Dataflow Techniques for Exploiting Fine-Grained Parallelism

by

Yanzhou Liu

Research outcome report submitted to the Marshall Plan Foundation
via the Salzburg University of Applied Sciences, in fulfillment of the requirements
for the Marshall Plan Scholarship
2016

Table of Contents

Abstract

The time required for jitter measurement in digital communications waveforms can be dominated by computation time which increases with waveform depth. Previous work on decreasing this computation time includes the use of parallel resources on microprocessors and graphics processing units. However, the waveform depth and computation speed were limited by the need to have the entire waveform and intermediate results derived from it in memory all at once. We present a new dataflow-based method for clock recovery and time interval error (TIE) and TIE standard deviation computation. Memory usage does not grow with waveform depth, so the latter is not limited by memory size. We describe an implementation in LIDE-OCL, a tool for simplifying implementation of dataflow signal processing using multicore processors and GPUs. The resulting measurement accuracy is compared on actual measured waveforms with prior methods.

Chapter 1

Introduction

In the design of advanced digital communication systems, timing jitter measurement must often be performed on *deep waveforms* — i.e., on signals that are of relatively long duration. By jitter measurement we mean summary statistics of time interval error (TIE), the difference (in units of time) between when an feature should be found in the waveform and when it actually occurred.

There are two main purposes for capturing and measuring deep waveforms in this context: (1) to increase the likelihood of capturing rare events that can cause communication errors [1], and (2) to enable estimation of tails in jitter probability distributions, as a replacement for or to improve the accuracy of distribution extrapolation [2]. Implementations of timing jitter measurement are available in instruments such as digital oscilloscopes. However, the computation time increases with waveform depth, and so it is desirable to seek methods for faster yet still cost-effective jitter computation from deep waveforms.

To address this problem and help accelerate jitter measurement, researchers have introduced parallel algorithms for constant clock period computation. For example, [3] exploits multi-core processors such as Intel central processing units (CPUs) together with their *streaming single instruction multiple data extensions* (*SSE*) [4] instruction sets to enable fast and accurate jitter measurement. However

this design suffers from large memory requirements and high latency due to its "swallow and wallow" characteristic whereby the computation is started only after all input data has arrived and has been stored in memory. This limits the amount of signal data that can be measured, and results in high response time for engineers to start seeing measurement results.

A new jitter measurement algorithm was demonstrated in [5] that significantly improves measurement response time by partitioning the overall data set into windows and allowing jitter measurement results to be reported for earlier windows before later windows are received. This re-formulation of jitter measurement eliminates the swallow and wallow characteristic, and provides improved speed. However, a memory requirement limitation still remains: the memory required (like of [3]) is unbounded. In other words, the memory requirement grows without bound as the size of the data set is increased. This characteristic again limits the amount of signal data that can be measured, which is problematic, for example, in measuring relatively long signals or signals with high sample rates.

In this report, we build on the previous work in [3, 5], and present a new jitter measurement system design that provides both fast, real-time response and memory-efficient operation.

We demonstrate that our proposed design involves novel implementation trade-offs among measurement response time, memory requirements, and accuracy. We optimize the algorithm, including the design of the actors, to improve the latency of the application. We present a detailed experimental study where we validate the correctness of our new jitter measurement system design; demonstrate its capability

for real-time operation; investigate the underlying trade-offs among latency, memory, requirements, and accuracy; and show how the realized trade-offs can be controlled by the system designer (for example, to provide complementary highest-accuracy and fastest-response-time modes).

For design and implementation of our novel jitter measurement system, we integrate the application of Graphics Processing Units (GPUs) [6], the Open Computing Language (OpenCL) [7], and dataflow-based modeling of signal processing systems [8]. GPUs are massively parallel processors that execute large numbers of specialized computational modules, called *kernels*, concurrently to achieve improved performance in terms of throughput and latency. OpenCL is an open standard for programming applications, and executing programs on heterogeneous computing platforms, including platforms that integrate CPU and GPU devices. Dataflow-based modeling provides representations for signal processing application design that help to formally capture high level algorithmic and computational structure in a systematic way. The structure exposed by well-designed dataflow models can help to significantly enhance the reliability and efficiency of derived implementations.

