Speeding Up Rao-Blackwellized Particle Filter SLAM with a Multithreaded Architecture

Bruno D. Gouveia, David Portugal and Lino Marques

Abstract—In this work we explore multiprocessor computer architectures to propose an effective method for solving the Simultaneous Localization and Mapping Problem. The proposed method makes use of multithreading to parallelize a Rao-Blackwellized Particle Filter approach. By applying the method in common computers found in robots, it is shown that a significant gain in efficiency can be obtained. Furthermore, the parallel method enables us to raise the number of particles up to values that would not be possible in a single threaded solution, thus gaining in localization precision and map accuracy. In order to analyze SLAM results, frequently used datasets by the robotics community were used, and a benchmarking metric was applied.

I. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) has gained increasing attention over the years, and today several approaches exist to address the problem with different levels of success. Despite the scientific advancements in this area, they usually accompany technological innovations, e.g., 2D SLAM reached higher levels with the introduction of 2D laser range finders (LRFs), intense research into Visual SLAM has been noted with the growing use of stereo cameras, and RGB-D SLAM research has been mostly driven by the proliferation of RGB-D sensors like the Microsoft Kinect. Still, modern day computer architectures have not been fully explored in this context and the absence of closed SLAM solutions based on parallel computing techniques is surprising.

Beyond proposing new techniques to reduce sensor measurement noise and produce accurate representations, roboticists have increasingly made an effort to use efficient data structures in their methods, such as pose graphs, to reduce memory and processing requirements. In this paper, we propose to leverage the distinct processing units that the vast majority of modern computers have by distributing the computation load of a Rao-Blackwellized Particle Filter (RBPF) SLAM approach. Besides presenting a new parallel SLAM method, which leads to a more balanced utilization of the processing resources, we provide a performance and CPU load analysis on a modern single-board computer (SBC) and on a commonly used netbook with a limited computational capability. We show that the approach developed enables us to increase the overall CPU utilization when compared to classical single threaded solution, therefore increasing localization and mapping performance.

In the next section, seminal work on SLAM and efficient particle filter methods is reviewed, and in Section III, the multithreaded approach proposed is described. Afterwards, in section IV we discuss the results of applying the approach described in this paper, which is compared against a single threaded solution. Finally, the article ends with conclusions.

II. RELATED WORK

In the field of mobile robotics, solutions for the problem of SLAM rely on probabilistic frameworks, which account for sensor measurement noise and estimation uncertainty. Classical techniques make use of Kalman Filters (KFs) [1] and Particle Filters (PFs) [2] to incrementally compute joint posterior distributions over robot poses and landmarks. Along the years, several enhancements based on these approaches have been described in the literature, such as Extended Kalman Filters (EKFs) and Rao-Blackwellized Particle Filters (RBPFs). In addition, graph-based algorithms, e.g. [3], have also gained popularity due to the efficiency gained when maintaining large-scale maps, which stems from discarding irrelevant measurements, and adopting graph optimization processes.

Also popular nowadays are methods which take advantage of high scanning rates of modern day Light Detection And Ranging (LIDAR) technology. These methods rely heavily on scan matching of consecutive sensor readings, with combination of other techniques like multi-resolution occupancy grid maps [4] or dynamic likelihood field models for measurement [5]. Despite the evident advancements in research in the SLAM problem, a robot with such capabilities still has to be equipped with a modern computer to adequately handle the processing and memory requirements.

Beyond the usage of optimized data structures and algorithms, some authors have turned to modern day computer architectures to handle heavy processing SLAM in real time. Multicore processors are more efficient in terms of energetic consumption since the speed requirements on each parallel unit are reduced, allowing for a reduction in voltage. For example, the authors in [6] propose a simple and effective scan-matching module that can be potentially used in several SLAM algorithms, which takes advantage of the Single Instruction, Multiple Data (SIMD) units available in modern processors. This allows the processor to execute an operation over a data vector in the same instruction time. Using this
approach, an average speedup\(^1\) of 3.5 compared to the non-SIMD version of the algorithm was observed. Additionally, the authors in [7] have proposed a Visual SLAM module implemented in the Compute Unified Device Architecture (CUDA), which runs at a 15 Hz rate. Their approach performs sparse scene flow, real-time feature tracking, visual odometry, loop detection and global mapping in parallel.

