Hardware Efficient Parallel Particle Filter for Tracking in Wireless Networks

YiQiao Zhang[†], Thuraiappah Sathyan[‡], Mark Hedley[‡], Philip H.W. Leong[†], Ahmed Pasha[†]
[†]School of Electrical and Information Engineering, The University of Sydney, NSW, Australia
[‡]Commonwealth Scientific and Industrial Research Organisation (CSIRO), Marsfield, NSW, Australia
yzha6371@uni.sydney.edu.au, tsathyan@ieee.org, mark.hedley@csiro.au, philip.leong@sydney.edu.au, ahmed.pasha@sydney.edu.au

***Abstract*—Location tracking is being increasingly used across many applications. While GPS is the most widely used location tracking technology, it is unavailable in many environments such as indoors and underground. Local positioning systems (LPS) that use time of arrival based ranging can provide high accuracy location tracking for many applications. Tracking location using range measurements is a non-linear state estimation problem and the measurement noise is often non-Gaussian in environments where LPS are typically used. Hence a particle filter is an appropriate state estimator for location tracking in LPS.**

Particle filters are computationally complex and have a serial bottleneck that prevents straightforward parallel implementation. In this paper we present a parallel architecture for the particle filter that can be efficiently implemented in a field programmable gate array or a fixed-point digital signal processor. We show that processing can be divided into up to twenty parallel particle filters to massively increase the processing speed. Mixing between the filters is essential and we present a new algorithm for this that minimises computational complexity and memory bandwidth. Finally, for efficient hardware implementation fixed point arithmetic should be used and we empirically determine the required precision.

***Index Terms*—Wireless Networks, Location Tracking, Time of Arrival, Particle filtering, FPGA, Hardware architecture.**

I. Introduction

Location awareness is a key feature requirement in many applications. For example, high accuracy tracking of athletes in both outdoor and indoor sports allows sports scientists and coaches to develop new training and gaming strategies. Accurately tracking vehicles and personnel in underground mines improves the automation and safety of the mine.

The Global Positioning System (GPS) is the most widely used localisation system primarily due to its global coverage and free access. However, satellite based positioning systems cannot provide tracking underground and in most indoor areas, hence there is a need for Local Positioning Systems (LPS). There are a number of technologies that have been used for a LPS, and time of arrival (TOA) based tracking is widely agreed to provide the most accurate tracking [7].

A TOA-based LPS typically consists of two types of nodes: anchor nodes, whose locations are assumed known a priori; and mobile nodes, whose locations are required to be estimated. The range (or pseudo-range) between nodes is calculated based on the measured TOA of radio signals transmitted between the nodes. When a mobile node has range measurements to multiple anchor nodes (at least three for two-dimensional localisation) the location of the mobile node can be calculated, and temporal tracking is typically used to reduce measurement noise. The use of a non-linear filter, such as a particle filter, is required to handle the non-Gaussian range error distribution encountered in indoor and other environments where multipath is encountered [8].

Particle filters are sequential Monte Carlo techniques that approximate the optimal Bayesian recursion using a point mass representation of the posterior densities [2]. This representation consists of a set of random samples (particles) and associated weights. Although the particle filter has been shown to perform well for state estimation problems, it typically requires a large number of particles, which leads to high computational cost, particularly in applications where a high update rate is required.

Particle filters apply simple operations to a large number of particles (typically several hundreds or thousands), and most of these operations can be performed independently, hence parallel processing can be effectively exploited. Further these operations can typically be performed using low precision arithmetic. These properties make particle filters ideally suited for implementation in field programmable gate arrays (FPGA), multicore digital signal processors (DSP) and single instruction multiple data (SIMD) processors such as the Sony Cell. In particular, modern FPGA devices can have thousands of dedicated multiply units and can support trillions of integer operations per second in a single package at power levels similar to a single processor in a personal computer. Unfortunately there is a serial bottleneck in conventional particle filter algorithms. The goal of our work is to develop algorithms that overcome this bottleneck and allow efficient implementation on low precision parallel computing platforms.

