Hardware/software co-design of particle filter in grid based Fast-SLAM algorithm

B.G. Sileshi¹, C.Ferrer², J.Oliver³

¹,³Departament de Microelecònica i Sistemes Electrònics, Universitat Autònoma de Barcelona
Bellaterra (Barcelona), Spain
²Institut de Microelecònica de Barcelona (CNM-CSIC)
Bellaterra (Barcelona), Spain

Abstract-A hardware/software co-design based on system on a chip method for particle filter in a grid based Fast-SLAM algorithm is presented in this work. By giving more emphasis on those steps of the algorithm that requires intensive computations, hardware blocks are designed in order to speed up the computational time and interfaced with a central Microblaze soft core processing core. The proposed hardware/software resulted in an improvement in the overall execution time of the algorithm.

Keywords: Particle filters, Hardware/software co-design, computational complexity.

Conference Name: ESA

1. Introduction

Particle filters (PFs) are among the estimation frameworks that offer superior flexibility on addressing non-linear and non-Gaussian estimation problems [1]. As a result they have been applied to a wide variety of real world problems such as navigation and positioning [2-6], tracking [7-8] and robotics[9]. Their flexibility comes from their efficient representation of a wide range of probability densities by sets of points (particles). Such representational power of particle filters, however, comes at the cost of higher computational complexity that has so far limited their applications in different types of real time problems.

To achieve real time performance in the application of particle filters, hardware or hardware/software based implementation is required to accelerate the computational time. In the literature several studies tackles the real time constrains of the particle filter through hardware implementations. In this sense, FPGA hardware platforms are considered to be one of the choices for such implementations of the particle filter due to the possibility to introduce massive parallelization and their low power consumption properties which is critical for many application such as navigation.

Many studies have been proposed for the hardware implementations of particle filtering on the FPGA platforms. The authors in [10] studied the implementation of particle filter on an FPGA for a mobile robot localization problem, where adapting the size of the particle set is used in reducing the run-time computation complexity of the algorithm. In [11] the FPGA hardware implementation of particle filter for location estimation is presented and compared with the software solution running on an ARM7-based microcontroller. A recent study in [12] presented a heterogeneous reconfigurable system consisting of an FPGA and CPU for a simultaneous mobile robot localization and people tracking application. In this study they propose a method to adapt the number of particles dynamically and utilize the run-time reconfigurability of the FPGA for reduced power and energy consumption. From their study up 7.39 times speed up is achieved compared to the Intel Xeon X5650 CPU software implementation. The authors in [13] presented a hardware/software co-design approach based on system on a chip technique on a NIOS II processor to calculate the weight for each particle and a hardware implementation to update the particles. They claim that their proposed method significantly improved the efficiency and it also provides flexibility in design for various applications due to the software implementation of the importance weight step.

The study in [14] presented the hardware architecture for an FPGA implementation of the sampling importance resampling (SIR) particle filter. The hardware architecture is simulated with Modelsim and the real time performance of the hardware architecture is also evaluated with use of UART for the input sensor data. In [15] a System-on-Chip architecture involving both hardware and software components for a class of particle filters is presented. In this work parameterized framework is used to enable the reuse of the architecture with minimal re-design effort for a wide range of particle filtering applications. In reference [16] three different hardware architectures are proposed and suggested the use of a piecewise linear function instead of the classical exponential function to in the decrease the complexity of the hardware architecture in the weighting step.

With the objective in solving the high computational requirements of the particle filter, this work presented a hardware/software co-design approach for the implementation of the particle filter algorithm with the following specific contributions:

- As the underlying computations in particle filtering vary from one application to the other, a generic approach is presented for the analysis, design and speedup of the particle
filter computational steps. The same approach can be used in developing other application specific particle filtering implementations.

- An embedded hardware/software implementation of the particle filter in grid-based Fast SLAM algorithm is presented.

- Fast hardware Gaussian random number generator design and its application in particle filter for grid based Fast SLAM algorithm is presented.

- The design and implementation of custom CORDIC hardware module for the evaluation of the complex mathematical operations involved in each step of the particle filter algorithm is presented.

