A development of this approach would consist in reducing the dependencies between the tasks. This could be achieved only with the adoption of a more hierarchical approach than simple Gauss-Seidel relaxation. The decomposition of the network suggests itself a hierarchy to be exploited in relaxation.

IV. RESULTS

In this section, we evaluate the performance of the proposed parallel constraints solver. To test the algorithm we used the constraint networks extracted from two commonly used datasets, ACES building on UT Austin campus (ACES) and Intel Research Lab (INTEL) [16]. Constraints between pairs of poses have been obtained by matching laser scans and grid map patches. The resulting graph is the input for the experiments described in the following.

First, the convergence rate of Gauss-Seidel relaxation has been evaluated to remark its advantages and drawbacks. In particular, we compared the Jacobi relaxation method, the Gauss-Seidel relaxation method, the parallel relaxation (Parallel Gauss-Seidel), and the offline version of the tree network optimizer (Toro) [10]. Figure 2 depicts the mean error per constraint of the network for the three algorithms at different iteration steps. After few iterations, the global error decreases faster with Toro than with Gauss-Seidel and parallel Gauss-Seidel. This outcome was expected and confirms the results previously obtained in the comparison between Gauss-Seidel relaxation and stochastic gradient descent [9]. Furthermore, the parallel version performs better than the Gauss-Seidel relaxation without reordering in the case of ACES dataset. In the case of INTEL, the errors per constraints of the two relaxation algorithms are almost overlapped.

Figures 3 and 4 show the adjusted network respectively for ACES and INTEL. These experiments show that Gauss-Seidel relaxation is not the best algorithm to estimate a maximum likelihood map, but rather it is a simple method that can be easily adapted to a distributed context.

As pointed out in section III, the main issue of a parallel algorithm is graph partitioning. In the proposed method, this operation is performed by a heuristic node-tearing technique. One of the main advantages of this approach is that it accepts bounds on the maximum size of a partition and on the preferred minimum size. Figure 5 shows the results of graph decomposition for ACES and INTEL with maximum size $n_{max} = 50$ and $perc = 0.6$. Table I reports data on the results of the clustering algorithm. The mean size of clusters and the size of contour partitions depend on the topology of the constraints network. In particular, the ACES dataset has a smaller mean node degree and the number of contour nodes is limited.

Dataset	Nodes number	Mean size	Cluster number	Contour size
ACES	648	39.7	19	43
INTEL	729	30.2	14	106

TABLE I
RESULTS OF CLUSTERING.

Despite the overhead there are advantages with the multi-thread implementation of the proposed algorithm. The system has been tested on an Intel Core 2 Quad Q9450 both

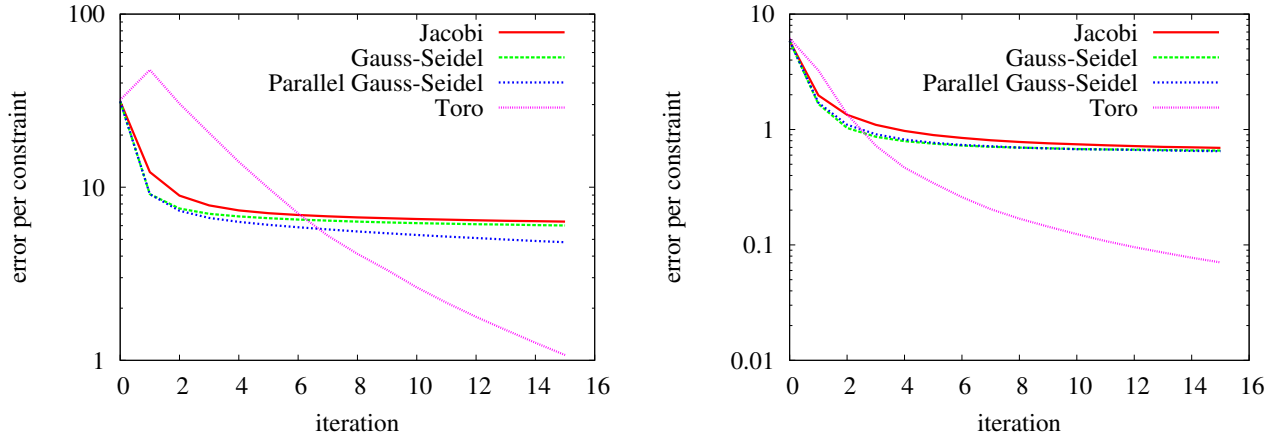

Fig. 2. Error per constraint of Gauss-Seidel relaxation and stochastic gradient descent (Toro) after each iteration for dataset ACES (left) and INTEL (right).