Parallel implementations of particle filters have been considered in the literature to improve execution of time [3–6]. A typical implementation of the particle filter consists of sampling, weight update, and resampling steps, all of which are amenable to parallel implementation on their own. The weights, which approximate the probability measure, however, are normalised before executing the resampling step, which introduces a serial bottleneck in the particle filter.

The literature on hardware implementation of particle filters primarily considers improvements to the resampling algorithm to improve execution time, and architectures to reduce the memory requirement. For example, [5] considered modifications to the resampling algorithm that avoided normalisation of the weights. This, however, entails non-deterministic and complicated data exchange patterns between processing elements and the control unit. In [3] architectures that reduce memory usage using a single dual-port memory were proposed, along with modifications to the resampling algorithm to improve the execution time. Effects of using finite precision implementation on performance were analysed in [4]. A parallel hardware implementation of particle filters was considered in [6], where distributed resampling algorithms with proportional and non-proportional allocation of particles were proposed.

The parallel particle filter architecture that we consider in this paper is the same as the distributed non-proportional allocation technique of [6]. In this technique the particles are divided into a number of groups with each group consisting of a fixed number of particles and the standard operations of the particle filter are performed in each group independently. Note that if no data are exchanged between the groups, we cannot expect the performance of the overall particle filter to be any better than that of a single group. Whereas in [6] a deterministic data exchange technique was considered, we propose four different data exchange techniques and compare their performance. We also discuss implications of the different data exchange techniques for hardware implementation in terms of memory utilisation and execution time.

The novel aspects of the work presented in this paper are: (i) explore the extent to which the tracking problem can be parallelized; (ii) develop a novel algorithm for mixing particles between the parallel filters that is efficient for fixed-point DSP and FPGA implementation; and (iii) determine the minimum precision arithmetic required to minimize the implementation cost.

This paper is organised as follows. Section II describes mobile node state estimation in wireless networks using the standard sequential importance sampling resampling particle filter and explains the serial bottleneck present in its implementation. Section III describes the architecture that we propose in this paper, along with simulation results comparing several mixing algorithms that we consider. Section IV discusses the hardware implementation issues, and Section V provides concluding remarks.

II. Sequential Particle filter for State Estimation of A Mobile Node

State estimation is formulated as an inference problem on an appropriately defined state-space model. The state transition model represents our prior knowledge of the motion of the mobile node and the measurement model relates the state of the node to the measurement.

A. State Model

The state vector for our application consists of the location and velocity of a mobile node, as we assume that the state transition of the mobile node is adequately represented by a single nearly constant velocity (NCV) model. Let $\mathbf{x}(\mathrm{k}) = [x(k), y(k), \dot{x}(k), \dot{y}(k)]^T$ denote the state of the mobile node at time k, where $(x(k), y(k))$ denotes the position and $(\dot{x}(k), \dot{y}(k))$ denotes the velocity components. The state transition model can then be written as

$$\mathbf{x}(k) = F\mathbf{x}(k-1) + \mathbf{v}(k) \qquad (1)$$

where F is the state transition matrix and $\mathbf{v}(k)$ is the process noise, which is assumed to be a zero-mean Gaussian random variable with covariance matrix Q. For the NCV model we can write F and Q as [11]

$$F = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Q = q\begin{bmatrix} \frac{T^3}{3} & 0 & \frac{T^2}{2} & 0 \\ 0 & \frac{T^3}{3} & 0 & \frac{T^2}{2} \\ \frac{T^2}{2} & 0 & T & 0 \\ 0 & \frac{T^2}{2} & 0 & T \end{bmatrix} \qquad (2)$$

where T is the sample period and q denotes the power spectral density of the process noise.