The organization of the rest of the paper is as follows: Section 2 introduces briefly the SIRF particle filter and its application in specific to SLAM algorithm called Fast SLAM. Section 3 is a discussion on the identification of the computational bottlenecks of the particle filter algorithm and the partitioning of the different steps for hardware and software implementations. Section 4 is dedicated to the discussion on the hardware architecture design of the hardware partitions. Section 5 is a discussion on the proposed hardware/software architecture of the particle filter followed by section 6 with explanations on the implantation results. Finally section 7 concludes the paper.

2. Particle filter and particle filter based SLAM algorithms

Simultaneous localization and mapping (SLAM) is the problem of localizing and building a map of a given environment simultaneously. It is usually described with a joint posterior probability density distribution (equation 1) of the map \( m \) and the robot states \( x_t \) at time \( t \) given the observations \( z_{0:t} \) and control inputs \( u_{0:t} \) up to and including time \( t \) together with the initial state of the robot \( x_0 \).

\[
p(x_t, m | z_{0:t}, u_{0:t}, x_0) \tag{1}
\]

For the computation of the conditional probability given by equation 1 for all times \( t \), a recursive solution based on the Bayes Theorem is used in SLAM by starting with an estimate for the distribution \( p(x_t-1, m | z_{0:t-1}, u_{0:t-1}) \) at time \( t - 1 \) and following a control \( u_t \) and observation \( z_t \). Therefore, the SLAM algorithm is implemented with the two standard recursive time update and measurement update steps. Where the recursion is a function of the robot motion model \( p(x_t | x_{t-1}, u_t) \) and an observation model \( p(z_t | x_t, m) \).

\[
\text{Time Update:} \quad p(x_t, m | z_{0:t-1}, u_{0:t-1}, x_0) = \int p(x_t | x_{t-1}, u_t)p(x_{t-1}, m | z_{0:t-1}, u_{0:t-1}, x_0) dx_{t-1} \tag{2}
\]

\[
\text{Measurements Update} \quad p(x_t, m | z_{0:t}, u_{0:t}, x_0) = \frac{p(z_t | x_t, m)p(x_t, m | z_{0:t-1}, u_{0:t-1}, x_0)}{p(z_t | z_{0:t-1}, u_{0:t})} \tag{3}
\]

Among the very few methods that use the particle filter for the whole SLAM problem is the grid-based Fast SLAM [17] algorithm, which is used in this work. As the particle filter is the core of this algorithm and the main focus of this work brief discussion is provided as follows.

In generic Sequential Importance Sampling (SIS) particle filtering method a probability density function is approximated by a discrete set of particles or samples with their associated weights \( \{x^i_t, w^i_t\}_{i=1}^N \). Where, \( i \) and \( N \) denotes the index of particle and total number of particles respectively. \( x^i_t \in \mathbb{R}^{d_x} \) and \( w^i_t \) denotes the state of the particles and their weights respectively and \( d_x \) is the dimension of the state. One of the most widely adopted variant of the SIS particle filtering is the Sampling Importance Resampling particle filter (SIRF) where it recursively propagation of the particle set \( S_t = \{x^i_t, w^i_t\}_{i=1}^N \) at time \( t \), from the set \( S_{t-1} \) at the previous time \( t - 1 \) through sampling, importance weight and resampling steps.

\[
\text{Sampling} \quad \text{In this step new particles are drawn from the prior density} \quad p(x^i_t | x^i_{t-1}) \text{ which is defined by the system equation} \quad x_t = f_t(x_{t-1}, v_{t-1}) \quad \text{where} \quad f_t \text{ is possibly non-linear function of the state} \quad x_{t-1} \text{ and process noise} \quad v_{t-1} \text{ at time} \quad t - 1. \tag{4}
\]

\[
\text{Importance Weight} \quad \text{In this step a weight is assigned to each particle according to the measurement} \quad z_t . \quad w^i_t \propto (w^i_{t-1})p(z_t | x^i_t) \tag{5}
\]

where, \( p(z_t | x^i_t) \) is the measurement likelihood.
Resampling

While the sampling and Importance weight steps are performed at every time step \( t \) when new measurements are available, particles may become degenerate (increase in variance of the particles) after few iterations, the drawback of SIS particle filtering. The resampling step overcomes this problem by replicating particles with large weights and discarding those with small weights.