A preliminary version of some of the work presented in this report has been published in [9].

In summary, the contributions of this report are three-fold. First, we present the design and implementation of a novel jitter measurement system that provides capabilities for real-time response and memory-efficient operation. Second, we investigate fundamental trade-offs among accuracy, processing speed, and memory requirements in the implementation of jitter measurement systems. Third, we

demonstrate the integrated application of GPU, OpenCL and dataflow technologies to address design challenges of high speed signal measurement applications.

Chapter 2

Jitter Measurement System Design

In our previous work, we developed dataflow modeling methods and window-based signal analysis methods to improve the efficiency of jitter measurement [5]. In Section 2.1 and Section 2.2, we provide a brief review of these methods, and of fundamental concepts in dataflow modeling. We then present the main contributions of this report, which enable real-time jitter measurement by (1) significantly improving response time, and (2) providing bounded memory requirements that are dependent on the window length rather than on the overall duration across which the jitter measurement is performed. We discuss novel GPU implementation techniques that we have applied to improve the efficiency of jitter measurement in these dimensions of response time and memory requirements.

2.1 Dataflow Modeling

Dataflow is a form of model-based design that is employed for design and implementation in many areas of signal processing [10, 8]. By providing formal methods to represent and analyze the flowgraph structures within signal processing applications, dataflow methods help to enhance the efficiency and reliability of implementations, and assist in the retargeting of designs across different hardware platforms. Dataflow methods involve representing applications as dataflow

graphs, where vertices (*actors*) represent computational tasks, and edges represent the passage of data between pairs of actors. Data passes through edges in a first-in, first-out (FIFO) fashion, and thus, the terms "edge" and "FIFO" are often used interchangeably in the context of dataflow graphs.

Connections between actors and edges are called *ports*. Each port is either an *input port* or an *output port*, depending on whether the actor consumes data from the incident edge or produces data on it. Each unit of data associated with a given dataflow edge is referred to as a *token*. Tokens can have arbitrary data types, although typically all tokens for a given edge have the same type.

Execution of dataflow actors is decomposed in terms of discrete units referred to as *firings*. During a firing, an actor consumes tokens from its input ports and produces tokens onto its output ports. The amount of data produced or consumed during a firing is referred to as the *production* or *consumption* rate, respectively, of the associated output or input port. These rates are collectively known as the *dataflow rates* of the actor. Dataflow rates can be constant, as in the synchronous dataflow (SDF) form of dataflow [11], or they may vary dynamically (i.e., from one firing to the next).

Our jitter measurement system design takes the form of a *computation graph* [12]. Computation graphs are similar to SDF, except that the consumption rate of a port can be different from the number of tokens from the associated input edge that is accessed during a firing. Tokens that are accessed but not consumed during a firing remain in the associated FIFO so that they can be accessed or consumed in subsequent firings. The number of tokens that is accessed from an edge is referred

to as the *threshold* for the associated actor port.

An important step in the implementation of a dataflow graph is the assignment of actors to processing resources, and the ordering of actors that share the same processor. This step is referred to as dataflow graph *scheduling*. The result of the scheduling step is a design component called a *schedule*, which is used to execute the actors in the graph. A wide variety of scheduling techniques have been developed based on specific constraints and objectives in different signal processing application areas (e.g., see [8]).

2.2 Window-based Signal Analysis

To help reduce memory requirements for jitter measurement computations on large input data sets, we have developed a windowing method that decomposes the jitter analysis process into fixed-size blocks of successive samples, where the block ("window") size W_s is relatively small compared to the size of the overall data set [5]. The dataflow graph can then be executed repeatedly on successive windows of the input data stream. The measurement system designer can set the window size W_s to influence an underlying trade-off between jitter measurement accuracy and memory requirements. Larger window sizes generally provide increased accuracy at the expense of increased memory cost.