B. Measurement Model

A mobile node measures the range to all the anchor nodes that are within its communication range. The range measurement between the i^{th} anchor node and the mobile node at $\mathbf{x}(k)$ is given by

$$z_i(k) = h_i(\mathbf{x}(k)) + \upsilon_i(k). \qquad (3)$$

The set of all measurements to anchor nodes can be written in vector form as

$$\mathbf{z}(k) = h(\mathbf{x}(k)) + \boldsymbol{\upsilon}(k) \qquad (4)$$

where $(x_i(k), y_i(k))$ is the location of the ith anchor node, $\upsilon_i(k)$ denotes the measurement noise and the non-linear mapping function is given by

$$h_i(\mathbf{x}(k)) = \sqrt{(x(k) - x_i)^2 + (y(k) - y_i)^2}. \qquad (5)$$

Note that the measurement equation is non-linear. Hence, even if the state transition model is linear, a non-linear filtering algorithm such as an extension of the Kalman filter (e.g., extended Kalman filter (EKF) and unscented Kalman filter (UKF)) or a particle filter is required for state estimation. The particle filter is the preferred choice since the range measurement noise can be non-Gaussian. This can be seen in Figure 1, which plots the outdoor and indoor range error distributions measured using our wireless ad-hoc system for positioning (WASP) [9]. The outdoor error distribution can be adequately modelled using a zero-mean Gaussian distribution (in this example, with mean 0.12 m), the indoor distribution is clearly non-Gaussian. This distribution is clearly asymmetric and biased.

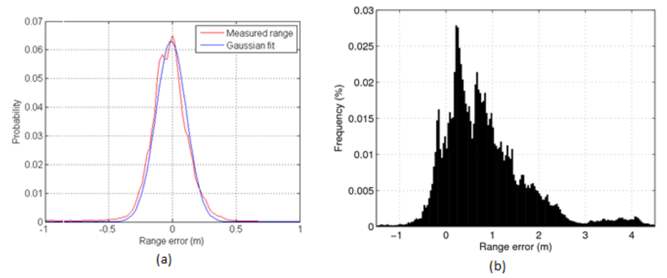

Figure 1. Range measurement noise characteristics in (a) outdoor and (b) indoor environments. Outdoor error can be approximated well with a Gaussian distribution, whereas the indoor distribution is symmetric and biased.

Like the Kalman filter, the EKF and UKF approximate the posterior density using a Gaussian distribution and propagate the first two moments over time. The particle filter on the other hand propagates samples of the posterior state distribution. Although we have found that the UKF can perform as well as the particle filter with outdoor data sets, its performance degraded severely with indoor data sets, where the performance of the particle filter is superior. Hence, from a system implementation perspective, particle filter is preferred for both outdoor and indoor tracking of nodes.

C. *Particle Filter for State Estimation*

Let $\{\mathbf{x}^i(k-1), w^i(k-1)\}_{i=1}^N$ denote the N particles and their corresponding weights that represent the posterior density at time $k-1$. With the availability of the range measurements at time k, the particle filter approximates the posterior density with a new set of particle-weight pairs $\{\mathbf{x}^i(k), w^i(k)\}_{i=1}^N$, through sampling, weight update, and resampling steps [2].

1) *Sampling*

New samples at time k are drawn from the importance density $q(.)$ according to

$$\mathbf{x}^{\mathrm{i}}(k) \sim q(x(k)|x^i(k-1), \mathbf{z}(1{:}k)) \tag{6}$$

The importance density that is most often used is the prior because it is easy to implement and intuitive. That is

$$q\left(\mathbf{x}(k)\middle|\mathbf{x}^i(k-1), \mathbf{z}(1{:}k)\right) = p\left(\mathbf{x}(k)\middle|\mathbf{x}^i(k-1)\right) \tag{7}$$

2) *Weight Update*

When the prior density is used as the posterior density it can be shown that the updated weight is given by

$$w^i(k) = w^i(k-1)p(\mathbf{z}(k)|\mathbf{x}^i(k)) \tag{8}$$

where $p(\mathbf{z}(k)|\mathbf{x}^i(k))$ is the measurement likelihood. Note that if resampling is used at every time step then the prior weights will all be equal to $1/N$, and hence, the updated weight is proportional to the likelihood.

