Parade: A Versatile Parallel Architecture for Accelerating Pulse Train Clustering

Amin Ansari, Dan Zhang, and Scott Mahlke
Computer Science and Engineering Department, University of Michigan – Ann Arbor
{ansary, danz, mahlke}@umich.edu

***Abstract* — In this paper, we present Parade, a novel and flexible parallel architecture for the deinterleaving of combined pulse-trains. This is a commonly performed task in various areas of signal processing applications, such as satellite communication. Most of these applications require the identification of the main characteristics of pulse-trains such as frequency. Previously suggested techniques for solving the clustering problem are restricted with several limiting assumptions. In contrast, Parade, based off a parallelized and improved version of the sequential search algorithm, solves the deinterleaving problem significantly faster and in a more general case by considering all conditions such as jitter, dropped pulses, arbitrary start and end points. Our scheme employs several parameters, such as the number of deinterleaving modules and the number of memory elements, in order to achieve a desirable combination of accuracy, speed, memory usage and area. Using an 8-way parallel architecture, Parade improves the PRI accuracy by 27% compared to the non-parallel baseline architecture. Our design, when synthesized on 90nm technology node, performs 940x faster compared to a software-based histogram technique.**

I. Introduction

Pulse-train deinterleaving deals with the separation of multiple interleaved repetitive sequences into their components. Although most famous for being a vital component in electronic support measures (ESM) processing, pulse-train deinterleaving also has applications in other domains. For example, satellite clusters need to deinterleave the pulses that it receives from different terrestrial emitters [17]. Also, when several nodes enter an asynchronous ad-hoc network and try to communicate with one point, we will encounter a similar problem [18]. In addition, one of the most important applications of pulse-train deinterleavers is for finding the physical location of the different sources according to the interleaved pattern of the received time of arrivals (TOAs) [1]. This latter has a wide range of applications in various wireless communications and sensor networks.

Figure 1 shows the outline of an ESM system utilizing a pulse-train deinterleaver, which consists of an antenna for receiving the pulses and other corresponding processing elements [12]. Here, the feature extractor determines the main characteristics of the incoming pulses such as degree of arrival (DOA), TOA, and pulse width (PW). In the next step, a subset of this information is used by the deinterleaver to cluster the combined pulse-trains. A few algorithms have been recently proposed in order to efficiently split these combined pulse-trains into distinct pulse sequences. The histogram method [2] is one of those approaches, in which a histogram of TOA differences with different orders are cumulatively formed and then evaluated to determine the pulse-trains pulse repetition interval (PRI). The sequence search algorithm [19] is a well-recognized method for decomposing the pulse-trains. In this approach, all of the possible PRIs are found based on the samples. Then, each postulated PRI is matched against the samples to determine whether or not it is a valid PRI. Another proposed algorithm is an extension of the Kalman filter approach to deinterleaving, using a modified version of the signal model [3][16]. Matrix-based methods have recently been proposed for PRI identification to utilize the inherent parallelism of such methods [4]. Other techniques for solving this problem are the Hough transform method [5], Monte-Carlo method [6], AI knowledge-based agent design method [9], Software-based improved histogram [11], Neural Network [10][13][14][15], Multiple Masks operation, Hidden Markov Model [7] and improved Fast Fourier Transform techniques [8].

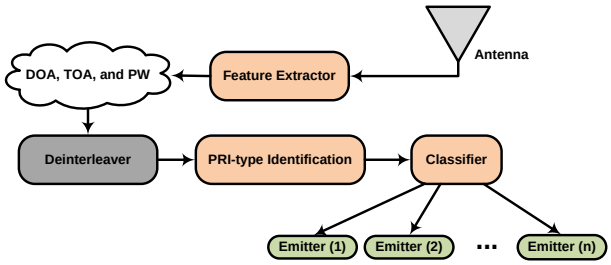

Figure 1. Outline of an ESM system

Out of these techniques, sequence search has several notable advantages: **1)** It is able to extract the most information regarding to each pulse-train (e.g. TOAs of pulses and phase of a pulse-train) **2)** As it will be discussed later, sequence search scheme can be efficiently parallelized. **3)** It can also account for the arbitrary start and end points of the pulse-trains. In contrast, most other techniques are only able to find the PRI of each pulse-train without accounting for the mentioned non-idealities. However, the conventional sequence search does not perform well in high-noise applications due to the large number of dropped pulses and high jitter value. In addition, sequence search can be extremely slow when it accounts for high jitter values and arbitrary start and end points [19].

