6-2007

Evaluation of a Parallel Architecture and Algorithm for Mapping and Localization

Damian M. Lyons
Fordham University

Giselle R. Isner
Fordham University

Evaluation of a Parallel Architecture and Algorithm for Mapping and Localization

Damian M. Lyons and Giselle R. Isner

Abstract — **The Beowulf cluster approach to parallel computation offers a potentially cheap and robust source of computational power for high complexity algorithms in robotics. The challenge is to integrate this approach with the mobility and time critical response constraints of many robotic algorithms. The key contributions of this paper are: (1) introduction of a computational architecture for integrating a cluster into the control architecture of one or more robots, (2) a cluster implementation of Thrun et al's Concurrent Localization and Mapping (CML) algorithm, and (3) presentation of results to illustrate the performance of the implemented CML algorithm and validate the architectural approach.**

I. Introduction

Mapping and Localization are crucial underlying skills for a robot operating in a partially-known or unknown environment [4][17][7]. Mapping is the process of using information from the robot sensors, in addition to whatever a-priori map knowledge exists, to acquire a metric or topological map of the environment in which the robot is operating. Localization is the process of determining the robot's location with respect to this map. A number of algorithmic approaches have been developed for acquiring and fusing sensor data (e.g., [2][6][9]) and for estimating the map and robot location (e.g., [1][5][16][17]). In general, these approaches require the accumulation of large amounts of sensory data and the repeated integration of this data with a map model.

The computational complexity of localization and mapping algorithms is polynomial $O(n^3)$ in the number of landmarks n, depending of course on assumptions about the difficulty of the environment, the sensor details, and the specific algorithm used, all of which can be exploited to reduce the exponent. However, the quality of navigation is impacted by the value of n, and it is preferable to make this large if at all possible. On the other hand, it is desirable and for some applications (e.g., [1]) essential that map information become available to the robot as quickly as possible. Because of this, it may be advantageous to exploit parallel computational resources to reduce the computation time.

Beowulf cluster technology is a cost effective approach to parallel computation. The first Beowulf cluster was build by Becker and Sterling at the Center for Excellence in Space Data and Information Sciences in 1994. Their goal was to build "COTS (Commodity Off-The-Shelf) based systems to satisfy specific computational requirements" [12]. They called their cluster of 16 off-the-shelf DX4 processors, "Beowulf," and that name has now come to denote the entire class of COTS based cluster machines.

In this paper, we propose a Beowulf cluster architecture and algorithmic approach to address the computational needs of timely robot mapping and localization. Section II presents the overview and architecture. Section III describes the MPI implementation of Thrun et al's CML algorithm [16]. Section IV presents performance results and metrics. Section V presents our conclusions.

II. Architecture

The hybrid reactive/deliberative architecture (e.g., [3][10]) was designed to allow high-level deliberative reasoning to be integrated with lower-level reflexive behavior in such a fashion that the advantages of deliberation could be leveraged without slowing reactive response time. Similar issues relate to the use of parallel computation for mapping and localization. If the robot moves without any map, it is reduced to pure reactive navigation. Nonetheless, no useful map can typically be built until the robot has moved, and map computation can significantly deflect computational resources from timely reaction.

A. Hybrid Architecture for Map-Building

More concisely, these two conflicting computational needs are:

1. The mobile robot needs to be capable of timely action: moving to avoid unexpected obstacles and hazards as well as progress towards its goal.
2. The robot may need to devote many seconds of computation to produce map and localization information based on all the sensory data received.

This is a conflict similar to that that gave rise to the hybrid reactive/deliberative architecture, and we argue that a similar approach can be useful here.

Figure 1 shows our hybrid architecture for mapping and localization. The reactive component is responsible for collecting sensory data for two purposes: timely reactions and map-building landmarks. The timely reaction loop is completely within the reactive component; however, the

Manuscript received February 15th, 2007.

D. M. Lyons is with the Robotics and Computer Vision Laboratory of the Computer & Information Science Department, 340 JMH, Fordham University, Bronx, NY 10458 USA. (phone: 718-817-4485; fax: 718-817-4488; e-mail: dlyons@cis.fordham.edu).