3) *Resampling*

The sampling and weight update are repeated every time new measurements become available. After a few iterations the variance of the particles will increase such that there are only a few particles with significant weights. This phenomenon is referred to as sample degeneracy and will lead to the divergence of the particle filter. The objective of the resampling step is to remove the particles with insignificant weights and to generate a new particle set by concentrating on the particles with significant weights.

In the resampling step new samples $\{\mathbf{x}^{i*}(k)\}_{i*=1}^N$ are drawn from the newly proposed particles $\{\mathbf{x}^i(k)\}_{i=1}^N$. New samples are drawn randomly according to their probabilities P^i given by

$$P^i(k) = \frac{w^i(k)}{\sum_{l=1}^N w^l(k)} \tag{9}$$

The resampled particles are independent and identically distributed samples of a discrete density and hence their weights are set to $w^i(k) = 1/N$.

III. Parallel Particle Filter

The particle filter is often implemented as a sequential process. A parallel implementation of the particle filter can achieve significant improvement in execution time. Although the sampling and weight update states can be implemented in parallel, in order to calculate the probabilities of the particles used in the resampling step, calculated according to (9), requires the weights of all the particles to be known. This creates a serial bottleneck where each parallel path must wait for the others to complete before the resampling step can begin. In this section we look at different ways to overcome this serial bottleneck. Note that the techniques that we look at are approximations to the original serial implementation and the objective is to find a parallel implementation that is efficient in memory usage and complexity, while not sacrificing tracking performance.

A. *Structure of the Parallel Implementation*

In the parallel particle filter implementation we consider particles are divided in to M groups with each consisting N/M particles. The sampling, weight update, and resampling steps for each group are implemented as in the standard serial implementation of the particle filter. One can think of just combining the individual state estimates of different groups to form a combined state estimate. The performance of such a filter cannot be any better than that obtained by running a single filter with N/M particles.

This necessitates a mixing step, where particles from different groups are mixed prior to processing the measurements from the next time step. The mixing step can be performed before or after the resampling step in each group and the final estimate of the parallel particle filter is the combined estimate of all the groups after the mixing step. Although the particle filter running in each group still has the serial bottleneck, by reducing the number of particles by a factor of M the speed of each particle filter can be increased by the same factor M. It is important that the new step, mixing between groups, does not introduce a new serial bottleneck.

We will now present four different mixing strategies in this section then compare their performance.

1) *Full Random Mixing (FRM)*

In this case we as the name suggests the mixing is performed in a random order. After performing the resampling step in each group we generate a random permutation of the numbers between 1 and N and swap the memory locations of all the N particles according to the random permutation prior to processing the next set of measurements.

This technique follows the spirit of the particle filter, i.e., operations are performed randomly. As a result the mixed particles will show good diversity and will lead to better performance. This is the most complex to implement in hardware.

2) *Lowest Weight Replacement (LWR)*

Mixing is implemented before the resampling step and uses the weights as a guide to perform mixing. After performing weight update, particles in each group are sorted according to their weights. The highest weighted particle in each group is then selected and the least weighted $M-1$ particles in each group are replaced by the highest weighted particle in the other $M-1$ groups. An example of LWR technique is shown in Figure 2.

This technique requires additional processing to find the particle with the largest weight, but has the lowest memory bandwidth requirement of the proposed techniques.

3) *Deterministic Mixing (DM)*

Mixing is performed in a predetermined order after the resampling step, at which point all particles have equal weight. There are many ways in which DM can be done. In the technique we implemented each group is numbered sequentially and groups with consecutive numbers are considered adjacent. We also consider the last and first groups to be adjacent. Two particles from each group are swapped with the correspondingly numbered particles in the adjacent group. The DM technique that was implemented in the simulations is illustrated in Figure 3.

This type of mixing is easy to implement due to its deterministic nature and it is easy to devise optimized hardware implementations. The memory bandwidth for each group is low and independent of the number of groups.