A. *Contributions*

As mentioned before, conventional sequence search faces with several issues when operating in a non-ideal environment. Consequently, in this work, we firstly modify the original algorithm to allow more robust operation in the presence of dropped pulses and high jitter values. These two problems along with arbitrary start and end points are the main causes of accuracy loss in most deinterleavers. Our improved sequence search leverages a chancing algorithm to handle dropped pulses and jitter-related issues. In order to tackle the arbitrary start and end points issue, we parallelize our improved sequence search algorithm. This parallelism allows us to achieve higher accuracy while accelerating the clustering process. Finally, our parallel pulse-train deinterleaver, Parade, solves the problem in the most general case, by considering the following: multiple consecutive dropped pulses, arbitrary starting points for the pulse-train in the entire time frame, arbitrary length for each pulse-train, arbitrary jitter value, and

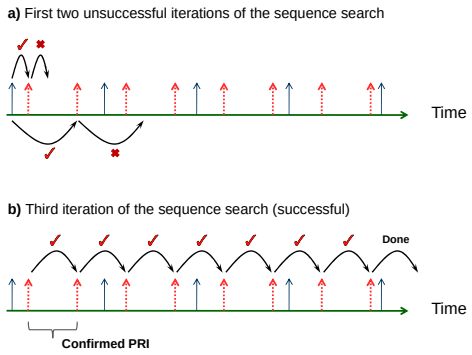

Figure 2. Demonstration of the sequence search algorithm

an arbitrary number of emitters in the environment.

The hierarchical design of the Parade's architecture allows easy changes to the architecture according to the type of the problem which is addressed. As an example, a sensor network mostly values low power and low area, while an ESM system does not. To ensure flexibility of the design, various input parameters control a combination of speed, accuracy, area, and power consumption. We also provide analytical and simulation-based analysis on power dissipation, area, and performance of our proposed architecture.

II. Background

In an environment with multiple pulse sequences travelling through a common medium, sensor devices will capture the pulses as an interleaved pulse-train. The goal of pulse-train deinterleaving is to convert the resulting interleaved signal into the original components. Pulse-train deinterleaving algorithms use various parameters of the received sample pulses such as TOA, DOA, pulse amplitude, pulse width, and carrier frequency. These parameters are given in order to extract associated pulse-trains from the sample. Pulse sequences are characterized by PRI, which specifies the length between two consecutive TOAs. PRI is the reciprocal of pulse repetition frequency (PRF). Signals with different PRIs can be assumed to be a unique characteristic of each emitter. Therefore, the objective of most of the previously studied algorithms is to extract the respective emitter pulses from the sample based on their PRIs. Another important aspect of the samples is the jitter due to inherent characteristics of the emitters, medium, and environment. In fact, jitter can cause TOAs to deviate around a nominal value which create difficulties for the deinterleaver to extract the right sequence from the sample pulses.

III. Conventional Sequence Search Algorithm

In a nutshell, the sequence search algorithm extracts sequences of identical intervals from the sample, which can be achieved by calculating all the TOA differences in the sample at all phases, and then matching each postulated difference with the sample (Figure 2). TOA values are sampled at the positive edge of each pulse. For bringing out the signal patterns, the sequence search algorithm needs sufficiently large samples. Therefore, the sample of pulses consists of a sequence of timing events.

When dealing with simple cases with low jitter value, this algorithm is efficient and accurate. However, in dense environments with measurement errors and missing pulses, the original algorithm is not enough. In fact, the algorithm frequently will incorrectly extract multiples of actual PRI values, or discard a correct PRI assumption due to a missing pulse. Due to these shortcomings, although sequence search is able to extract the most information from the pulse-trains, it is generally not very popular. Therefore, further modifications to the algorithm are necessary to improve overall accuracy and speed.