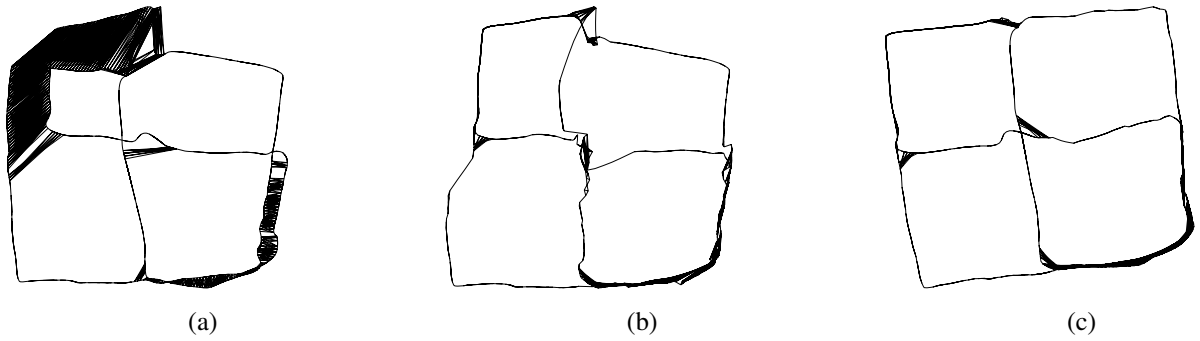

Fig. 3. Network from ACES before the optimization (a), after adjustment with Gauss-Seidel relaxation at iteration 20 (b), and with Toro (c).

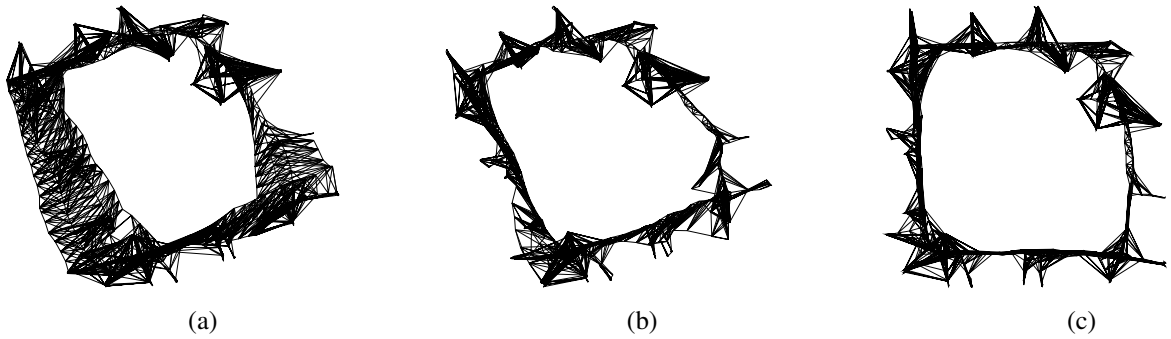

Fig. 4. Network from INTEL before the optimization (a), after adjustment with Gauss-Seidel relaxation at iteration 20 (b), and with Toro (c).

with a sequential version of the algorithm and with a variable number of threads. Results in table II report the average time required to complete a Gauss-Seidel iteration for the two datasets. Node clustering has been performed with default parameter values $n_{max} = 50$ and $perc = 0.6$. With two threads the advantage of the multi-threaded version over the sequential one is remarkable. By further increasing the number of threads, the average time slightly decreases. It should be noted that no advanced thread programming features have been exploited and thread priorities have not been modified. Hence, the parallel execution of tasks is not guaranteed and relies on operating system scheduler. However, the improvement is significant and would have possibly been even more notable with a larger network.

V. CONCLUSION

In this paper, we have presented a parallel algorithm that estimates a map consisting of poses and constraints. The proposed method combines elementary Gauss-Seidel relaxation on linearized likelihood function and node clustering into partitions. Constraint network decomposition removes mutual dependencies among the node partitions, except for special nodes called contour nodes. Formally, this operation corresponds to a permutation of the variables of the network