4) *Partial Random Mixing (PRM)*

This is similar to FRM, however to reduce memory bandwidth we only mix a randomly selected subset of particles from adjacent groups. This is performed after resampling. While enjoying simpler complexity compared to the full random mixing, this still provides benefits of randomness.

B. Simulation Results

We conducted a simulation study to compare the performance of the parallel particle filter implemented with the different mixing techniques considered. The benchmark performance was considered to be the one obtained using the standard serial implementation.

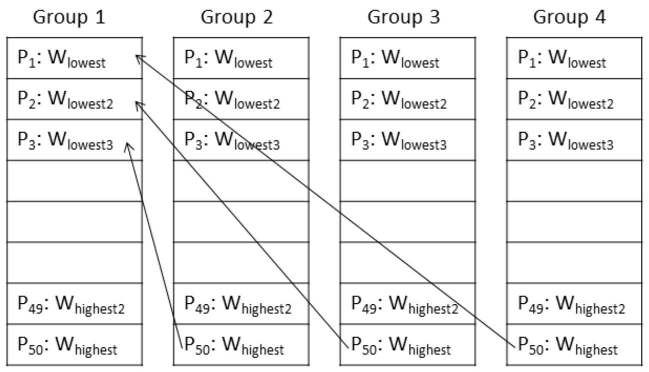

Figure 2. Illustration of lowest weight replacement mixing. There are four groups of 50 particles each and the highest weighted particles from groups two to four replaces the lowest weighted three particles from group 1.

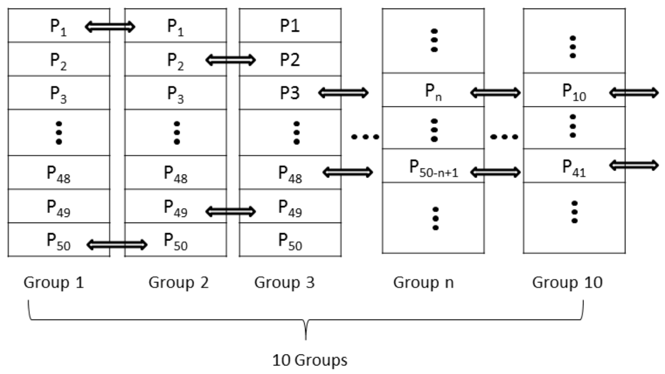

Figure 3. Illustration of deterministic mixing. There are ten groups of 50 particles each and two particles from a group are swapped with two particles in the adjacent group. First and last groups are assumed adjacent.

In the scenario considered there were B=4 anchor nodes located at the edges of a 100×100 m square. A single mobile node was simulated whose initial position and velocity were set to (4, 3.5) m and (0.25, 0.15) m/s, respectively. The mobile node trajectory was generated using a NCV model with the power spectral density of the process noise set to 0.05 m^2/s^3. The range between the anchor nodes and the mobile node was determined at each time step and the measurement noise was generated from a zero-mean Gaussian distribution with standard deviation 0.5 m. The simulation duration was 80 s.

A total of 500 particles were used in all implementations of the particle filter. Through experiments we found this is the minimum number of particles that is required in the standard serial implementation to provide non-diverging tracks of the mobile node. By using the minimum number of particles of the serial implementation, any degradation of the performance in the parallel implementations would immediately be observable.

Figures 4-7 show the root mean square error (RMSE) of the position and velocity of the mobile node calculated over 50 Monte Carlo runs for the four mixing techniques discussed in this section. These plots compare the performance of two parallel particle filter implementations with 10 and 20 groups with that of a non-parallel implementation (i.e. one group).