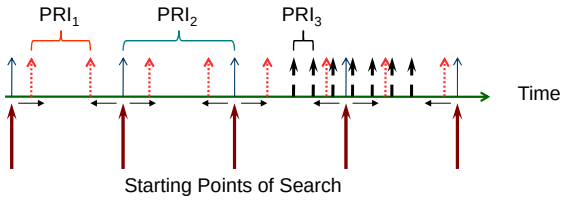

Figure 3. Demonstration of our sequence search algorithm

IV. Parade's Parallel Deinterleaving Algorithm

In order to parallelize and improve the accuracy of the conventional sequence search algorithm, we made several major changes to the base-line version [19]. Our modified algorithm searches the whole interleaved pulse-train from several different starting points. These starting points have a uniform distribution over the sampling interval. Thus, different deinterleaver modules can work in parallel starting at these locations in the pulse-train. Our second contribution to the base algorithm is that we search for valid PRIs in both directions when possible. This kind of forward and backward search for finding a valid PRI will increase the accuracy because it will test approximately two times more candidate PRIs when searching for a valid PRI in a particular neighborhood. For instance, in Figure 3, third pulse-train (with PRI_3) can be identified more accurately using the bidirectional search. The jitter problem can be addressed by supposing slightly larger or shorter PRIs below a certain threshold value are also acceptable. In other words, our algorithm accepts some reasonable error value in our candidate PRI when attempting verification of the PRI.

However, in some situations, our algorithm may accidentally remove signals that do not belong to the proposed PRI pulse-train. Moreover, problems present in noisy environments, such as dropped pulses, can cause errors in the final identification of pulse-trains. In order to tackle this latter problem, we propose a flexible and adaptive chancing algorithm.

A. The Necessity of Chancing

In a non-ideal environment, jitter is present in the entire pulse sequence. Thus, when eliminating a sequence of pulses from the interleaved pulse-train, we should consider an interval around each TOA in order to find the associated TOAs in the distinct PRI sequence of each emitter. The presence of jitter in our model can create problems especially when dealing with high-density interleaved pulse-trains. Since the TOAs of different emitters are close to each other in high-pulse-density emitters, trying to eliminate the TOAs of emitter with PRI_i may lead to some erroneous efforts when removing the TOAs of emitter with PRI_j. Therefore, when removing the TOA sequence of emitter with PRI_j from the sample, we are faced with some gaps in its sequence. The conventional sequential search algorithm would stop upon reaching a gap where there is no TOA to remove.

However, we employ a chancing state machine to avoid halting upon reaching a gap in the PRI sequence associated with each emitter. Instead, we test a few more consecutive

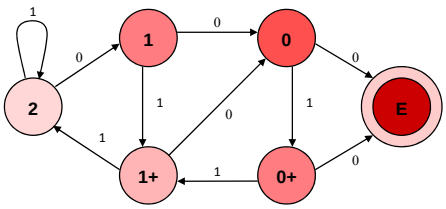

Figure 4. Chancing state diagram for allowing 2 consecutive dropped pulses. Here, a "0" transition represents a gap.

TOAs in order to find out whether this gap shows an ending to the TOA sequence.

B. The Chancing Algorithm

One of the possible chancing state diagrams is shown in Figure 4. When extracting TOAs from the postulated PRI sequence, we follow this state diagram. The general idea, in the case of this example, is to assign 3 chances for successive failures when finding TOAs. However, 1 or even 2 successive gaps do not necessarily show the end of the sequence. Each state in this diagram shows the number of chances left to have gaps in the sequence; so, the initial state named 2 shows the state in which we have 2 chances left for visiting gaps in the sequence and thus not the reaching the end of the process. The finish state *E* shows the state in which we have committed 3 successive failures in reaching TOAs, thus the extracting phase is completed with the postulated PRI. The shown state diagram is specially designed to prevent cases of finding multiples of the real PRI. Depending on the density of the combined pulse-trains, environmental noises, and jitter value, this state diagram might change to allow more/less number of consecutive gaps in each sequence. However, allowing large number of consecutive dropped pulses results in significant runtime and area penalties.

V. PARADE'S ARCHITECTURE

Our proposed architecture consists of several major stages, as illustrated in Figure 5. In the first stage, the incoming signals are received in real-time from the external source of pulses. Depending on the current time frame, the timing module inserts the received pulses into either the left or the right main memory. The received pulses stored sequentially in the main memory to their TOA. After the time frame is over, the timing module sends a start signal to the distributor and starts to fill the other main memory while other parts of the architecture start to work with the first filled main memory. In the next stage, the distributor divides the entire interleaved pulse-train from the main memory between the memory modules Mem_1 to Mem_K. After all the memory modules have been filled, the distributor sends a start signal to the controller, which arbitrates multiple read requests to the same memory module. Each main module sends memory read requests to its controller and tries to identify candidate PRIs and verify them using the deinterleaving algorithm discussed in the previous section. In the next step, the minimum circuit stage compares all the discovered PRIs and sends the smallest to the remover, which will delete the corresponding pulses from the main memory. The process repeats until either the time frame is up, or no other pulse-trains can be extracted from the main memory.

