EFFICIENT MAPPING OF ADVANCED SIGNAL PROCESSING ALGORITHMS ON MULTI-PROCESSOR ARCHITECTURES

Bhavana B. Manjunath, Aaron S. Williams, Chaitali Chakrabarti, Antonia Papandreou-Suppappola

Dept. of Electrical Engineering, Arizona State University, Tempe, AZ 85287

ABSTRACT

Modern microprocessor technology is migrating from simply increasing clock speeds on a single processor to placing multiple processors on a die to increase throughput and power performance in every generation. To utilize the potential of such a system, signal processing algorithms have to be efficiently parallelized so that the load can be distributed evenly among the multiple processing units. In this paper, we study several advanced deterministic and stochastic signal processing algorithms and their computation using multiple processing units. Specifically, we consider two commonly used time-frequency signal representations, the short-time Fourier transform and the Wigner distribution, and we demonstrate their parallelization with low communication overhead. We also consider sequential Monte Carlo estimation techniques such as particle filtering, and we demonstrate that its multiple processor implementation requires large data exchanges and thus a high communication overhead. We propose a modified mapping scheme that reduces this overhead at the expense of a slight loss in accuracy, and we evaluate the performance of the scheme for a state estimation problem with respect to accuracy and scalability.

1. INTRODUCTION

Modern signal processing techniques are extensively used in diverse fields such as communications, acoustics, radar, sonar, and biomedical engineering. Some of the main processing schemes used in these applications include signal transformation for analysis, filtering, detection, estimation and classification. Literature provides numerous advanced algorithms to perform this processing, including the Fourier transform to extract the frequency content of a signal [1], time-frequency representations to analyze signals with time varying spectra, [2, 3, 4], filtering techniques to extract information from noisy data and sequential Monte Carlo algorithms to estimate state parameters [5, 6]. Most of these algorithms have high computational complexity and are used in applications that require close to real time calculations.

This work is partly supported under NSF Grant No. 0615135 and MURI Grant No. AFOSR FA9550-05-1-0443.

Recently, the IC industry has been pushing to place multiple processor cores on a die to increase throughput without significant increase in the power consumption. Examples of multi-core processors include the Intel Core 2 Duo, the AMD Opteron, and the Sun Niagra processor. In order to fully utilize the computational power of the multi-processor architectures, the algorithms have to be efficiently parallelized as well. However, the amount of performance enhancement possible is governed by the extent the algorithm can be parallelized.

The main objective of this paper is to parallelize some advanced signal processing algorithms and map them onto multi-processor architectures. Such an implementation requires partitioning the computational load between the multiple processors to minimize the overhead . In order to achieve this in some cases, algorithmic modifications are required. We focus our study on the following algorithms.

- **Time-frequency signal processing techniques:** short-time Fourier transform (STFT) and Wigner distribution (WD) time-frequency representations (TFRs).
- **Sequential Monte Carlo techniques:** particle filtering (PF).

This paper is organized as follows. In Section 2, we provide the implementation steps for the WD and the STFT TFRs on a multi-processor platform, and we show how these TFRs can be parallelized without modifying their algorithms. In Section 3, we develop modifications to the PF algorithm to minimize its communication overhead, and we discuss the effect of these modifications to estimation accuracy. In Section 4, we conclude the paper.

2. TIME-FREQUENCY SIGNAL PROCESSING

Although there has been an increasing demand in using TFRs such as the STFT and the WD to analyze time-varying signals in real applications, the TFR implementation can be computationally intensive. Here we provide a short description of these TFR algorithms, followed by their mapping onto multi-processor architectures.