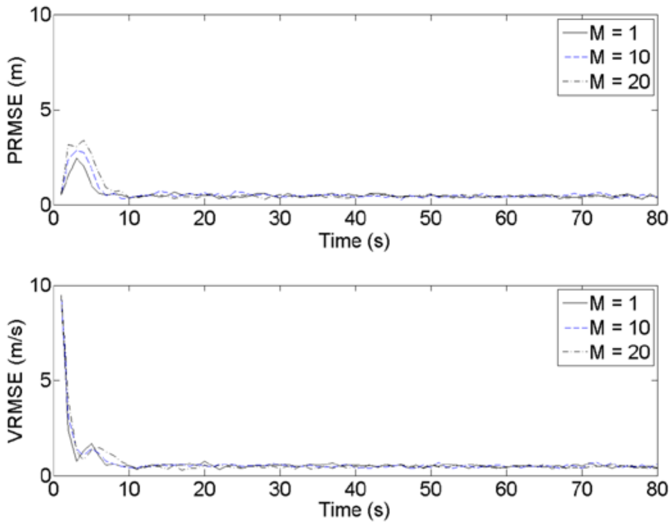

Figure 4. Full random mixing.

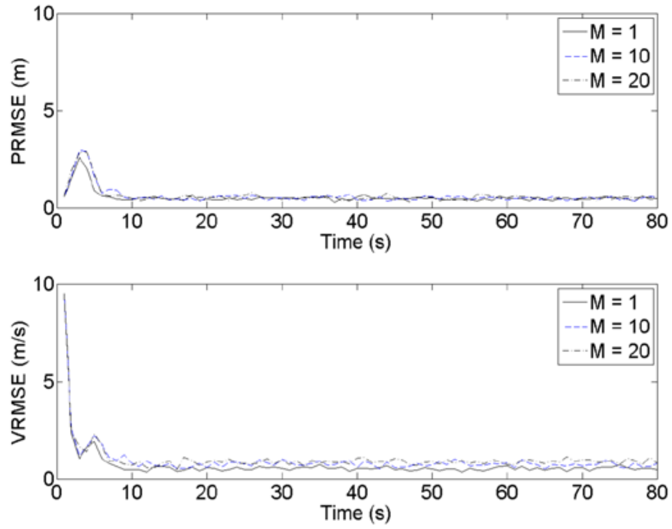

Figure 5. Lowest weight replacement.

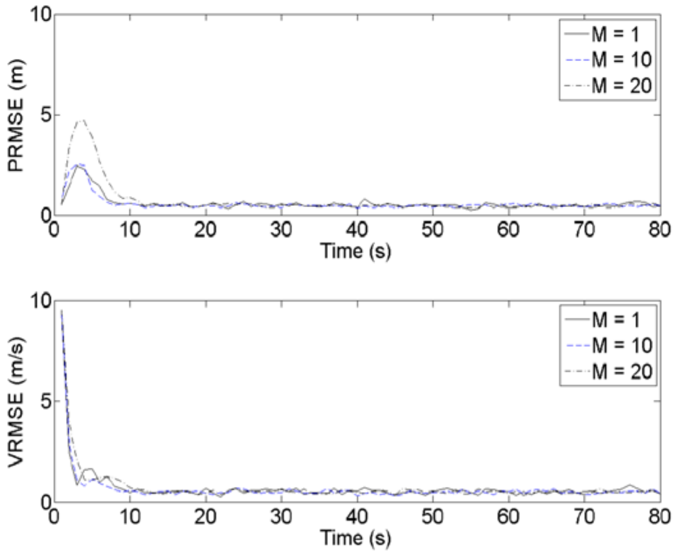

Figure 6. Deterministic mixing.