G. R. Isner is a graduate student with the Computer & Information Science Department, 340 JMH, Fordham University, Bronx, NY 10458 USA. (e-mail: isner@cis.fordham.edu).

map-building landmarks are transferred on a continual basis to the deliberative component. The deliberative component is responsible for taking the landmarks and asynchronously constructing a map. The map is transferred back to the reactive component on a continual basis.

A second, deliberative planning loop is shown with the dotted lines in Figure 1. This takes navigation objectives and resolves them with the aid of the map to a set of motion goals that can be sent to the reactive component. We will not address navigation planning, and it is included only for completeness.

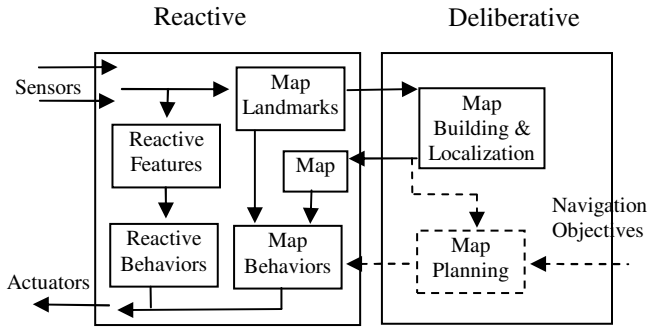


Figure 1: Hybrid Architecture for Map Building

B. *Cluster Realization of Architecture*

This architecture can be easily mapped to a robot and Beowulf cluster. In our work, the robot is a Pioneer DX2 with on-board computer and wireless network connection. The on-board computer is used to implement the Reactive component. The Deliberative component is implemented off-board, on a Beowulf cluster. The communication between Beowulf and robot is via the wireless network connection: Map landmark data is transmitted to the Beowulf on a regular basis, and the most recent estimate of the map and robot location is transmitted back to the robot.

Note that the on-board and off-board components can operate effectively with some variability in communication times: The robot can move and respond in a timely fashion using the last best map estimate transmitted, while the Beowulf can usefully continue map estimation until it receives new sensory data.

For the remainder of the paper, we will focus on the deliberative component, and the cluster implementation of map building and localization.

III. MPI Implementation of CML Algorithm

A. *CML Algorithm*

Thrun, Burgard and Fox [16] proposed a probabilistic approach to concurrent mapping and localization (the *TBF algorithm*, for convenience). They phrased the problem as a maximum likelihood estimation:

$$m^* = \underset{m \in M}{argmax}\, P(m \mid d)$$

where d is the set of sensor measurements, odometry and sonar in our case, and motion commands, and where M is the set of all possible maps and m^* is the most likely map. We will adopt an occupancy grid approach; hence, m is a grid of dimension NxN values representing a physical space. The value at location (x,y) represents the probability that the physical location is occupied. TBF is an *Expectation Maximization* (EM) based algorithm that iteratively estimates the best map based on the sensor data and estimated robot location, and then the best robot location based on the estimated map. We selected this algorithm because:

1. There are several successful cluster implementations of EM already (e.g., [11]).
2. The EM iteration loop provides a convenient basis for the continual communication between the reactive and deliberative modules.

TBF assumes two known probabilistic models:

1. The **motion model** $P(c' \mid u,c)$ that specifies the probability that the robot configuration is c' if the motion command u was executed in configuration c.
2. The **perceptual model** $P(o \mid m,c)$ specifies the probability of sensor measurement o when at configuration c in map m.

In our implementation, both models were assumed to be Gaussian and were estimated by empirical measurements over a range of motion and sensing conditions for indoor environments. The iterative two steps of the TBF algorithm are described in A.1 and A.2 below.