2.1. Short-time Fourier Transform

The short-time Fourier Transform (STFT) is a linear TFR. It preserves time and frequency shifts on the analysis signal. Thus, it is a popular tool in speech processing, image processing, and filter bank decoding applications [2]. The STFT of a signal $x(t)$ is defined as:

$$S_x(t, f; h) = \int x(\tau)h^*(\tau - t)e^{-j2\pi f\tau}d\tau \qquad (1)$$

where t and f represent time and frequency respectively, τ is the variable of integration, and $h(t)$ is a lowpass window of duration $M\,T_s$, where T_s is the sampling period. For any given time, t, the signal is multiplied by a shifted version of $h^*(t)$, and the Fourier transform of the windowed signal is computed. This approximates the frequency components at time t by assuming that the frequencies present in this window do not change with time. The shorter the duration of this window, the better the frequency resolution in the signal analysis. However, this implies that more Fourier transforms need to be computed and thus the computational complexity and implementation time increases.

2.1.1. STFT Multi-processor implementation

The STFT can be computed in a multiple processor architecture with no loss of accuracy in the algorithm. The processors can communicate with each other through a global shared bus or through dedicated interprocessor links. For instance, processor i can communicate to processor $i-1$ and processor $i+1$ through these links. Assuming that we have N signal samples, M window samples and P processors, the data can be divided in a preprocessing stage into P sets. Each processor m computes N/P consecutive outputs, from mN/P to $(m+1)[N/P-1]$. To compute these outputs, the processor needs mN/P to $(m+1)N/P+M-1$. There are two options. In the first option, each processor must have the data samples mN/P to $(m+1)N/P+M-1$. The extra $M-1$ samples are sent to each processor so that it can complete its calculations without needing to request samples from its neighboring processor. The second option is that after each DFT is computed, processor m sends its latest data samples to processor $m-1$. While this takes only 1 cycle for an architecture with dedicated links, it takes P cycles for the shared bus architecture. However, if the computations in each processor are staggered, then the computations and the communication can be overlapped and the communication overhead is negligible.

Each processor computes the N/P outputs in the following way. It computes the first output from scratch and the subsequent $N/P-1$ outputs using the moving window discrete Fourier transform (DFT). Specifically, it uses the outputs of window i to compute the outputs of window $i+1$. Let $Y_i[k]$ be the kth output of window i, $0 \le k < M$. Then $Y_{i+1}[k]$ is given by:

$$Y_{i+1}[k] = (Y_i[k] + x[i+n] - x[i])e^{-\frac{j2\pi k}{N}} \qquad (2)$$

Thus, the M DFT outputs can be computed using M complex multiplications and additions using the moving window method.

The calculation of the STFT on a single core processor takes $M^2+(N-1)M$ complex multiplications. However, if the calculation is spread across P processors, the number of complex multiplications increases to $[M^2+(N/P-1)M]P$.

Example: Consider a signal for speech coding that is $N = 256$ samples long. If the window length, $M = 10$ samples, a uni-processor architecture takes 2650 complex multiplications to complete the STFT of this data segment. Now if we have 4 processors, each processor computes 530 multiplications and the total number of complex multiplications increases to 2920. However, if we continue to increase the number of processors to 8, the total number of complex multiplications increases to 3280. This is because the processor spends most of its time on the first DFT computation and cannot utilize the savings obtained by using the moving window DFT.

Figure 1 shows the total number of complex multiplications as the number of processors increases from 1 to 16. While the total number of multiplications increases, the number of complex multiplications that each processor performs decreases as P increases. Assuming that each complex multiplication takes one cycle, Figure 2 plots the number of cycles needed to complete all the multiplications. We see that the time to completion reduces as the number of processors increases. While the decrease is fairly sharp when P increases from 1 to 4, the decrease is quite modest when P increases from 4 to 16.