A. Timing Module

The timing module is the top level module, and interacts with two main memories. Upon receiving the input sample from the external source of pulses and extracting its TOA by measuring the times in which a leading edge happens in the pulse-train, this module writes the result into one of the main memory banks. When time frame (t_s) is over, the distributor is notified and begins processing the data contained in the filled main memory bank. Meanwhile, the timing module fills the other main memory bank. After another t_s , the timing module again switches to write into the first main memory bank and notifies the distributor to work on the second main memory bank, and this procedure continues repeatedly.

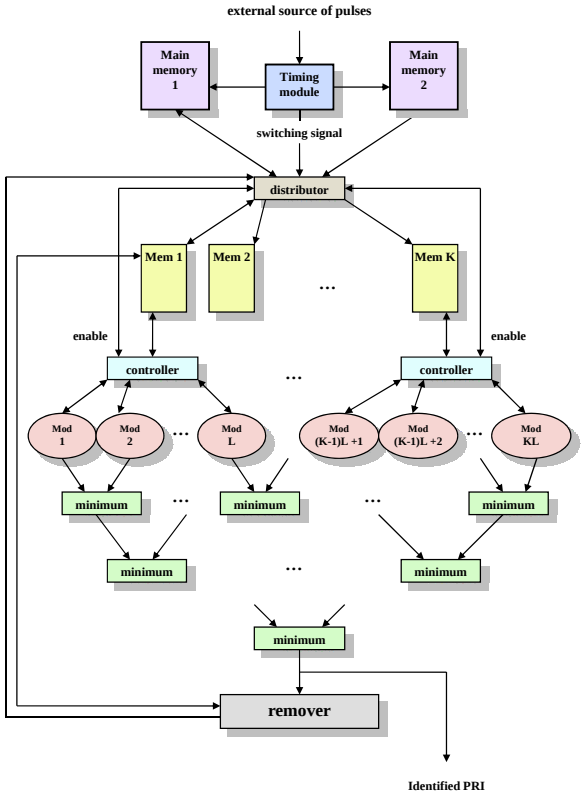

Figure 5. Outline of the Parade's architecture

B. Distributor Module

The main task of the distributor is to divide the contents of the current main memory chosen by the timing module between memory modules $Mem_1, \ldots, Mem_K$. Afterwards, the number of sampling intervals is broken down into K equal intervals. We let memory Mem_i have the TOAs starting from the start-time of the i^{th} interval to the end-time of the last interval in the pulse sequence. In other words, memory module 1 contains the entire TOA pulse sequence, while each following memory module contains a smaller and smaller subset of the pulse sequence, starting at some offset value and continuing to the end of the pulse sequence. This asymmetry is an area optimization that exploits the fact that each main module starts its deinterleaving algorithm at a different location in the pulse-train. Splitting the memory into K evenly-sized components is not possible since this would fail to discover pulse-trains with large PRIs.

C. Controllers

In the next stage, a controller module acts as an arbitrator between each memory module and the L main deinterleaving modules that connect to each memory module. Note that the total execution time of the group of L main modules is not a function of the reading order of them. Therefore, in the

controller module, we can simply use a priority encoder.

D. Main Modules

The main modules perform the work of finding PRIs within the interleaved pulse-train sequence using the deinterleaving algorithm described in Section IV. Each main module postulates its candidate PRI based on the TOAs of the i^{th} pulse group to the end of a sampling sequence. After the main modules finish running the deinterleaving algorithm, they place the found PRI on its output line and inform the minimum circuit that their output is valid. Each main module tries several times different candidate PRIs and goes through the next pulses to verify that PRI is actually exists in the interleaved pulse-train.

It is possible to have a long interleaved pulse-train. When such a case happens, it will take a long time for the main module to go through the entire sequence to verify one PRI. Thus, we select a sufficient threshold number S which describes the minimum number of pulses found belonging to a pulse-train such that the algorithm can consider the assumed PRI to be correct. This number would be directly proportional to the jitter and PRF. When one candidate pulse-train has a large estimated jitter/PRF value, the main module will increase the value of S to avoid identifying an incorrect PRI.