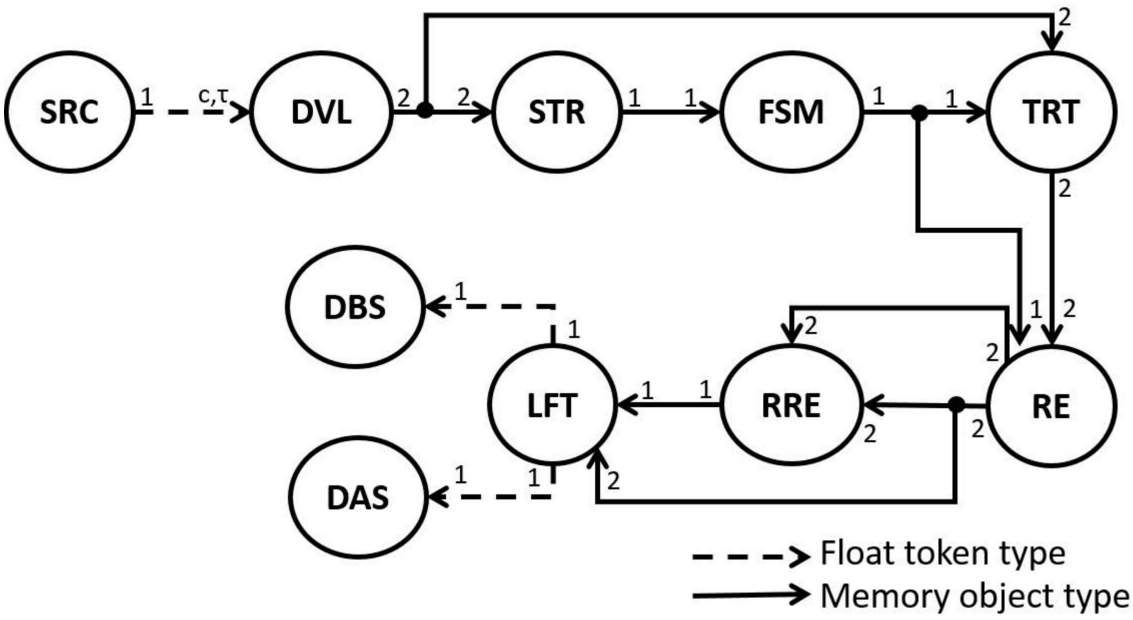

Figure 2.1: Dataflow model for our real-time jitter measurement system.

2.3 System-Level Model

The dataflow (computation graph) model of our jitter measurement system is illustrated in Figure 2.1. We implement the individual dataflow modeling components (actors and edges) in OpenCL. Details on these implementations are discussed below. The integers next to the actor ports represent the production and consumption rates associated with the ports. For all input ports except one (the input port of the DVL actor), the consumption rate and threshold are equal, so they are not shown separately. The dataflow behavior of the input port of the DVL actor is represented by the parameter pair $[c, \tau]$, where c is the consumption rate and τ is the threshold of the port. The parameter τ is the window size for the actor, and satisfies $\tau \geq c$.

2.3.1 Actor Descriptions

Here, we briefly summarize selected actors that are employed in Figure 2.1. The actors summarized here are those whose implementations have changed (compared to the system presented in [5]) due to our use of a GPU for the new measurement system.

Actor DVL (Determine Voltage Level) sorts the input data of the current window and determines the high and low voltage thresholds. Actor STR (State Representation) performs analog-to-digital conversion based on the voltage thresholds to assign low or high voltage states. Actor FSM (Finite State Machine) identifies voltage transitions from high to low or low to high voltage states. Actor TRT (Compute Transition Time) computes the transition time for every voltage transition in the current window. Actor RE (Rough Estimation) derives a preliminary estimate of the clock period. Actor RRE (Refine Rough Estimation) refines the rough estimate of the clock period to improve the accuracy. Actor LFT (Linear Fitting) further refines the clock period estimate using a linear fitting method, and computes time interval errors using the refined clock period estimate. For descriptions of the other actors in Figure 2.1, we refer the reader to [5].