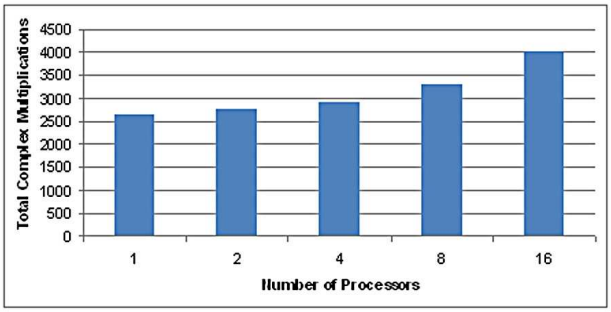

Fig. 1. STFT: Total number of complex multiplications

2.2. Wigner Distribution

The Wigner distribution (WD) is a quadratic TFR that satisfies many desirable properties such as time and frequency shift covariance and energy preservation. It has been used in communications, signal classification, sound synthesis and

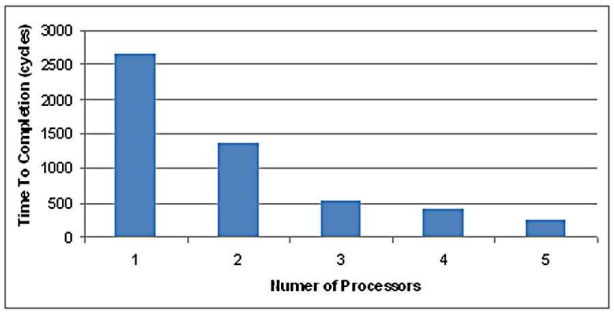

Fig. 2. STFT: Time to completion for a multi-processor architecture

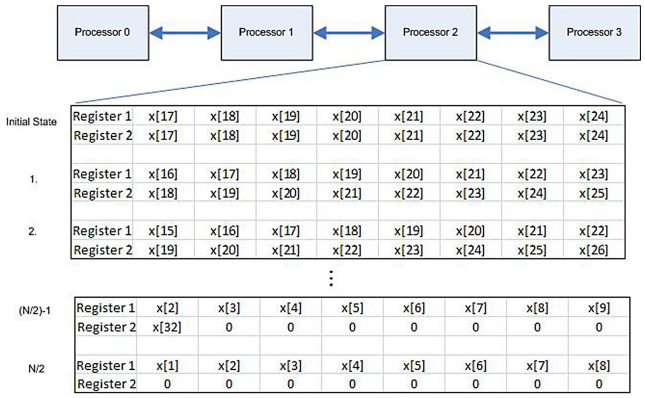

Fig. 3. Snapshot of Wigner-Ville computations in a 4 processor architecture

laser frequency analysis to name a few applications. The WD is defined as:

$$\mathrm{WD}_x(t,f) = \int x(t+\frac{\tau}{2})x^*(t-\frac{\tau}{2})e^{-j2\pi\tau f}d\tau \qquad (3)$$

where t and f are time and frequency, τ is the shift variable. For each t, the DFT is computed along τ.

There are two main steps for each time value t.

Step 1: Multiply the signal shifted by $\tau/2$ with itself shifted by $-\tau/2$; for different values of τ, and after discretization we obtain an $N \times N$ array indexed by $n = t/T_s$ and $m = \tau/T_s$.

Step 2: Compute the Fourier transform of the product.

2.2.1. Multi-processor implementation

If the number of processors is P, the discrete signal with N samples is first divided into P sets. Each processor has data values indexed from mN/P to $(m+1)N/P-1$ in register bank one (R1) and the corresponding conjugate values in register bank two (R2). Each processor m calculates the array entries for N/P time values from $t = mN/P$ to $t = (m+1)N/P-1$.

In order to calculate the set of values for $\tau = 1$, processor m transfers the data value $x[(m+1)N/P-1]$ to processor $m+1$. It shifts all the values it has in its first register bank over one allowing room for the incoming value from $m-1$. If there is no incoming value, a zero is simply written to this register. The opposite shift occurs for the conjugate values of the data in R2. Processor m transfers the value $x^*[mN/P]$ to processor $m-1$ and the new value is shifted in from processor $m+1$.

In each step, the products of the values in R1 and R2 are computed and stored followed by the data shift. This step is repeated until both signals have been shifted by $N/2$. Figure 3 shows a snap shot of the values in the two register sets as the steps are completed. We notice we must only calculate the values up to τ equals $N/2$ because the values for $\tau = -1,\ldots,-N/2$ and $\tau = 1,\ldots,N/2$ will be equivalent at a given time. We complete the matrix of data by duplicating these values about the row $\tau = N/2$. In this procedure, in each step $2P$ data are reassigned to the different processors. While this requires only 2 cycles for a multiprocessor architecture with dedicated interprocessor links, the overhead is a lot more for the shared bus architecture.