Each main module requires a considerable number of memory accesses while performing its PRI calculations. First of all, it sets up two pointers (P_1 and P_2) to two successive TOAs in the memory (T_1 and T_2). The difference between P_1 and P_2 is used as the candidate PRI. After that, the main module will try to verify that candidate PRI by going through the whole interleaved pulse-train in the memory. Since TOAs are stored in memory in a sequential manner, the module simply walks P_2 through memory, using another pointer P_3 to keep track of the last value of P_2. At each stage, the main module computes the difference between the TOA values with T_2 until it reaches a situation in which the difference becomes equal to or greater than the candidate PRI. At this point, it will compare the difference between P_2 and P_1 as well as P_3 and P_1. The value closest to the candidate PRI is checked to see whether it is acceptable according to the jitter and chancing algorithm. If the candidate is acceptable, the module will update P_1 to the newly accepted TOA and will try to find another acceptable pulse by going through the TOAs through the memory in a similar fashion. On the other hand, if it would not be acceptable because of the chancing problem or the error would be greater than jitter value, it will try another candidate PRI by calculating the difference between a new pair of T_1 and T_2.

E. Remover

After the minimum circuit tree output becomes valid, the final stage, the remover, is activated. The remover attempts to remove the pulse-train with minimum PRI found by the minimum circuit tree from all the main modules. After being provided with a starting point and the PRI, the remover uses an algorithm similar to the one described in Section IV to remove all the pulses it finds in each memory module. After a pulse is identified to be removed, the remover would invalidate the pulse entry and shift the next pulse into the invalidated entry such that the memory remains contiguous. This process is repeated until all of the pulse-train sequences are extracted from the sample.

VI. Parameters of the Architecture

Parade's architecture provides different characteristics depending on the required performance, accuracy, and also the design constraints (e.g. chip area and power consumption).

A. Resources

As we have K memory modules each of which having L main modules, the total number of modules is LK. Thus, increasing the amount of memories and modules would lead to considerably more area. Secondly, the size of the memory depends directly on the parameter K.

B. Speed

Considering a group of main modules, the control circuit at the upper part of the group controls access to the memory modules for their respective main modules so that only one memory access is done at each time. Moreover, the minimum circuits at the end of each group should be supplied with valid data in order to calculate the minimum of PRIs. We can therefore improve the speed by minimizing the number of memory induced stalls by the main modules.

C. Accuracy

To analyze the dependency of accuracy on its associated parameters, we should note that extracting the pulse-train with the lowest PRI would enable us to remove the largest number of TOA entries. Therefore, deinterleaving process would be easier and more reliable as we are left with a much lower pulse density at the end of each removal step. In addition, since sequence search is prone to removing multiples of the correct PRI value, removing the minimum values first would prevent the occurrence of this artifact. By splitting the sample length into more intervals and letting the modules work on a much smaller interval, each group of modules would more likely declare the lowest PRI as their first try. As a result, having more main modules increases the number of intervals to work on and thus higher accuracy can be achieved.

VII. Complexity Analysis

A. Space Analysis

We have K memory modules named $Mem_1, \dots, Mem_K$. If the amount of memory needed to store the entire TOA sequence is M bytes, then the i^{th} memory module from the left in Figure 5 would require $(K - (i - 1)) \times (M / K)$ bytes, as it only needs to store the TOA sequence from the start time of the i^{th} interval. Therefore, the total amount of memory required is $M_T = 0.5 \times (K + 1) \times M$.

Also, if we assume that we need N_m gates for a minimum circuit, N_{mo} for a main module, and N_c for a controller module, the total number of gates would be, the following, in which F is the number of input lines to each minimum circuit:

$$KL \times N_{mo} + \lceil \log_F LK \rceil \times N_m + K \times N_c$$

B. Run-Time Analysis

In order to estimate the runtime of our proposed architecture, we should first note that the algorithm is run $O(m)$ (where m is the number of emitters) times to remove the associated pulse-train of each emitter. In each iteration, $O(n)$ is needed to write n TOAs into the memory. Our sequence search algorithm running in the presence of $m$emitters and a sample of n pulses would take an asymptotic runtime of $O(m^2n)$.The third stage has K parallel groups of main modules and each group work concurrently with other groups; however, inside each group, modules perform their task sequentially. Therefore, the running time of this layer would be L times the running time of our sequence search algorithm which leads to $O(m^2nL)$. For the

minimum circuits, the critical path of the architecture is $O(\lceil \log_F LK \rceil)$. Lastly, the remover can perform its task in $O(n)$. Thus, the runtime of the architecture in the worst case is:

$$O(m) \times \left(O(n) + O(m^2 Ln) + O(\lceil \log_F LK \rceil) + O(n)\right)$$

Since in a realistic situation, m and L are mostly less than or equal to eight [20], we can conclude that the runtime is $O(n)$.