Some effort has also been conducted to efficiently distribute the computation load of classical particle filter implementations on multiprocessing systems in a variety of applications. Chitchian et al. [8] have proposed a NVIDIA Graphics Processing Unit (GPU) implementation of a particle filter framework for complex real-time estimation problems, using both CUDA and the Open Computing Language (OpenCL). This was applied in a complex robot arm application with 48 state variables and using a classical particle filtering approach with 1 million particles. The authors conclude that it would not be possible to run the system in real-time without resorting to the new generation of GPUs with several cores. Similarly, Par and Tosun [9] have addressed classical PF-based Sequential Monte Carlo Localization of a vehicle in a known map with multicore and manycore processors. They have used the Open Multi-Processing (OpenMP) programming model for the parallelization of the predict and update phases of the PF in a multicore CPU, and also implemented a GPU version, obtaining speedup values of up to 4.7 and 75 respectively. Perhaps, the work most closely related to ours is described in [10], where a parallel implementation of a RBPF was also addressed. The authors not only propose to assign different particles among available processing elements in order to compute their weights, but also to conduct the resampling process. Despite the potential of the work, it was only validated by tracking a maneuvering vehicle in Matlab simulations.

Other related philosophies for distributing the computation load in robotic application are emerging. These are the cases of Cloud Robotics and Robotic Clusters. On one hand, in [11] robots performing visual SLAM query a Cloud service to run expensive steps of the algorithm and allocate storage space, thus freeing the robot embedded computers from most of the computation effort. However, this implies the continued availability of the Cloud entity. On the other hand, [12] empowers heterogeneous robots with the ability of sharing their processing resources when solving complex collective problems. This was applied to the problem of topological map merging in a distributed multi-robot system. These philosophies differ from the one presented in this paper in the sense that they imply communication to/from external processing units, and network latency is non negligible in real-time tasks. Also, our approach does not suffer from network failures, since all processing units belong to the same computing component.

In contrast to all previously mentioned works, we leverage not only modern day computer architectures composed of multiple CPU units by presenting a multithreaded RBPF approach, but also apply it in a 2D SLAM application for the first time as far as our knowledge goes. Furthermore, we contribute to the state-of-the-art by quantitatively analyzing the benefits of such approach in commonly used and well-known SLAM datasets.

### III. Multithreaded RBPF SLAM

Recognized as the most widely used laser-based SLAM algorithm in robots worldwide, GMapping is an implementation of the RBPF SLAM approach presented by Grisetti et al. [13]. Being also a grid-based algorithm, it maintains an occupancy grid map divided in cells, which represent whether the state of the corresponding space is occupied (e.g., an obstacle), free (open space) or unknown. Furthermore, since it uses a PF implementation, each particle encodes a pose of the robot and the corresponding laser scan, thus providing an hypothesis to the localization and mapping problem at a given time step.

There are several advantages for choosing GMapping as the key SLAM approach addressed in this study. Firstly, it is an open source ready-to-use algorithm available in the Robot Operating System (ROS).\(^2\) Secondly, it is well-established

---

\(^1\)In parallel computing, speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm.

\(^2\)http://wiki.ros.org/slam_gmapping
with recognized performance as proven by recent works, e.g. [14]. Finally, the nature of the algorithm is particularly suitable for parallelization, given that each particle is computed independently and there is no data flow between them. For more details on the algorithm the reader should refer to [13].

Every time a new laser scan is acquired by the robot, the algorithm follows all the steps illustrated in the flowchart of Fig. 1. Using a profiling tool, it was possible to verify that on average, 98.47% of the computation time of the ProcessScan function is spent in the Scan Matching step. This occurs because scan matching involves comparing the set of data returned from the LRF with the 2D map obtained thus far, which is a process that is repeated for all the $N$ particles involved. Therefore, it is executed many times during localization and involves several mathematical operations, representing a high computational burden for any SLAM algorithm. Additionally, it should be noted that scan matching occurs when the distance traveled by the robot surpasses a predetermined linear threshold $\gamma$ or an angular threshold $\theta$ (cf. Fig. 1), which are important parameters of the algorithm. Hence, we expect that frequent computation peaks will occur every time the condition is verified. This is in fact shown in Fig. 2a, where a run of the algorithm in an Asus eeePC 1015 netbook with $N = 5$, $\gamma = 1.0\ m$ and $\theta = 0.5\ rad$ is depicted.