A.1 Expectation Step. This step uses the map m to compute estimates for the robot configuration at times $t=1,..,T$ where the sensor data measure at these times is $o^{(1)},\ldots,o^{(T)}$ and the robot motion commands are $u^{(1)},\ldots,u^{(T)}$. TBF expresses the probability that the robot is in configuration $c^{(t)}$ at time t as:

$$P(c^{(t)} \mid d,m) = \eta\, P(c^{(t)} \mid o^{(1)},\ldots,o^{(t)},m)\, P(c^{(t)} \mid u^{(t+1)},\ldots,o^{(T)},m)$$

Where η is a normalizer over all $P(c^{(t)} \mid d,m)$ and where the first probability term propagates the probability going forward from a starting position; TBF denotes this term as $\alpha^{(t)}$. The second term propagates the probability going backwards from a final position; TBF denotes this term as $\beta^{(t)}$. Both terms can be calculated recursively as:

$$\alpha^{(t)} = \eta P(o^{(t)} \mid c^{(t)},m) \int P(c^{(t)} \mid u^{(t-1)},c^{(t-1)}\, \alpha^{(t-1)})\, dc^{(t-1)}$$

$$\beta^{(t)} = \eta \int P(c^{(t+1)} \mid u^{(t)},c^{(t)})\, P(o^{(t+1)} \mid c^{(t+1)},m)\, \beta^{(t+1)}\, dc^{(t+1)}$$

The recursion is bounded by $\alpha^{(0)}$ which is 1 only at the origin of the configuration space, and by $\beta^{(T)}$ which is uniformly distributed over the configuration space.

A.2 Maximization Step. This step uses the robot configuration from step one to compute the most likely map m^* using a weight likelihood ratio (how often a map location is seen as occupied). We introduce a landmark set $L = \{l_0, l_1\}$ where l_0 indicates an unoccupied location and l_1 indicates

an occupied location. The probabilistic map can then be written:

$$P(m_{x,y}=l \mid d) = \frac{1}{T}\sum_{t=1}^{T}\int P(m_{x,y}=l \mid o^{(t)}, c^{(t)})\alpha^{(t)}\beta^{(t)}dc^{(t)}$$
$$= \frac{1}{T}\sum_{t=1}^{T}\int \eta P(o^{(t)} \mid m_{x,y}=l, c^{(t)})\alpha^{(t)}\beta^{(t)}dc^{(t)}$$

where η is a normalizer.

B. MPI Implementation

The *Message Passing Interface (MPI)* [15] is a de facto industry standard for parallel computation. MPI provides an application library that algorithm designers can use to easily avail of parallel resources. Communication in MPI can be as basic as the individual sending and receiving of messages. However, our implementation leverages the *collective communication operations* of MPI: a group of processors is coordinated in an efficient manner to perform a designated operation collectively. These operations include:

- **Broadcast:** Send data from one processor to a group of processors.
- **Scatter:** Distribute data from one processor among all processors.
- **Gather:** Collect and group data from each of the processors onto one processor.
- **Reduce:** Combine the data from a group of processors using a reduction (binary, associative) operation onto one processor.

B.1 Data Structures. There are two key data structures: the robot configuration space *C* and map *M,* both *NxN* arrays[1]. The map represents a square area of physical space of side length N_s and the spatial granularity of the map representation is therefore N_s/N. We partition the TBF calculation in a *data parallel* fashion: Each *NxN* array is divided amongst the *P* available processors on a row basis. Each processor is identified by a rank number $p \in 1, \ldots, P$. Processor *p* is assigned the calculations for a block of data from row B_LO(*p*) to B_HI(*p*) on all *N* columns[15]:

$$\text{B\_LO}(p) = \frac{pN}{P} \quad \text{B\_HI(p)=B\_LO}(p+1)\text{-}1$$

The sequence *Dp* of sensor measurements and *Dm* robot motion for each time step $t \in 1, \ldots, T$ is broadcast to all *P* processors before the first EM iteration. The perceptual *Mp* and motion *Mm* models, represented as arrays with the same spatial granularity as *C* and *M,* are also broadcast to all processors.