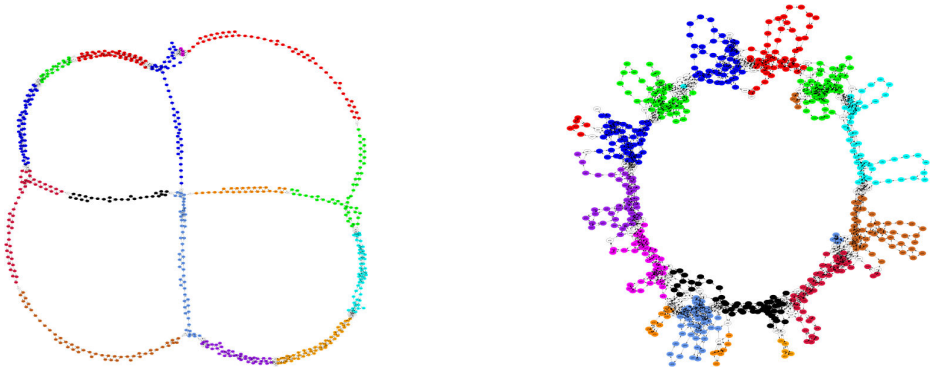

Fig. 5. The clustered graph of datasets ACES (left) and INTEL (right). Clusters are identified by different colors and contour nodes appear as clear spots dividing clusters. The poses of nodes have been slightly modified by plotting program.

Thread number	Aces Time (ms)		Intel Time (ms)	
	Mean	Std. Dev.	Mean	Std. Dev.
clustering	16.453	0.339	2.769	0.122
no thread	17.668	0.699	8.485	0.722
2	5.659	0.516	5.571	0.433
3	4.507	0.418	5.643	0.721
4	4.316	0.410	5.413	0.373

TABLE II

AVERAGE TIME OF GAUSS-SEIDEL RELAXATION FOR THE ACES AND INTEL DATASET.

that transform the information matrix in block-bordered diagonal form. Thus, a Gauss-Seidel iteration is decomposed in tasks to be executed in parallel.

We implemented a preliminar version of the algorithm in a multi-thread design to exploit commodity multi-core processors. Experiments with two different datasets show the effectiveness of graph decomposition and a better exploitation of computational resources in map estimation. Although faster methods have been proposed, Gauss-Seidel relaxation has proven appropriate for a parallel design. Advantages of parallel design are apparent in multi-robot mapping and in general distributed contexts. A parallel mapping algorithm has important applications for multi-robot systems. In our future work, we expect to exploit constraints map decomposition for a more efficient ML method and, in addition to the multi-threaded version, to implement a fully distributed version of the algorithm suitable for multi-robot systems. We intend also to address the problem of distributed data association.

VI. ACKNOWLEDGMENTS

The authors gratefully thank Giorgio Grisetti for providing the Toro implementation and the datasets already processed. This research is partially supported by laboratory AER-TECH of Regione Emilia-Romagna, Italy.

REFERENCES

[1] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Journal of Autonomous Robots*, vol. 4, pp. 333–349, 1997.

[2] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 1999, pp. 318–325.

[3] T. Duckett, S. Marsland, and J. Shapiro, "Fast, on-line learning of globally consistent maps," *Journal of Autonomous Robots*, vol. 12, no. 3, pp. 287 – 300, 2002.

[4] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localisation and mapping," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 1–12, 2005.

[5] U. Frese, "Treemap: An $O(logn)$ algorithm for indoor Simultaneous Localization and Mapping," *Journal of Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006. [Online]. Available: http://www.informatik.uni-bremen.de/ ufrese/published/fresear06b.pdf

[6] F. Dellaert and M. Kaess, "Square root sam: Simultaneous location and mapping via square root information smoothing," *Int. Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1204, 2006.

[7] K. Ni, D. Steedly, and F. Dellaert, "Tectonic SAM: Exact, out-of-core, submap-based SLAM," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[8] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[9] E. Olson, J. Leonard, and S. Teller, "Fast iterative optimization of pose graphs with poor initial estimates," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2006, pp. 2262–2269.

[10] G. Grisetti, D. Lodi Rizzini, C. Stachniss, E. Olson, and W. Burgard, "Online constraint network optimization for efficient maximum likelihood map learning," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008, pp. 1880–1885.

[11] F. Dellaert and P. Krauthausen, "A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping," in *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2005, pp. 1261–1266.

[12] R. Eustice, H. Singh, and J. Leonard, "Exactly sparse delayed-state filters," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005, pp. 2428–2435.

[13] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[14] D. P. Koester, S. Ranka, and G. C. Fox, "A parallel Gauss-Seidel algorithm for sparse power system matrices," in *In SuperComputing '94*, 1994, pp. 184–193.

[15] A. Sangiovanni-Vincentelli, L. Chen, and L. Chua, "An efficient heuristic cluster algorithm for tearing large scale networks," *IEEE Transactions on Circuits and Systems*, vol. 24, no. 12, pp. 709–717, Dec 1977.

[16] A. Howard and N. Roy, "Radish: The robotics data set repository, standard data sets for the robotics community." [Online]. Available: http://radish.sourceforge.net/