Step 2 is done in parallel in each processor. Specifically, each processor computes an N point DFT for N/P sets of data. This does not require any interprocessor communication.

Example: We consider a speech signal with 256 samples. For $P = 4$, the number of complex multiplications for Step 1 is 64. As the number of processors increases to 4, the number of complex multiplications in each processor reduces by a factor of 4. Thus there is no computational overhead due to parallelization. There is communications overhead in the shared bus architecture as described earlier.

3. PARTICLE FILTERING ALGORITHM

3.1. Background

Particle filtering is a sequential Monte-Carlo approach used to solve non-linear filtering problems. It involves sequential estimation of the states of a dynamic system based on received noisy measurements. Such problems are frequently encountered in applications such as sensing and target tracking, communications, navigation and neural network training [6].

Consider a filtering problem that estimates the unknown state vector $\mathbf{x}_k$ at discrete time instant k, based on a discrete time stochastic model given by,

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}) \qquad (4)$$

where $\mathbf{f}_{k-1}$ is a known, non-linear function and $\mathbf{v}_{k-1}$ is the process vector that represents possible state modeling errors. The estimation is based on available information that includes a set of noisy measurement $\mathbf{z}_k$, at time k. The measurement

is related to the state by the measurement equation given by,

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{w}_k) \tag{5}$$

where $\mathbf{h}_k$ is a known, non-linear function and $\mathbf{w}_k$ is the measurement noise vector. The filtering problem involves the estimation of the probability density function $p(\mathbf{x}_k|\mathbf{Z}_k)$, where, $\mathbf{Z}_k = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k\}$ which is achieved by the prediction and update stages [6, 7]. The prediction stage is a first order Markov process described by,

$$p(\mathbf{x}_k|\mathbf{Z}_{k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{Z}_{k-1})d\mathbf{x}_{k-1} \tag{6}$$

and the update of the prediction is achieved using Bayes' theorem,

$$p(\mathbf{x}_k|\mathbf{Z}_k) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Z}_{k-1})}{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Z}_{k-1})d\mathbf{x}_k}. \tag{7}$$

The PF method estimates the posterior probability density function by a set of random samples (particles) with associated weights that can be used to estimate the state. Let $\mathbf{X}_k = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k\}$ represent all the states upto time k. Let the posterior density $p(\mathbf{x}_k|\mathbf{Z}_k)$ be represented by a set of N particles $\{\mathbf{X}_k^i, i = 1, \ldots, N\}$ and associated weights $\{w_k^i, i = 1, \ldots, N\}$. From [7], the posterior probability density function can be approximated as,

$$p(\mathbf{x}_k|\mathbf{Z}_k) \approx \sum_{i=1}^{N} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i). \tag{8}$$

In the most commonly used PF algorithm, the Sequential Importance Resampling particle filter (SIRPF), the particles $\mathbf{x}_k^i$ are drawn from the transitional prior $p(\mathbf{x}_k|\mathbf{x}_k^i)$. The importance weights are approximated as [7],

$$w_k^i \propto w_{k-1}^i p(\mathbf{z}_k|\mathbf{x}_k^i). \tag{9}$$

The weights are normalized such that $\frac{1}{N}\sum_i w_k^i = 1$ and the particles are resampled based on the normalized weights.

The four main steps in the particle filtering algorithm can thus be summarized as (i) particle generation, (ii) weight evaluation, (iii) normalization of the weights and (iv) resampling. There has been some recent contributions for parallel implementations of particle filters. These include a parameterized framework for FPGA implementation [8] that reuses blocks, and algorithmic modifications to improve the speed of operation for the Gaussian particle filter [9] and KLD sampling [10]. New methods of resampling [11] such as the residual-systematic resampling, partial resampling, and delayed resampling have also been introduced to overcome the hardware complexity in the resampling stage.