B.2 EM Step One. Step one calculates the robot configuration at each time *t* as the normalized product of the alpha and beta configuration estimates, $C\alpha$ and $C\beta$. At initialization, $C\alpha[0]$ is set to 0 except for the origin, and $C\beta[T]$ is set to $1/N^2$ throughout. The calculation for $C\alpha$ is shown in Fig. 2. Each processor steps through each time and each configuration in $C\alpha$. If there is a non-zero probability

```
For t ∈ 0,…,T-1
 For x ∈ 0,…,N
  For y ∈ B_LO(p),…,B_HI(p)
   If Cα[t][x][y]>0
     Cα[t+1] =
         Cα[t] + Apply(Dm[t],Mm,x,y)

 Reduce(Cα, SUM)
 Broadcast(Cα)
```

Figure 2: Alpha MPI Algorithm

of that configuration, then the motion model is used to project the time *t* motion from that configuration and modify the configuration probabilities accordingly. Although each processor takes care of a block of data, the resultant configuration may have probabilities set outside the block,

```
For t ∈ 1,…,T
 For x ∈ 0,…,N
  For y ∈ B_LO(p),…,B_HI(p)
   If C[t][x][y]>0
     M[t] = M[t] +
            Apply(C[t],Dp[t],Mp,x,y)
     IncMcounts(M[t],Dp[t],x,y)
  For x ∈ 0,…,N
   For y ∈ B_LO(p),…,B_HI(p)
    Normalize(M[t],x,y)

 Reduce(M, SUM)
 Broadcast(M)
```

Figure 3: Map MPI Algorithm

due to direction of motion and the motion model. Therefore, all the blocks need to be added together to produce the final configuration estimate. This is accomplished by an MPI sum reduction operation, and the new configuration broadcast to all (to be ready for step two).

The calculation of $C\beta$ is similar, but backwards in time, and the final configuration estimate *C* is the normalized product of both.

B.3 EM Step Two. Step two calculates the most likely map based on the sensor readings and configuration estimate. The algorithm is shown in Fig. 3. The initial map is empty for all times. Each configuration is then used in conjunction with the sensor measurements and sensor model to calculate a weighted (by *C[t]*) ratio of landmark (occupied or not) observation counts. Each processor takes care of its rows in the map and a reduction operation is used to reconstruct the full map, which is broadcast to all processors for use in step one.

Iteration of step one and step two produces better estimates of the map and robot location. In our implementation, after around typically 5 iterations, the map ceased to change.

[1] Omitting the orientation dimension for ease of presentation.

IV. Results and Performance Measurements

A. Map

The MPI version of the TBF concurrent mapping and localization algorithm was implemented on a 21 node Beowulf cluster. Ultrasound and odometry measurements were collected by hand from a Pioneer DX2 in a laboratory environment and used to evaluate the performance of the algorithm. The results below are for a 55x55 array.

(a)

(b)

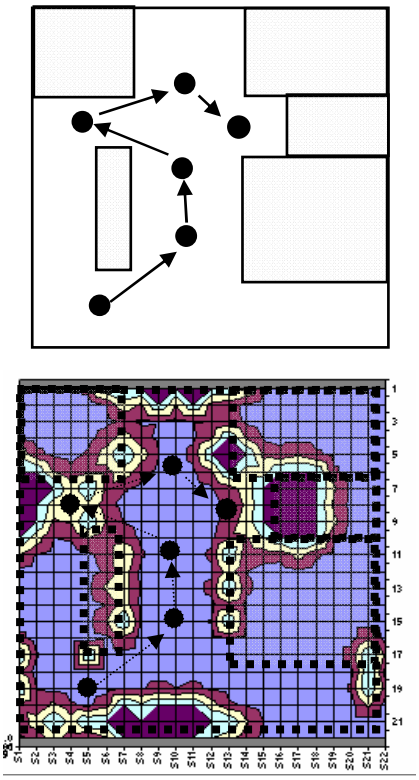


Figure 4: (a) Actual map and robot path
(b) probabilistic map after 5 iterations

B. Analysis of Parallel Performance

The MPI algorithm was run first on 1 processor of the Beowolf, then on 2, on 3 and so forth up to 21. A number of time measurements were made throughout the algorithm to gauge the computational demands. Figure 5 below shows the graph of total computation time. Notice that it initially falls,