In the resampling step of the particle filter, different resampling algorithms [18] can be used as they are non application specific algorithms. The specific resampling method adopted in this study is Independent Metropolis Hastings (IMHA) resampling algorithm. This specific resampling algorithm has the lowest computational complexity compared to most conventional resampling schemes like systematic resampling [19]. As a result of such interesting characteristics it is the preferred choice in this work.

3. Computational bottlenecks identification and hardware/software partitioning

In the speedup of the overall computation in particle filters, a preliminary study in the identification of the critical bottlenecks in the algorithm is necessary for the design of hardware modules to accelerate those computational bottleneck steps. Our study is based on the particle filter in a grid based Fast SLAM algorithm discussed in section 2. In this section detailed study is undertaken for the identification of the computational bottlenecks among each step of the algorithm for an embedded implementation on FPGA platform.

For each step of the particle filter algorithm in the grid based Fast SLAM algorithm, profiling is done using a hardware timer. The summary of the results obtained from such study is given in Table I. As per the profiling results shown in Table I, the computation of sine and cosine functions account for most of the execution time in the sampling and importance weight steps of the particle filter algorithm. Furthermore, the generation of Gaussian and uniform random numbers accounts to most of the execution time in the sampling and resampling steps of the particle filter respectively. The computation of an exponential function also contributes in the computational complexity in the importance weight step of the algorithm for the evaluation of each particle weight.

The profiling information of the particle filter given in Table I can be used in the hardware/software partitioning i.e. which parts of the algorithm should be implemented in hardware and which ones can be kept in software (run on the FPGA’s embedded processor). It is clear from Table I that, the sine and cosine functions and random number generation are the critical bottlenecks of the grid based Fast SLAM algorithm and required to be implemented in hardware.

The hardware implementation for the sine, cosine and exponential functions is based on the COordinate Rotation Digital Computer (CORDIC) algorithm [20]. The uniform and Gaussian random number generation is based on the Tausworthe[21] and Ziggurat[22] algorithms respectively. The details of these algorithms are provided below and their hardware designs are given in section 4.

A. CORDIC Algorithm

CORDIC is an iterative algorithm that requires simple shift and addition operations, for hardware realization of basic elementary functions. It is based on the rotation of a vector in a Cartesian coordinate system and the evaluation of the length and angle of the vector. It can operate in one of three configurations: circular, linear, or hyperbolic. Within each of these configurations the algorithm functions in one of two modes rotation or vectoring.

$$
\begin{align*}
x_{i+1} &= x_i - \mu d_i y_i 2^{-i} \\
y_{i+1} &= y_i + d_i x_i 2^{-i} \\
z_{i+1} &= z_i - d_i e_i
\end{align*}
$$

Circular: \( \mu = 1 \)  
Linear: \( \mu = 0 \)  
Hyperbolic: \( \mu = -1 \)

Rotation mode: \( d_i = sign(z_i) \)  
Vectoring mode: \( d_i = -sign(y_i) \)

Figure 1 The unified CORDIC algorithm [23,25].
Using the unified CORDIC iteration equation in figure 1, wide range of functions can be evaluated. However the discussion here is focused to the evaluation of the sine, cosine, exponential and logarithmic functions (both required in the Ziggurat algorithm).

For the evaluation of exponential and logarithmic functions, the CORDIC is required to operate in hyperbolic configuration \((\mu = -1)\) with rotation \((d_i = \text{sign}(z_i))\) and vectoring \((d_i = -\text{sign}(y_i))\) modes respectively. For the evaluation of sine and cosine functions it is required to operate in Circular \((\mu = 1)\) rotation mode \((d_i = \text{sign}(z_i))\).

After \(i\) iterations, the unified CORDIC equations given in figure 1 converge to the following set of equations.

\[
x = K(xf_1 z - yf_2 z) \tag{6}
\]
\[
y = K(yf_1 z - xf_2 z) \tag{7}
\]
\[
z = 0 \tag{8}
\]

where, in circular-rotation mode of configuration the functions \(f_1\) and \(f_2\) correspond to the sine() and cosine() respectively and the constant \(K = 1.646\). For hyperbolic-rotation configuration \(f_1\) and \(f_2\) correspond sinh() and cosh() functions and the hyperbolic constant \(K = 0.828159\).