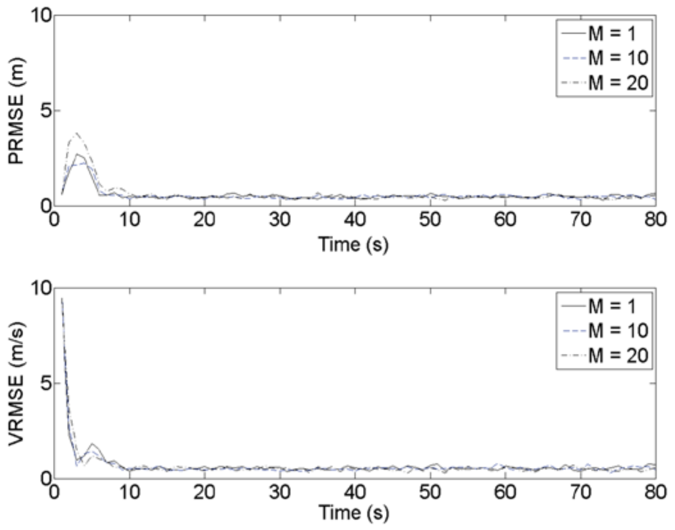

Figure 7. Partial random mixing.

TABLE 1
COMPARISON OF MIXING TECHNIQUES PER GROUP

Technique	Memory Accesses	Extra Processing
FRM	2*N*/*M*	random numbers
LWR	*M*+1	largest number
DM	8	-
PRM	20	random numbers

C. *Analysis of Results*

The first important observation from the data is that it is possible to have up to 20 parallel filters and still provide good tracking performance, hence a speedup of almost twenty times can be achieved.

Table 1 shows the number of memory accesses required for each technique, where for this data $N = 500$ and M has values 10 and 20. The accesses are for each group, as an implementation will require a separate memory for each group. The least number of memory accesses is required by DM and the most by FRM. For high levels of parallelization LWR requires more memory accesses than PRM.

From the results it is seen that DM has the worst performance, and FRM has the best performance, as expected from the amount of mixing performed. PRM generally performed better than LWR and does not require an extra processing step to find the largest number. Hence it is the preferred technique among other mixing strategies and was finally chosen to be evaluated in hardware.

IV. FILTER IMPLEMENTATION

The previous section described the mixing operation required between multiple particle filters. In this section we focus on the implementation of the particle filter in each group.

A breakdown of the arithmetic operations required per iteration of a particle filter is given in Table 2, where N is the total number of particles, and B is the number of base stations. The weight update step is seen to dominate the computational complexity.

TABLE 2
BREAKDOWN OF OPERATIONS OF EACH STEP IN PARTICLE FILTER

Operations / Step	Add Subtract	Multiply	Divide	Exp	Sqrt	Random numbers
Sampling	$6N$	$2N$	-	-	-	$4N$
Weight	$4BN$	$2(B+2)N$	N	N	BN	-
Resampling	$3N$	-	-	-	-	N

Besides the basic operations (add, subtract, multiply), the implementation also requires division (div), exponential (exp), and square root (sqrt). Standard integer div and sqrt algorithms are used since they are reasonably area efficient and produce exact results. The exponential function is computed using the CORDIC algorithm in rotation mode [1,4]. Moreover, four independent, Gaussian distributed random numbers are required per iteration. These can be efficiently generated in software or hardware using the Ziggurat method [10].

Although floating-point arithmetic is convenient for simulations on desktop computers, fixed-point designs are superior in terms of silicon area, power consumption, speed and latency. As our target implementation platforms are FPGAs and low-cost DSPs, fixed-point is preferred.

A study of the effects of precision on accuracy was undertaken. All arithmetic operations were implemented using the MATLAB Fixed-Point Toolbox, with the exception of the exp function, which used CORDIC. The signal was represented as an integer and its word length was varied in order to observe the effects of finite precision. Simulations using a serial configuration of the particle filter at different precisions were made and a double-precision accumulator was used to calculate the position root mean square error (PRMSE) of the mobile node. The results are shown in Figure 8. It can be seen that 12-bit fractional precision has significant error whereas 18-bit or higher precision results in similar accuracy to a double precision implementation.

V. CONCLUSIONS

A good choice for tracking in wireless networks in GPS denied spaces is range based localization using a particle filter. Particle filters are computationally complex and have a serial bottleneck. In this paper we described a parallel particle filter design suitable for implementation in an FPGA or parallel fixed-point DSP. The serial bottleneck is overcome by dividing the particles between multiple particle filters operating in parallel, and performing a mixing operation between the filters at each time step. We showed that for tracking up to twenty parallel particle filters works well, hence, allowing a twenty-fold speedup. We evaluated four techniques for particle mixing and found that partial random mixing achieves good performance while significantly reducing memory bandwidth.