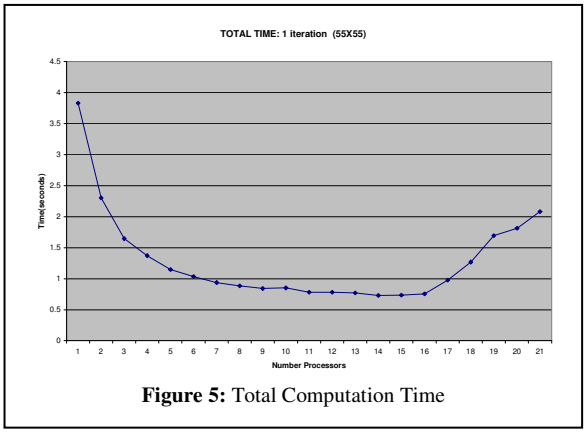


Figure 5: Total Computation Time

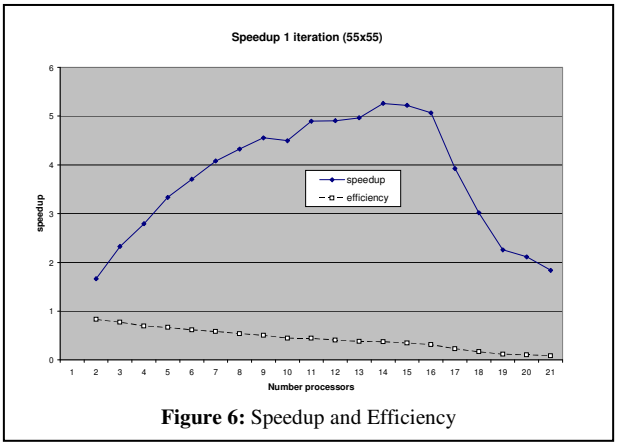


Figure 6: Speedup and Efficiency

in a dramatic fashion as the number of processors is increased. It then evens out, falling gently. At around 16 processors, the time clearly rises again.

This effect is due to the tradeoff between the increasing advantage of parallelism, and the increasing disadvantage of communication costs (the reduction and broadcast operations).

The *speedup* ψ of a parallel algorithm is defined as the sequential time T_s divided by the parallel time T_p when running in parallel on P processors. This is a measurement of the advantage gained by parallelism. The *efficiency* ε is defined as the speedup divided by the number of processors, and this is a measure of processor utilization:

$$\varepsilon = \frac{\psi}{P} = \frac{T_s}{T_p P}$$

Figure 6 shows these measures for one map iteration. Speedup improves initially - this is the *Amdahl effect*: computation complexity is typically higher than communication complexity, and hence addressing the computation with parallelism shows good returns initially. There is a peak speedup $\psi=5.26$ for 14 processors. Note however, that the efficiency falls off with the number of processors. At peak speedup, the efficiency is $\varepsilon = 0.38$. This phenomenon is typical for parallel algorithms [15]; however,

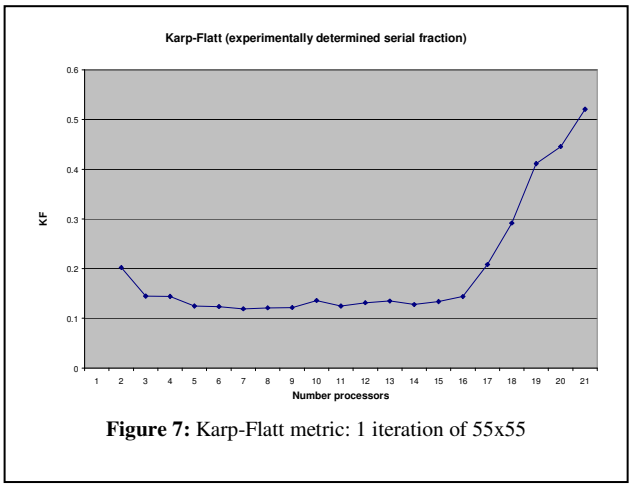
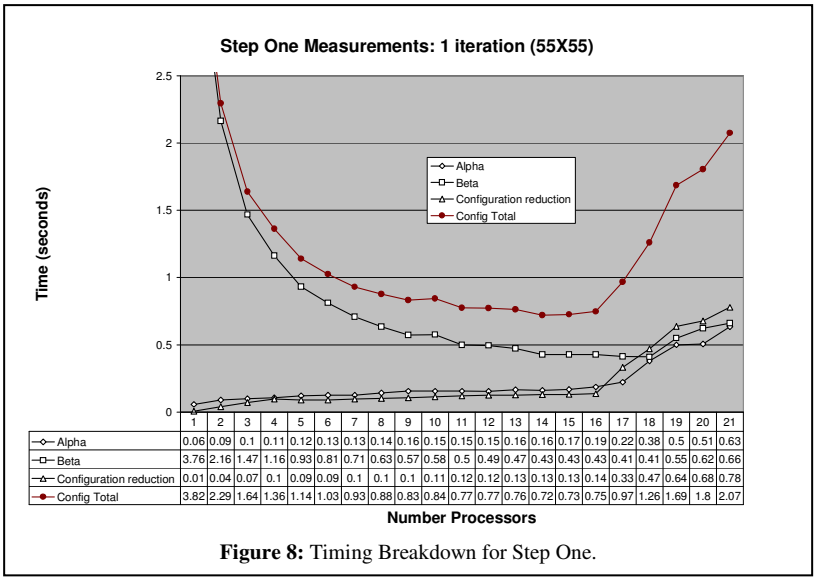