VIII. Experimental Results

This section presents our experimental platform, as well as results of the experiments performed regarding accuracy, area, power efficiency, performance, and clock speed across multiple architectural configurations.

A. Evaluation Platform

We implemented Parade's architecture in Verilog using the ModelSim environment. In order to obtain area, power, and delay of the design, Synopsys standard industrial tool-chain was employed (with TSMC 90nm technology library). Furthermore, a Monte Carlo engine was developed in C/C++ to evaluate the accuracy and runtime of the different system configurations. In each iteration, a combined pulse-train with average size of 5000 pulses is generated as the input to the Verilog simulator. For each system configuration, 100 such iterations are run for conducting the Monte Carlo study.

In order to highlight effectiveness of our proposed scheme, PRF of the generated pulse-trains varied between 100Hz to 300KHz. By considering such a wide PRF range, detection of the emitters becomes significantly harder. However, this conservative range covers all practically possible PRFs reported in [20]. It should be noted that the usage of a narrower PRF range simply results in a higher PRI extraction accuracy [8][11]. In addition, during the Monte Carlo simulation, jitter and pulse-train offset values drawn uniformly from zero to 10% (for extremely noisy environments) of the pulse-train PRI and zero to 75% of the sampling interval, respectively [6][14][15][20].

B. Results

To explore the effect of parallelism on accuracy, we varied the number of main modules from one to eight and varied the number of memory modules from one to four. Each of the 100 test cases was run on the all of the different configurations. The output PRI and completion time of each of the different configurations were recorded. In this context, accuracy is defined as the number of correct PRI predictions divided by the total number of interleaved pulse-trains within the system. Runtime, our second performance metric, is calculated as the number of cycles that it takes to consume the pulse data in a main memory. The configurations are represented in the graphs as *x.y*, where *x* is the number of memory modules (i.e. *K* in Figure 5) and *y* is the number of main modules attached to each memory module (i.e. *L* in Figure 5).

Figure 6 depicts the PRI extraction accuracy of each configuration. Here, runtime results are normalized to the runtime of the baseline (i.e. *1.1* configuration). As can be seen, varying the number of memory modules did not have any effect on accuracy as expected. The baseline case performs poorly, with correct outputs only 69% of the time. We then see an improvement in accuracy going to two and then four main modules, with diminishing returns with eight main modules at 96% accuracy. These results demonstrate the flexibility of our architecture: in embedded applications such a car sensor for smart roads and highways or an ad-hoc sensor network, a two

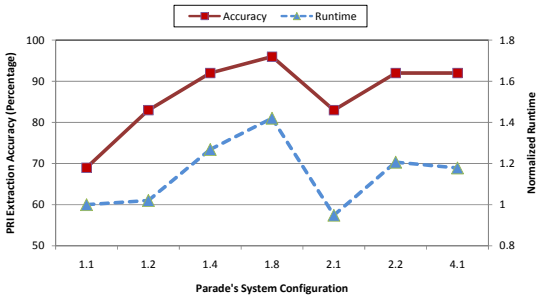

Figure 6. Accuracy and runtime of each configuration

or four module implementation provides good performance while maintaining a low cost. In accuracy-critical applications where size matters less such as ESM, an 8, or 16 module architecture would make more sense.

Figure 6 also shows that runtime can vary considerably based on the configuration. This variation is due to the smaller configurations finding incorrect multiples of the real PRIs. As can be seen, increasing the number of modules without increasing the number of memory modules has a detrimental effect on run time. This result is expected, since the number of memory read requests increases linearly with the number of main modules. Our results show that we gain a speedup of 9% from increasing the number of memory modules from 1 to 4. Here, sequential operation of the remover reduces the amount of speedup which can be achieved by increasing the number of memory modules. Therefore, we can conclude that unless the time constraint is severe or the main module to memory module ratio is extremely high, keeping a small amount of memory modules would be ideal to save on storage area.