The evaluation of the sine and cosine functions are realized by setting \(x = 1/K, y = 0\) and \(z\) as the input argument to the CORDIC algorithm. In case of exponential function evaluation is obtained by setting \(x = 1/K, y = 0\) and \(z\) as the input argument and applying the property:

\[
\text{exp}(z) = \text{sinh}(z) + \text{cosh}(z) \tag{9}
\]

For the evaluation of natural logarithmic function, the CORDIC has to be configured in hyperbolic-vectoring mode and setting \(x = 1, z = 0\) and \(y\) as an input argument, the algorithm converges to:

\[
x = K'\sqrt{x^2 - y^2} \tag{10}
\]
\[
y = 0 \tag{11}
\]
\[
z = z + \tanh^{-1}\left(\frac{y}{x}\right) \tag{12}
\]

And the evaluation of natural logarithm function is obtained indirectly by:

\[
\ln w = 2\tanh^{-1}\left(\frac{y}{x}\right) \tag{13}
\]

Where, \(x = w + 1\) and \(y = w - 1\)

As a CORDIC algorithm works for a limited range of the input arguments for the evaluation of the elementary functions, it is required to extend the range of the inputs for each mode of operation by applying proper pre-scaling identities. This is achieved by dividing the original input arguments to the CORDIC algorithm by a constant to obtain a quotient \(Q\) and remainder \(D\) [23]. In the case of the sine and cosine functions the constant value corresponds to \(\frac{\pi}{2}\) and \(\log_2 2\) for the exponential and logarithmic functions. The pre-scaling identities for all the required functions are given in Table II.

### Table II

<table>
<thead>
<tr>
<th>Identity</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sin(Q^{\pi \pm D} + D))</td>
<td>(</td>
</tr>
<tr>
<td>(\cos(Q^{\pi \pm D} + D))</td>
<td>(</td>
</tr>
<tr>
<td>(\exp\left(Q\log_2 D + \sinh D\right))</td>
<td>(</td>
</tr>
<tr>
<td>(\log_2(MZ^h) = \log_2(M) + E\log_2 2)</td>
<td>(0.5 \leq M &lt; 1.0)</td>
</tr>
</tbody>
</table>

**B. Ziggurat algorithm**

For the generation of the normal random numbers required in the sampling step of the particle filter a Ziggurat algorithm is used in this work. In the Ziggurat method, normally distributed random variates are generated considering set of points \(C = \{(x, y)\}\) under the curve of a probability density function \(y = f(x)\) and a subset the points \(Z\) i.e. \(Z \subset C\) by uniformly taking the random points \((x, y)\) until \((x, y) \in C\).

The function \(f(x)\) is divided in to \(n\) rectangular blocks \(R_i\), where \(i = 0, 1, 2, ..., (n - 1)\), of equal area \(v\) except the bottom block which consists of a rectangular block joined with the rest of the density starting from a point \(r\). The rectangular blocks extend horizontally from \(x = 0\) to \(x_i\) and vertically from \(f(x_i)\) to \(f(x_{i-1})\).

- Given the set \(\{x_i\}_{i=0}^{n-1}\) generate normal random number \(x\).
  1. Choose an index \(0 \leq i \leq n - 1\) at random with uniform probability \(1/n\)
  2. Draw a random number \(U_0\) from a uniform distribution \(U(0,1)\) and let \(x = U_0 x_i\)
  3. **REPEAT**
  4. **IF** \(i \geq 1\) and \(x < x_{i-1}\) **RETURN** \(x\)
  5. **ELSEIF** \(i = 0\) (Generate an \(x\) from the tail \(x > x_n\))
    6. **DO**
    7. Generate i.i.d. uniform \((0,1)\) variates \(u_0\) and \(u_1\)
    8. \(x \leftarrow -\ln(u_0)\) \(r/y \leftarrow -\ln(u_1)\)
    9. **WHILE** \((y + y) < x^2\)
    10. **RETURN** \(x > 0\) \((r + x) : = (r + x)\)
  11. **ELSE**
  12. Draw a random number \(U_1\) from the uniform distribution
  13. **IF** \(U_1 |f(x_{i-1}) - f(x_i)| < f(x) - f(x_i)\) **RETURN** \(x\)
  14. **UNTIL** FALSE