Figure 7: Karp-Flatt metric: 1 iteration of 55x55

it can be due to either an algorithm with limited opportunity for parallelism, or due to architectural or parallel overhead considerations. Clearly if the former is the case, then the architecture and algorithm we propose in Fig. 1 is not appropriate.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Alpha	0.06	0.09	0.1	0.11	0.12	0.13	0.13	0.14	0.16	0.15	0.15	0.15	0.16	0.16	0.17	0.19	0.22	0.38	0.5	0.51	0.63
Beta	3.76	2.16	1.47	1.16	0.93	0.81	0.71	0.63	0.57	0.58	0.5	0.49	0.47	0.43	0.43	0.43	0.41	0.41	0.55	0.62	0.66
Configuration reduction	0.01	0.04	0.07	0.1	0.09	0.09	0.1	0.1	0.1	0.11	0.12	0.12	0.13	0.13	0.13	0.14	0.33	0.47	0.64	0.68	0.78
Config Total	3.82	2.29	1.64	1.36	1.14	1.03	0.93	0.88	0.83	0.84	0.77	0.77	0.76	0.72	0.73	0.75	0.97	1.26	1.69	1.8	2.07

Figure 8: Timing Breakdown for Step One.

Figure 9: Timing Breakdown for Step two.

C. *Karp-Flatt Metric*

Whether an algorithm is a good candidate for parallelism can be judged by looking at the Karp-Flatt metric [8]. For a parallel algorithm with speed ψ on p processors, the *experimentally determined serial fraction* is defined as:

$$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$

This metric measures the parallel overhead experienced by an algorithm and we can analyze the performance of the MPI KPF implementation using it. After peak speedup, we noted that the speedup decreases quickly to very disappointing levels. We must ask the question whether this is due to an approach with limited opportunity for parallelism, or whether it is due to the increasing parallel or architectural overhead. If the value of e does not change significantly with increasing number of processors, then the reason for the disappointing speedup is that the algorithm has limited opportunity for parallelism (too much inherent sequentiality). If the value of e starts to rise with an increasing number of processors, then parallel overhead is clearly the culprit.

Fig. 7 shows the Karp-Flatt metric for one iteration of the algorithm on the 55x55 map size. The metric is constant for the period of increasing speedup, and increases dramatically for the period of decreasing speedup. The phenomenon shown in Fig. 6, therefore, is purely a function of parallel overhead, most likely increasing communication time.

D. *Breakdown of Measurements*

The graphs below show the breakdown of measurements between the calculation of alpha, beta, configuration reduction, and then map construction and reduction.

Fig. 8 shows the breakdown for step one, and Fig.9 the breakdown for step 2. In Fig. 8, note that the $C\beta$ calculation is the dominant component of total time up to the peak speedup. After peak speedup, the configuration reduction (mainly a communication operation) becomes dominant. This evidence explains the Karp-Flatt results in Fig. 7. Note also, that the $C\alpha$ calculation increases substantially after the peak speedup.

Fig. 8 shows the breakdown for step two. This has almost no effect on the total timing as it is an order of magnitude smaller than the step one times.