In this work, area, power, and delay evaluations were done in 90nm technology node. A 5-stage in-order datapath using a subset of the Alpha instruction set was laid out and analyzed with Synopsys PrimePower as a comparison. In 90nm, area and peak power consumption of our Alpha pipeline are 2.15mm^2 and 771mW, respectively. Figure 7 depicts the area and power consumption breakdowns of each Parade's system configuration normalized to the Alpha pipeline's area and power consumption, respectively. The main module uses a more significant amount of area, as well as the remover and distributor logic. However, since the main modules are duplicated and only one instance of the remover and distributor is needed, the main module area dominates the total chip area in larger configurations. Main memory can either reside off-chip or on-chip. Since the algorithm implemented in this paper accesses memory locations sequentially, it is possible to utilize general caching algorithms to exploit spatial locality. As can be seen in this figure, only the *1.8* configuration comes close to the area consumed by the datapath. On the other hand, the datapath consumes a substantially greater amount of power than our design. Note that the power consumptions of all of Parade's configurations (i.e. $\leq$ 200mW) are adequate for embedded applications. In terms of clock speed, Alpha pipeline can operate at 480MHz. For the mentioned system configurations, the remover module limits the clock frequency to 540MHz. Nevertheless, as it will be discussed later, Alpha pipeline does not have enough processing power to adequately perform the deinterleaving algorithm in a real-time situation.

C. Comparison

Since Parade is the first ASIC design for solving this particular type of deinterleaving problem, here, we compare it with a recently proposed software-based deinterleaver [11].

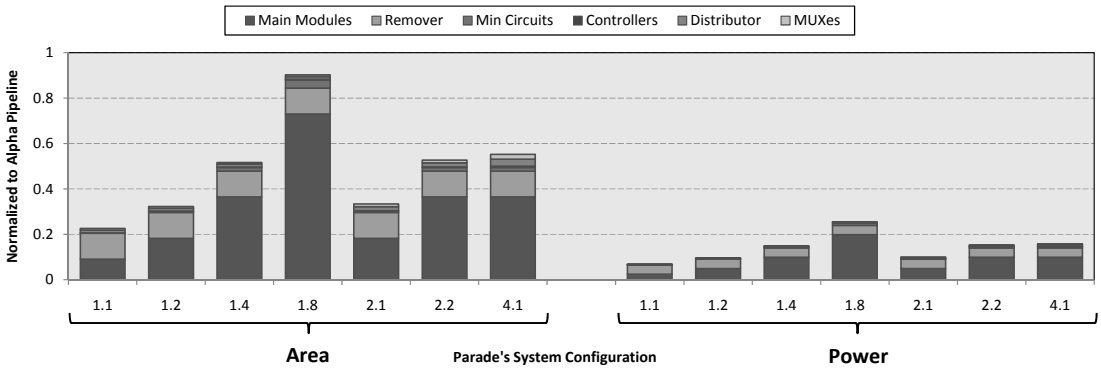

Figure 7. Normalized area and power consumption of the different system configurations

This histogram-based solution operates similar to the other Fourier transformation based methods by analyzing combined pulse-trains in the frequency domain. As it is shown in [11], for Max{PRI} = 1KHz, their proposed solution running on a Pentium III processor can deinterleave 10 pulse-trains in 17.2ms for a 100ms sampling interval. Note that the processor which is used in [11] is significantly faster than the in-order Alpha pipeline which was used for the area/power comparison.

Considering the *1.8* configuration, each pulse in average needs 8.2 cycles to be extracted. Since the clock frequency of this configuration can be as high as 540MHz, it can cluster 10 pulse-trains with PRF ≤ 6.58MHz. It should be noted that this frequency is much higher than the mentioned practical upper bound for a pulse-train PRF (i.e. 300KHz). Nonetheless, even achieving 10KHz is quite challenging for most of the previously proposed deinterleavers [3][8][16]. In terms of the mentioned method in [11], even by using a Pentium III processor, deinterleaving of pulse-trains with PRF higher than 7KHz is not feasible. Consequently, Parade performs around 940X faster compared to the mentioned histogram-based technique. Furthermore, area and power consumption of the *1.8* configuration are less than 5% of a Pentium III's area and power consumption – excluding power dissipation of bonding pads [21].

IX. CONCLUSION

In this paper, we proposed Parade, a parallel architecture which exploits an improved and parallelized sequence search algorithm for the deinterleaving of combined pulse-trains. Our proposed architecture has two main parameters, the number of memory modules and the number of main modules, which allow us to achieve an optimized combination of performance, accuracy, efficiency, and area based on the design constraints. Furthermore, Parade tackles the clustering problem in a more general case compared to the previous efforts by considering non-idealities such as dropped pulses, jitter, and arbitrary start and stop points for the pulse-trains. Based on our simulation results, our architecture achieves up to a 96% PRI accuracy with an 8-way parallel configuration, a 27% improvement over the single module baseline design. Moreover, by using less than 5% of a Pentium III's resources, Parade can perform around 940x faster compared to a soft-ware based histogram technique.