While in Fig. 2a the algorithm is run with a low number of particles, in Fig. 2b we present a similar chart with $N = 40$. In this situation, one cannot distinguish the computational peaks, because scan matching takes too long.

![Fig. 2: CPU load in the single threaded GMapping algorithm running on an Asus EeePC 1015 with different number of particles $N$.](image)

![Fig. 3: CPU load in the Multithreaded GMapping algorithm with $N = 40$ running on an Asus EeePC 1015.](image)

Therefore, when it is necessary to proceed with the next scan matches, the previous ones still have not finished processing, and consequently, the algorithm skips steps. This has a tremendous impact on localization and mapping, as shown later on. Thus, in this case the algorithm will only run in real-time if the robot moves slowly, giving the computer enough time to process laser scan matching. Despite the computer being subjected to overload, it is clear that the computational resources are not being efficiently used, since the total computational load never exceeds 50%. The reason behind this is that the algorithm was not designed to run in parallel. More particularly, only one processing unit is scheduled for the heavy scan matching task. Thus, we propose to distribute this task among the existing processing units.

As previously mentioned, after conducting a data dependence analysis it was verified that laser scan matching is computed independently for each particle, which represents the best case scenario for parallelization, since there is no need to share data between particles. This is known as an embarrassingly parallel problem. This study not only led us to parallelize this step, but also to keep the other steps as single threaded, since the overhead of parallelization together with their low computational demand would render the parallelization inefficient in these steps. Thus, our method divides the $N$ particles in equally sized chunks and distributes them over the available processor cores while keeping the rest of the algorithm single threaded.

In order to implement the parallel architecture, we have used OpenMP, by directly modifying the original GMapping package available for the ROS framework. OpenMP is supported by a large number of compilers on different platforms, providing a simple and elegant solution to the single threaded limitation of the original algorithm, without the need to restructure and modify the existing code or use external libraries. This approach has the ability to autodetect the number of available processor cores, also providing an option to choose the number of worker threads. This way, the algorithm adapts itself to the host architecture, making an efficient use of the processing resources, as illustrated by Fig. 3. Furthermore, this approach is not limited to GMapping, having also the potential to be used in any SLAM algorithm that relies on laser scan matching in a similar particle-independent way.

The benefits of using a multithreaded version of the
TABLE I: Specifications of the Asus EeePC netbook and the Odroid X2 single-board computer used in the experiments.

<table>
<thead>
<tr>
<th>Tested Computers</th>
<th>Asus EeePC 1015 PEM</th>
<th>Odroid X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Atom™ N550 with Hyper-Threading</td>
<td>Exynos4412 Prime ARM Cortex-A9 Quad Core</td>
</tr>
<tr>
<td># Cores</td>
<td>2 Physical (4 Virtual) Cores</td>
<td>4 Physical Cores</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>1.5 GHz</td>
<td>1.7 GHz</td>
</tr>
<tr>
<td>Cache L2</td>
<td>1 MB</td>
<td>1 MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB (DDR3)</td>
<td>2GB (DDR2)</td>
</tr>
<tr>
<td>Storage</td>
<td>250GB, 5,400rpm</td>
<td>16GB, SD card</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu 12.04, 64 bit</td>
<td>Ubuntu-based Linaro 12.11, 32 bit</td>
</tr>
<tr>
<td>Approximate Cost</td>
<td>$289</td>
<td>$135</td>
</tr>
</tbody>
</table>

GMapping algorithm are analyzed and discussed in the next section. Also, the code of this work is publicly available for download.4