V. Conclusion

The goal of cluster technology, supercomputing performance at off-the-shelf prices, is one that is directly in tune with the current needs of robotics and computer vision. Inexpensive, high performance computing is one crucial factor in building robots that can sense and respond effectively to events in their new range of unstructured environments. The additional computing power can be used to process sonar and visual information more quickly or more thoroughly; it can be used to recognize and track additional features of the robot's environment; and it can be used to pick better action strategies, even considering those based on past performance and observations.

In this paper we introduce an architecture for incorporating cluster technology. We demonstrate a cluster implementation of a concurrent mapping and localization algorithm, and we analyze the timing and performance of the algorithm. We use timing results and parallel performance metrics to show that the architecture and algorithm can effectively exploit parallelism.

Our results are for one, relatively small map size and for a short path in an indoor room. The next step is to analyze a range of maps sizes and also path sizes. We chose the KBF algorithm for its implementation ease. However, we note that potentially faster approaches exist [13] and our goal is to implement and analyze one of these.

References

[1] Adluru, N., Latecki, L., Lakaemper, R., and Madhavan, R., Robot Mapping for Rescue Robots, *IEEE Int. Workshop on Safety, Security and Rescue Robotics*, NIST, Gaithersburg, August 2006.

[2] Alba, E., Luque, G., Evaluation of Parallel Metaheuristics. In: Proc. Workshop on *Empirical methods for the analysis of Algorithms,* Reykjavik, Iceland, Sept 2006.

[3] Arkin, R., and Mackensie, D., Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation. 1994 Int. Symp. on Robotics in Manufacturing. Maui, HI, Aug., 1994.

[4] Castellanos, J., Neira, J., amd Tardos, J., Multisensor Fusion for Simultaneous Localization and Map Building. *IEEE Trans Robt & Aut.* V17 N6 2001. pp. 908-914.

[5] Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M., "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. of Robotics and Automation*, 2001.

[6] DeSouza, G., and Kak, A., Vision for Mobile Robot Navigation: A Survey. *IEEE PAMI* V24, N2, Feb 2002. pp.237-267.

[7] Guazzelli, A., Corbacho, F., Bota, M., and Arbib, M., "Affordances, Motivation and World Graph Theory" *Adaptive Behavior* 6:435-471 1998.

[8] Karp, A., Flatt, H.: Measuring parallel processor performance. *Communications of the ACM* 33(5) pp.53-543, 1990.

[9] Lyons, D.M., Hsu, D.F., Ma, Q., and Wang, L., Selection of fusion operations using rank-score diversity for robot mapping and localization. To appear: *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications* at the SPIE Defense and Security Symposium, 9-13 April 2007, Orlando (Kissimmee), FL.

[10] Lyons, D.M. and Hendriks, A., Planning as Incremental Adaptation of a Reactive System. *Journal of Robotics & Autonomous Systems* 14, 1995, pp.255-288.

[11] López de Teruel, P., García J., and Acacio, M. The Parallel EM Algorithm and its Applications in Computer Vision. *Int. Conf. Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 1999.

[12] Merkey, P., Beowulf History. www.beowulf.org.

[13] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B., FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, Proc. AAI Nat. Conf. on AI., Edmonton, Canada, 2002.

[14] Mundhenk, T., Ackerman, C., Chung, D., Dhavale, N., Hudson, B., Hirata, R., Pichon, E., Shi, Z., Tsui, A., and Itti, L.. Low-cost high-performance mobile robot design utilizing off-the-shelf parts and the Beowulf concept: the Beobot project. *SPIE Conference on Intelligent Robots and Computer Vision*, V5267, October 2003, pp. 293-303.

[15] Quinn, M., Parallel Programming in C with MPI and OpenMP. McGraw Hill, 2003.

[16] Thrun, S., Burgard, W., and Fox, D., A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning and Aut. Robots* 31/5 1998, pp. 1-25.

[17] Thrun, S., Robot mapping: A Survey. Technical report CMU-CS-02-111, School of Computer Science, Carnegie-Mellon University, Pittsburgh PA Feb. 2002. In: *Exploring Artificial Intelligence in the New Millenium*, (Eds. Lakemeyer, G. and Nebel, B.) Morgan Kaufmann 2002.