REFERENCES

[1] X. Wang, Z. Wang, "A TOA-based location algorithm reducing the errors due to non-line-of-sight (NLOS) propagation", IEEE Transactions on Vehicular Technology, pp. 112–116, Volume 52, Jan 2003.

[2] D. J. Milojevic and B. M. Popovic, "Improved algorithm for the deinterleaving of radar pulses," Proc. Inst. Elect. Eng. F, vol. 139, no. 1, pp. 98–104, Feb. 1992.

[3] T. L. Conroy and J. B. Moore, "On the estimation of interleaved pulse train phases," IEEE Trans. on Signal Processing, vol. 48, no. 12, pp. 3420–3425, Dec. 2000.

[4] H. S. Shahhoseini, A. Naseri, and M. Naderi, "A new matrix method for pulse train identification: Implementing by systolic array," in XI EUSIPCO, vol. 3, 2002, pp. 19–22.

[5] J. Perkins and I. Coat, "Pulse train deinterleaving via the hough transform," in Proc. Int. Conf. Acoust., Speech, Signal Process., vol. 3, 1994, pp. 197–200.

[6] J. J. Szkolnik, "Application des methodes de monte-carlo sequentielles a lextraction de trames radar," Ph.D. dissertation, L'Universite De Bretagne Occidentale, Nov. 2004.

[7] A. Logothetis and V. Krishnamurthy, "An interval-amplitude algorithm for deinterleaving stochastic pulse train sources," IEEE Trans. on Signal Processing, vol. 46, no. 5, pp. 1344–1350, 1998.

[8] R. J. Orsi, J. B. Moore, and R. E. Mahony, "Spectrum estimation of interleaved pulse trains," IEEE Trans. on Signal Processing, vol. 47, no. 6, pp. 1646–1653, June 1999.

[9] J. Roe, S. Cussons, and A. Feltham, "Knowledge-based signal processing for radar ESM systems", IEE Proc., vol. 137, pp. 293–301, 1990.

[10] G. P. Noone, "A Neural Approach to Tracking Radar Pulse Trains with Complex Pulse Repetition Interval Modulations", Proc. ICONIP IEEE, pp. 1075-1080, 1999.

[11] Y. Kuang, Q. Shi, Q. Chen, L. Yun, and K. Long, "A Simple Way to Deinterleave Repetitive Pulse Sequences", 7th WSEAS Int. Conference on Mathematical Methods and Computational Techniques in Electrical Engineering, pp. 218–222, Sofia, 2005.

[12] A.W. Ata'a and S.N. Abdullah, "Deinterleaving of radar signals and PRF identification algorithms", IET Radar, Sonar & Navigation, Volume 1, Issue 5, pp. 340–347, Oct. 2007.

[13] S. Hsuen-Chyun, C. Chih-Chi, L. Yueh-Jyun, L. Ching-Hai, "Radar signal clustering and deinterleaving by a neural network", IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E80-A, No.5, pp. 903-91.

[14] C.R. Gent and C.P. Sheppard, "A general purpose neural network architecture for time series prediction", In Proc. IEE ICAN'92, pp. 323–327.

[15] G. Noone, "Radar pulse train parameter estimation and tracking using neural networks", In Proc. IEEE ANEES'95, pp. 95, November 1995.

[16] T. Conroy and J. B. Moore, "The Limits of Extended Kalman Filtering for Pulse Train Deinterleaving", IEEE Trans. on Signal Processing, Vol. 46, pp. 3326–3332, No. 12, December 1998.

[17] B. E. Stenersen, "Satellite Cluster Consepts: A system evaluation with emphasis on deinterleaving and emitter recognition", Norwegian University of Science and Technology, Master's Thesis, Department of Electronics and Telecommunications, June 2006.

[18] R. Zheng, J.C. Hou and L. Sha, "Asynchronous Wakeup for Ad Hoc Networks", ACM International Symp. on Mobile Ad Hoc Networking and Computing, pp. 35–45, June 2003.

[19] H. K. Mardia, "New techniques for the deinterleaving of repetitive sequences," Proc. Inst. Elect. Eng. F, vol. 136, no. 4, pp. 149–154, Aug. 1989.

[20] P. Hansson, "Analysis of some methods for deinterleaving of pulse trains", Royal Institute of Technology, Nov. 2007.

[21] Intel Corp., Intel Pentium III Processor Datasheet, http://developer.intel.com/design/archives/processors/pentiumiii/index.htm#datasheets.