3.2. PF Multi-processor Implementation

The multi-processor platform consists of a control processor and several processors that communicate with each other through a common bus. In order to implement the SIRPF on this multi-processor platform, the processing of N particles has to be distributed among the P available processors. The distribution has to be such that $g = N/P$ particles is processed in each processor, at each time step. The operations of the SIRPF can be divided into different stages; each processor performs some of these stages concurrently and interacts with the central processor in other stages. In this section, we first describe a straightforward mapping that involves significant communication between the control processor and the other processing processors. Next we show how the interprocessor communication can be minimized by algorithm modification.

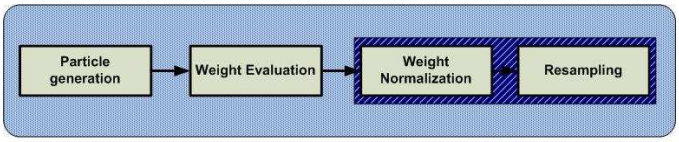

Fig. 4. Block diagram showing the different stages of a SIRPF

3.2.1. Direct Mapping

The block diagram of the operations to be accomplished by each processor in each time step k is shown in Figure 4. The shaded blocks represent the stages which require inter-processor communication. The particle generation stage involves the evolution of the particles based on the state equation. The importance weights are evaluated based on the received data using the measurement equation. These stages can be performed in each processor independently. The normalization of weights require the information of the weights of all the particles. Let $x_{i,m}$ represent the ith particle in the mth processor with $\{i = 1, \ldots, g\}$ and $\{m = 1, \ldots, P\}$ and $w_{i,m}$ be the corresponding normalized weights. Each processor computes and sends the sum of the weights $\Omega_m = \sum_i w_{i,m}$ to the central processor. The central processor computes and returns the normalization factor $W = \sum_m \Omega_m$ to each processor. Each of the $w_{i,m}$ weights are normalized using this W to obtain the normalized weights.

Next, resampling is performed to eliminate the problem of degeneracy. The technique involves elimination of particles with low importance weights and repetition of those with high importance weights. To perform resampling without any modification, the central processor needs the information of all the particles and the corresponding weights. This requires the movement of two g-dimensional vectors, (particle and weight vectors) from each processor to the central processor for a 1-D state estimation problem. During resampling, a g-dimensional vector of replication factors is computed for each of the particles in the central processor. Upon reception of this vector, the particles are replicated, which may give

rise to imbalance in the number of particles in each processor. Inter-processor communication would be necessary to maintain g particles in each processor. It must be noted that the amount of data transfer for such an operation varies in each iteration and also depends on the distribution of the particles. In a worst case scenario, only one of the processors may have all the particles that have non-zero weights. In such a case, particles from that processor need to be distributed among all other processor. This stage greatly impacts the overhead and thus the speed of the multi-processor implementation.

3.2.2. Modified Mapping

In order to reduce the interprocessor communication, we make some algorithm level modifications such that the information about *all* the particles and weights is not provided to the central processor. Rather, a reduced set of information is provided to the central processor which now has to use this to obtain the replication factor for the particles. These modifications come at the cost of some accuracy. The modifications are as follows.

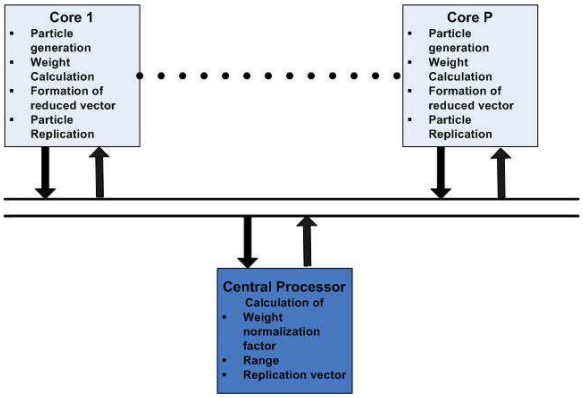

Fig. 5. Functional block diagram of multi-processor implementation

- *Step 1:* First, in each processor, sort the $x_{i,m}$ particles in increasing order. The lowest and highest values of $x_{i,m}$ represented by xl_m and xh_m are sent to the central processor. The processor calculates $XL = \min(xl_m)$ and $XH = \max(xh_m)$ and returns these values to each processor.