Figure 2 Ziggurat Gaussian random number generation algorithm.
As can be seen from the pseudo code of the Ziggurat algorithm given in figure 2, a table of $x_i$ points and their corresponding function values $f_i$ are required for the algorithm. Normally, the number of the rectangular blocks $n$ is chosen as a power of $2(64, 128, 256)$. For $n = 128$ a value of $r = 3.442619855899$ is used is to determined the $x_i$ points that are required in the hardware implementation of the algorithm [21].

4. Hardware designs

In this section the hardware design of the CORDIC and Ziggurat algorithms is presented.

The implementation of the CORDIC module comprises of three hardware computational blocks; CORDIC-core, pre-scaling and post-scaling. The CORDIC-core implements the unified equations given in figure 1. The pre-scaling and post-scaling modules implement the equations given in table II for extending the range of the input arguments for the different functions.

To avoid the need for using multiple instances of the CORDIC module for the evaluation of different functions, a run-time configuration to the CORDIC module is provided through its config port. With this port the CORDIC module can be configured to operate any one of the functions as per the requirement during run-time. This helps to minimize the extra resource requirement if many instances of the CORDIC module is used for the evaluation of every function. The CORDIC-core in the proposed architecture of the CORDIC module, implements a serial CORDIC with an accuracy of 24 bits. Figure 3 shows the input/output interfaces to the CORDIC module. Where it has two possible inputs $U_0$ and $U_1$, and a two bit configuration input port config to choose a specific function for evaluation. Depending on the configuration bits on the config input interface to the CORDIC module, one of the functions are evaluated and the result is provided at the output interface. Even though not shown in figure 3, there are also other control signals to the input and output interface of the CORDIC module for its correct functionality.

The hardware architecture of the random number generator module is given in figure 4 and it comprises of the Tausworthe URNG, CORDIC and Ziggurat NRNG sub-modules. The Tausworthe URNG module is responsible for generation of two uniform random numbers ($U_0$ and $U_1$) that are used by the Ziggurat module and in the resampling step of the particle filter. The CORDIC sub-module implements the architecture given in figure 3.

The hardware design of the Ziggurat sub-module shown in figure 5 follows the description of the Ziggurat algorithm given in figure 2, where it composed of individual hardware blocks to generate the normal random number from the rectangular, wedge and tail region of the distribution (figure 2). All the three modules (wedge, rectangular and tail) work independently of one another. For the generation of random numbers from the wedge region (lines 8-9) the evaluation of an exponential function is required and in the case of the tail region (line 6) it is required to compute the natural logarithm function. For the computation of these functions a CORDIC algorithm is used.

As the Ziggurat algorithm requires simultaneous access to the coefficients $x_i$’s and $f_i$’s, it’s important to provide these values in parallel manner to speed up the normal random number generation process. To achieve this, the random index $i$ is divided in to even and odd values and the respective values of the $x_i$ and $f_i$ are stored in separate memories (figure 6). For example, if the generated index $i$ is an odd value the $x_i$ and $x_{i-1}$ are read from the odd and even memories at the same memory index positions simultaneously. However, if the
5. Proposed hardware/software architecture

The architecture of the proposed system for the particle filter in a grid based Fast SLAM algorithm is given in figure 7, where it comprises of an embedded Microblaze processor responsible for the execution of software functions, particle filter hardware accelerator (PF HW accelerator), timer and UART cores for the purpose analysis and verification of the system. The PF HW accelerator composed of CORDIC and random number generator cores explained in section IV and they are connected to the Microblaze soft-core processor through a dedicated one to one communication bus (Fast simplex Link) for fast streaming of data.

In the implementation of the grid based Fast SLAM algorithm, real time odometry and laser data collected from a mobile robot platform are stored in a Block RAM memory to be used for post processing. The odometry data is used by the sampling step to generate new particle instances and the laser data is used in the importance weight step for the evaluation of particle weights. The particles and their associated weights are stored in Block RAM for fast accessing. For accommodating all the code and data in the embedded processor two Block RAMS units are used in our implementation.