IV. RESULTS AND DISCUSSION

Experimental results are presented and discussed in this section. We propose to analyze the efficiency and performance gain when using multithreaded (MT) GMapping as proposed in this paper, comparing it to the single threaded (ST) one, which is available for ROS and widely used by the community. Having this in mind, we have tested 3 datasets typically used in SLAM benchmarking: ACES Building, Intel Research Lab, and Killian Court.5 These were tested in the two computer architectures depicted in Table I, both in ST and MT mode. In addition, we have also conducted experimental tests, increasing the number of particles by 30, starting with \( N = 30 \) and going up to \( N = 180 \), while stopping at the point where GMapping returns an “out of memory” error. For each different configuration we run 5 trials, resulting in a total of 250 trials of the following combinations:

1) dataset = \{ACES Building, Intel Research Lab, Killian Court\},
2) \( N \) = \{30, 60, 90, ...\},
3) method = \{ST, MT\},
4) computer = \{Asus EeePC, Odroid X2\}.

We start by analyzing the average time taken for the Scan Matching step of the algorithm in both the ST and MT method. Table II presents the average time in seconds over all the 5 trials with different configuration. Note that there are no results for the Asus EeePC 1015 for \( N > 90 \) in the ACES Building and \( N > 60 \) in the Killian Court datasets, due to the 1GB memory limitation. On the other hand, the Odroid X2 was able to handle \( N \leq 150 \) and \( N \leq 90 \) for the ACES Building and Killian Court datasets respectively.

On the Intel Research Lab, both computers were able to run experiments with \( 30 \leq N \leq 180 \).

The first immediate evidence is that scan matching takes less time to process in the Odroid X2 than in the Asus EeePC, as expected due to the former’s superior computation power. Also, it is clear that multithreading accelerates the scan matching process by an approximate factor of 2. In fact, the acceleration observed in the Odroid X2 presents an acceleration factor between [2.22, 2.08], while the one observed in the EeePC has a factor between [1.93, 2.34]. The efficiency in the Odroid X2 is superior, possibly due to its more optimized architecture with 4 de facto processors, whereas the EeePC only has 2 real cores (4 threads).

As a consequence of the above facts, it is possible to run a MT method with several more particles than a ST method in the same computer, thus gaining in localization and mapping quality. A clear example of this is shown in the results for the ACES Building in the Odroid X2, where a ST method with 60 particles takes approximately the same amount of time to process laser scan matching as the MT method with 150 particles. Furthermore, since multithreading accelerates the scan matching step it enables the robot that is collecting data to move at higher speeds. When moving at higher speeds, the algorithm reaches the \( \gamma \) or the \( \theta \) threshold quicker. Performing SLAM in real-time without dropping data is possible as long as the processing time is shorter than the time between the thresholding condition is verified.

In the analysis presented before, we have only addressed the time to process laser scan matching, which was the only step of the algorithm that was parallelized. However, it is important to understand how the overall time to run the algorithm has been accelerated with a multithreading solution. In parallel computing, it is common to use the speedup metric to measure how fast a parallel algorithm is, compared to the corresponding sequential algorithm. Thus, speedup \( \nu \) is defined as:

\[
\nu = \frac{T_{ST}}{T_{MT}}.
\]

\(^4\)https://github.com/brNX/gmapping-openmp
\(^5\)http://kaspar.informatik.uni-freiburg.de/~slamevaluation/datasets.php
where $T_{ST}$ represents the total processing time of the single threaded method, and $T_{MT}$ the total processing time of the multithreading method in the same exact conditions. In Fig. 4, the evolution of the speedup metric with increasing number of particles is shown. The illustrated curves confirm that the efficiency gain is higher when using the Odroid X2 single-board computer, where $v \in [2.08, 2.60]$, while the Asus EeePC 1015 has lower speedup, $v \in [1.45, 2.20]$. Moreover, speedup consistently increases with the processing load, i.e. the number of particles $N$. This is common in multithreading, because the overhead of parallelization, such as the creation and management of threads, is approximately the same, independently of the size of the problem. Therefore, the ratio between overhead and workload decreases, when the processing load grows. As a result, the efficiency increases with the size of the problem.