- *Step 2:* Next, represent the particles and the corresponding weights in a reduced set comprising of $R \leq g$ values. In order to do so, choose a parameter δ which defines the number of groups into which the g particles are divided, and obtain $R_m = \mathcal{C}\left(\frac{XH-XL}{\delta}\right)$ where $\mathcal{C}(a)$ is the least integer greater than a. The choice of δ is crucial as it impacts the accuracy of the performance. Represent the information in each group by the averages of the particles and the corresponding weights. Thus processor m contains a set of averages and a set of weights for the R_m groups, $\{\mu x_{j,m}\}$ and $\{\mu w_{j,m}\}$, respectively.

- *Step 3:* Each processor sends only $\mu w_{j,m}$ to the central processor which then evaluates the vector of replication factor, ρ_j, for each of the $\mu x_{j,m}$. It also ensures that the replication factor is an integer value by simple rounding-off operations and that the $\sum_j \rho_j = g$.

The above procedure reduces the amount of data communication significantly. However, there is a trade-off in the performance accuracy as the average value in each group ($\mu x_{j,m}$) is replicated instead of each $x_{i,m}$. Figure 5 provides the functional block diagram of the multi-processor implementation using P processor. Figure 6 shows the sequence of operations between one processor and the central processor for one time step.

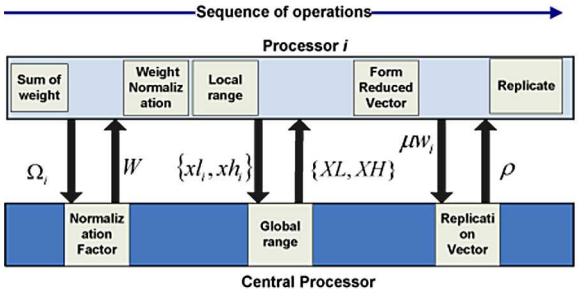

Fig. 6. Communication between processor i and central processor in each time step.

3.3. Simulation Results

To evaluate the performance of the above procedure, we simulated the scalar estimation problem discussed in [12] where the state sequence x is estimated from noisy measurements. The process and observation models are given by,

$$x_{k+1} = 1 + \sin(\omega\pi k) + \phi_1 x_k + v_k \tag{10}$$

$$y_k = \begin{cases} \phi_2 x_k^2 + n_k, & k \leq 30 \\ \phi_3 x_k - 2 + n_k, & k > 30 \end{cases} \tag{11}$$

where v_k is the process noise modeled by a Gamma random variable with shape 3 and scale 2 and n_k is zero mean additive white Gaussian noise with variance 0.00001. The scalar parameters chosen are $\omega = 4 \times 10^{-2}, \phi_1 = 0.5, \phi_2 = 0.2$, and $\phi_3 = 0.5$. The estimation for 60 time steps was conducted for SIRPF with 1000 particles. Three platforms are considered with $P =$ 1,4 and 8. The computation of the 1000 particles was distributed equally among the P processors. For the 4 and 8-processor implementation, δ was chosen as 10. Each algorithm iteration was averaged over 100 Monte-Carlo simulations on an Intel dual-core Pentium-D 3GHz system with 2GB RAM. Random re-initialization were done for

each run. Figure 7 shows the overlaid performance plot for the x_k estimation for $P = 1$,4 and 8 processor system. We see that the estimation performed using a single processor has very high accuracy. Infact, the average deviation from the true value is as low as .1%. For the 4-processor and 8-processor case the relative deviation was 5% and 11.13% respectively. However, the time taken reduces as expected. Basically, as the number of processor increases the number of particles processed by each processor reduces and so the processing time in each processor reduces. But, the amount of data sent to the central processor increases thereby increasing the communication time.