Individual CORDIC hardware modules are assigned for the evaluation of the elementary functions in the sampling and importance weight steps. Uniform and Gaussian random numbers are provided to the resampling and sampling steps of the particle filter respectively by the random number generator module.

6. Hardware/software co-design implementation results and discussion

The implementation of the architecture given in figure 7 is performed on a Xilinx Kintex-7 KC705 FPGA device running at 100MHz. The design of the hardware modules is written in VHDL language. For the implementation of the Zigurat module a value of $n = 128$ is used and, the $x_i$ and $f_{i_1}$ coefficients are represented as fixed point numbers with $Q_{8.24}$ format (i.e. 8 bits for the integer part and 24 bits for the fractional part). The same fixed point format is used for the representation of the different data in the CORDIC module design. For the variables in the software part of the algorithm a 32 bit floating point representation is used by enabling the floating point unit (FPU) of the Microblaze processor.

The summary of the execution time results in clock cycles for the three steps of the particle filter in the grid based Fast SLAM algorithm is given in table III. The obtained results in general shows that the hardware acceleration leads to better speed up in the execution time of the particle filter in all the three steps of the algorithm. In particular, significant speedup is achieved in the sampling step which can be attributed to the fast generation of Gaussian random numbers by the random number generator hardware module.

As the results of table III shows, the importance weight step takes relatively a large number of clock cycles compared to the sampling and resampling steps. This is due to the fact that, for the evaluation of the weight of each particle in the importance weight step, first it is required to transform every

<table>
<thead>
<tr>
<th>TABLE III</th>
<th>EXECUTION TIME FOR SAMPLING, IMPORTANCE WEIGHT AND RESAMPLING STEPS IN EMBEDDED SOFTWARE AND HARDWARE/SOFTWARE CO-DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Particles (N)</td>
<td>Sampling</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>50</td>
<td>N × 37192</td>
</tr>
<tr>
<td>60</td>
<td>N × 37192</td>
</tr>
<tr>
<td>70</td>
<td>N × 37192</td>
</tr>
<tr>
<td>80</td>
<td>N × 37192</td>
</tr>
<tr>
<td>90</td>
<td>N × 37192</td>
</tr>
<tr>
<td>100</td>
<td>N × 37192</td>
</tr>
</tbody>
</table>

Figure 6 Illustration for the storage of the $x_i$ and $f_i$ coefficients in memory and the parallel access of the $x_i \cdot x_{i-1} \cdot f_i$ and $f_{i-1}$ for $n=128$ generated index is with an even value, then $x_i$ is read from the even memory and the $x_{i-1}$ is read from the odd memory at the next memory position in parallel. As a result the parallel access of the coefficients is achieved with such simple but effective method.

Figure 7 Architecture of the hardware/software co-design system
Figure 8 Speedup in HW/SW co-design of particle filter laser scan measurements point data (range and bearing angle) from a robot frame of reference to a global frame of reference where normally more than hundreds of laser scans points have to be evaluated from the sensor. This requires the evaluation of sine and cosine functions for every scan point. After this step follows the search for the closest point between every laser scan end point and occupied points in a map (which is a computationally intensive step). However, at this stage this step is implemented in software in the embedded Microblaze processor but in the future we intend to speed up this process in hardware. Furthermore, computation of an exponential function is required in the calculation of the weight of each particle for every laser scan points. These are the main reasons attributed to the relatively large clock cycles obtained in the importance weight step.

7. Conclusions

This work presents a hardware/software co-design approach to speed up the computational time of the particle filter in a grid based Fast SLAM algorithm. With initial identification of the critical bottlenecks in each step of the particle filter algorithm, hardware acceleration blocks are designed and implemented to accelerate the computational time. Such simple and effective approach resulted in an improvement in the speedup of 140x, 14.87x and 19.36x in the sampling, importance weight and resampling steps respectively (figure 8). Such an approach can also be applied to other applications of the particle filter due the flexibility in the hardware/software co-design approach.

REFERENCES