The efficiency analysis that was conducted is fundamental to assess our method. However, it is also important to discuss the impact of the multithreading algorithm in terms of performance. Particularly in SLAM, performance is related with localization accuracy and mapping precision.

Due to the acceleration observed in the scan matching step, it has been shown that the multithreading algorithm is able to handle a number of particles that cannot necessarily be handled in single threading without dropping data, e.g. Fig. 2b and Fig. 3. Hereupon, the performance obtained with the MT approach is expected to be superior in situations where the ST approach skips scan matching steps, having at least the same performance levels when this is not the case. This is clearly noticed in the ACES building using 90 particles, and in the Intel Research Lab using 150 particles (cf. Figures 5c and 6c).

Despite the evident increase of mapping quality with multithreading, visually inspection of the resulting maps does not allow a correct comparison. So, the need to precisely evaluate the results asks for a more accurate method - a quantitative scale. In this work, we make use of the benchmarking metric presented in [15] to understand the impact of multithreading in SLAM performance. This metric evaluates the accuracy of the poses in robot trajectories during data acquisition. It uses only relative relations between poses and does not rely on a global reference frame, which even allows to compare algorithms with different estimation techniques and sensor modalities. Moreover, using this metric, the error is not cumulative, instead it is isolated by comparing the displacement of the differences between estimated poses of the robot to the ground truth relation between real poses, which were manually measured by the authors in [15] for commonly used datasets.

Taking the example of the maps in Figures 5 and 6, the translational error and the angular error were calculated. In the rightmost charts (cf. Figures 5c and 6c), one can see the evolution of the translational error over the different relations, and Table III presents the mean error in translation and rotation with the corresponding standard deviation. Results demonstrate that the errors of the MT method are smaller than those of the ST method, both for the translational and rotational error. The peaks in the evolution chart show situations where relations measured by the ST method have an unusual high error, which explains why the resulting maps generated were only consistent locally, but tend to be inconsistent as a whole.

V. CONCLUSIONS

This work presents a multithreaded architecture to speed up a 2D RBPF SLAM approach, widely used by the Robotics community. We started by conducting an initial study where we identified scan matching as the most costly step of the algorithm in terms of computation. After describing the implementation details, several experimental tests were conducted using two different computer architectures and running the RBPF algorithm with increasing number of particles in commonly used benchmarking datasets. We have analyzed and compared the single threaded approach against our multithreaded implementation, and have shown that the processing time drastically decreases, as the computational resources are used in a much more efficient way. Furthermore, the multithreaded architecture allows us to increase the number of particles in the algorithm by a factor of 2 or even more. Besides the gain in computation efficiency, we have also compared SLAM performance in both methods by adopting a recognized benchmarking metric by the community. Results have shown that multithreading leads to increases in localization and mapping when the number of particles cannot be fully handled by the single threaded method.

In addition to the method and the quantitative analysis presented, the authors openly provide the code to the community in the hope that it will be useful. Even though other methods were not tested in this paper, the authors suggest that the approach can be used in the future to speed up any SLAM algorithm that relies in laser scan matching.

ACKNOWLEDGMENT

The authors would like to thank Eurico Pedrosa and Nuno Lau (University of Aveiro, Portugal) for the code that converts the log files of frequently used datasets in the robotics community to ROS.
Fig. 5: Maps obtained and translational error using the single threaded and the multithreaded method in the ACES Building dataset.

Fig. 6: Maps obtained and translational error using the single threaded and the multithreaded method in the Intel Research Lab dataset.

TABLE III: Mean translational error $\epsilon_{\text{trans}}$ and mean rotational error $\epsilon_{\text{rot}}$.

<table>
<thead>
<tr>
<th></th>
<th>ST</th>
<th>MT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACES Building ($N = 90$)</td>
<td>$0.197 \pm 2.107$</td>
<td>$3.764 \pm 15.099$</td>
</tr>
<tr>
<td>Intel Research Lab ($N = 150$)</td>
<td>$0.102 \pm 0.760$</td>
<td>$14.476 \pm 21.991$</td>
</tr>
</tbody>
</table>

REFERENCES