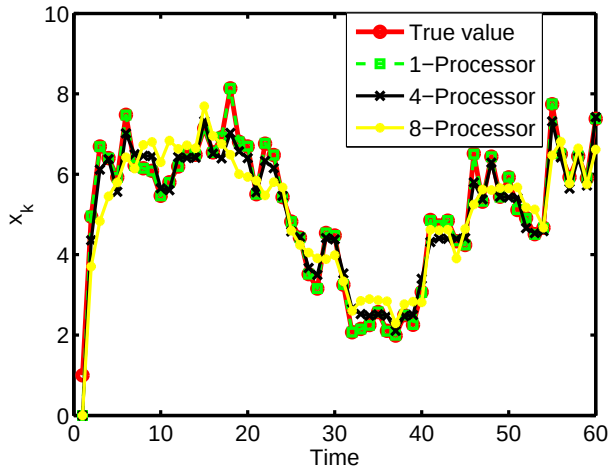

Fig. 7. Comparison of scalar estimation performance.

4. CONCLUSION

In this work, we have investigated the mapping of the time-frequency signal processing techniques, STFT and WD, and the particle filtering algorithms onto multi-processor architectures. Both the STFT and WD can be parallelized without any loss of accuracy. In the case of the STFT, as the number of processor increases, the computational overhead increases. However, in the case of the WD, the overhead remains constant. For the particle filtering algorithm, we have presented an implementation with some algorithmic modifications to overcome the high communication bandwidth requirement. Application of the modified scheme to a state estimation problem with 1000 particles divided among 4 processors shows a deviation of 5% from the true estimate. However, the savings in time was greater than 50% in the 4-processor implementation. Also, the the time spent by the central processor for communication and processing was less than 20% of the total time. In contrast, the direct mapping has high estimation accuracy with a deviation of only .1% but requires communication of all the particles to the central processor.

5. REFERENCES

[1] J. W. Cooley and J. W. Tukey, "An algorithm for machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[2] A. Papandreou-Suppappola, Ed., *Applications in Time-Frequency Signal Processing*, CRC Press, Florida, 2002.

[3] F. Hlawatsch and G.F. Boudreaux-Bartels, "Linear and quadratic time-frequency signal representations," *IEEE Signal Processing Magazine*, vol. 9, no. 2, pp. 21–67, April 1992.

[4] L. Cohen, "Time-frequency distribution - A Review," *IEEE proceedings*, vol. 77, no. 7, pp. 941–981, July 1989.

[5] R. A. Altes, "Detection, estimation and classification with spectrograms," *The Journal of the Acoustic Society of America*, vol. 67, no. 4, pp. 1232–1246, April 1980.

[6] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House Publishers, 2004.

[7] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, Feb 2002.

[8] S. Saha, N. K. Bambha, and S. S. Bhattacharyya, "A parameterized design framework for hardware implementation of particle filters," *International Conference on Acoustics, Speech, and Signal Processing*, March 2008.

[9] M. Bolic, "Architectures for efficient implementation of particle filters," *Dissertation, The Graduate School, Electrical Engineering, Stony Brook University*, 2004.

[10] B. B. Manjunath, A. Papandreou-Suppappola, C. Chakrabarti, and D. Morrell, "Computationally efficient particle filtering using adaptive techniques," in *Sensor, Signal and Information Processing Workshop*, May 2008.

[11] M. Bolic, P.M. Djuric, and S. Hong, "Resampling algorithms for particle filters a computational complexity perspective," *EURASIP Journal on Applied Signal Processing*, vol. 15, pp. 2267–2277, 2004.

[12] R. van der Merwe, J.F.G. de Freitas, A. Doucet, and E. A. Wan, "The unscented particle filter," Tech. Rep. CUED/F-INFENG/TR380, Cambridge University Engineering Department, Aug. 2000.