We analysed the operations required in each particle filter and proposed a fixed point implementation that leads to efficient implementation with almost no loss in accuracy compared to a double precision floating point implementation.

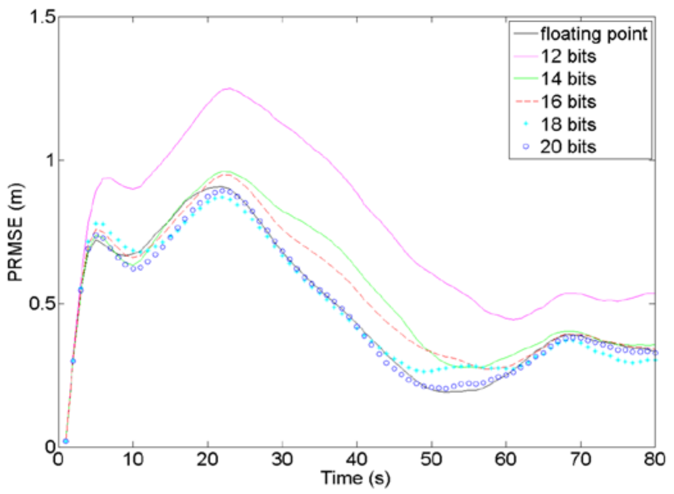

Figure 8. Effect of precision on location error (M=10).

Our future work will focus on FPGA implementations, which combine the task-level parallelisation techniques described in this paper with other hardware optimisations such as pipelining, parallel arithmetic units and multi-ported memories. Other techniques to further reduce precision requirements while maintaining high accuracy will also be investigated.

REFERENCES

[1] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers", *Proc. ACM/SIGDA Symp. on FPGAs,* 1998, pp. 191-200.

[2] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking", *IEEE Trans. on Signal Processing,* Vo. 50, No. 2, pp. 174 – 187, 2002.

[3] A. Athalye, M. Bolic, S. Hong, and P. M. Djuric, "Generic Hardware Architectures for Sampling and Resampling in Particle Filters", *EURASIP Journal on Applied Signal Processing*, Vo. 17, pp. 2888 – 2902, 2005.

[4] M. Bolic, S. Hong, and P. M. Djuric, "Finite Precision Effect on Performance and Complexity of Particle Filters for Bearing-Only Tracking", *Signals, Systems and Computers*, Vol.1, pp. 838 – 842, 2002.

[5] M. Bolic, A. Athalye, P. M. Djuric, S. Hong, "Algorithmic Modification of Particle Filters for Hardware Implementation," *Proc. European Signal Processing Conference*, Vienna, Austria, 2004.

[6] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Processing,* Vol. 53, No. 7, pp. 2442 – 2450, 2005.

[7] K. Pahlavan, X. Li, and J.-P. Makela, "Indoor geolocation science and technology", *IEEE Communications Mag.* Vol. 40, No. 2, 112–118, 2002.

[8] T. Sathyan and M. Hedley, "Efficient Particle Filtering for Tracking Maneuvering Objects," *Proc. IEEE/ION Position Location and Navigation Symp.*, Indian Wells, CA, May 2010.

[9] T. Sathyan, D. Humphrey, and M. Hedley, "A System and Algorithms for Accurate Radio Localization using Low-cost Hardware," *IEEE Trans. System, Man and Cybernetics – Part C,* Vol. 41, No. 2, pp. 211-222, 2011.

[10] D.B. Thomas, W. Luk, P.H.W. Leong and J.D. Villasenor, "*Gaussian random number generators*," ACM Computing Surveys, vol. 39, no. 4, pp. 11.1-11.38, October 2007.

[11] Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons, New York, NY, 2001.