In addition to changing the implementation platform to a GPU, we have made important improvements in the dataflow graph structure compared to the graph in [5]. In particular, phase computation is now implemented as part of the LFT actor instead of as a separate actor. This graph transformation is motivated from observations that (1) GPU kernels for both actors have similar levels of parallelism,

and (2) combination of the actors provides significant reduction in GPU memory requirements. In the transformed graph of Figure 2.1, the outputs of the LFT actor encapsulate the derived clock period and time interval error (TIE) for jitter estimation.

2.4 Actor Implementation

In this section, we discuss the implementation of the dataflow graph actors summarized in Section 2.3. We emphasize our optimized application of GPU features to jitter measurement tasks, and improvements incorporated in our GPU-based actor implementations compared to our previously reported system in [5].

The GPU-targeted actors in our implementation are developed using LIDE-OCL. LIDE-OCL provides an integrated software tool for implementing signal processing dataflow graphs using OpenCL. LIDE-OCL is centered on OpenCL implementations of the abstract (platform- and language-independent) dataflow programming APIs (application programming interfaces) in the lightweight dataflow environment (LIDE) [13].

2.4.1 Jitter Measurement Optimization using LIDE-OCL

In LIDE-OCL implementation, computations in an actor can be decomposed into one or more kernels with different amounts of concurrency (GPU "work group" sizes). Initialization of device and kernel configurations is performed before graph execution. Before actors and FIFOs are constructed, relevant host device and GPU

device properties, including device and command queue initialization, are set up. The *command queue* can be viewed as a dynamically-managed list of commands that have been submitted ("issued") to the GPU for execution, and are waiting to be fetched and executed by the GPU. When a GPU-targeted actor is constructed, GPU memory used by the kernels in the actor is allocated, and the kernels are dynamically loaded and compiled so that they are available to the dataflow graph schedule when the graph is executed. Once these initialization and configuration steps are completed, the dataflow graph is ready to execute.

In our implementation of jitter measurement, the deep waveform analysis computations are performed on the GPU. After a window of data is processed, the derived clock period results, and Time Interval Error (TIE) results are sent from GPU memory to the host device (CPU). Since all of the waveform analysis is performed within the GPU, parallelism be exploited extensively throughout the associated computations, and all inter-actor communication is performed within the GPU (rather than between the GPU and the host). These features help significantly to improve jitter measurement response time. The token type used by the GPU-targeted actors is a generic type associated with OpenCL objects [7]. This organization provides flexibility and efficiency in processing different kinds of data within actors on the GPU.

In the remainder of this section on actor implementation, we provide details on how efficient parallel execution of jitter measurement computations is achieved on the targeted GPU platform.

2.4.2 DVL and RE

Efficient sorting of numeric values is important in our jitter measurement system. For example, in the DVL actor shown in Figure 2.1, a sorting algorithm is applied to determine thresholds for high and low voltage values. Similarly, the RE actor operates by sorting the differences between neighboring transition times.

We apply a sorting algorithm called *bitonic sort* to accelerate the sorting operations required for jitter measurement calculation. Bitonic sorting was applied originally in the construction of sorting networks [14], which can be viewed as interconnections of comparators and wires that are used to sort collections of values. It is well known that bitonic sort is useful for its utility as a parallel sorting algorithm. We apply this feature to derive fast implementations of the DVL and RE actors on the targeted GPU.

Figure 2.2 illustrates our design of the RE actor.

2.4.3 RRE and LFT

The RRE and LFT actors involve computing sums over large numbers of data values. The associativity of the addition operator allows for use of efficient GPU implementation techniques that are based on parallel computation methods for *reduction operations* [15]. However, due to the large volumes of data involved, reduction techniques must be applied carefully in our implementation to optimize performance. For example, non-local synchronization of data between every component operation within a reduction operation is costly. In OpenCL-based GPU

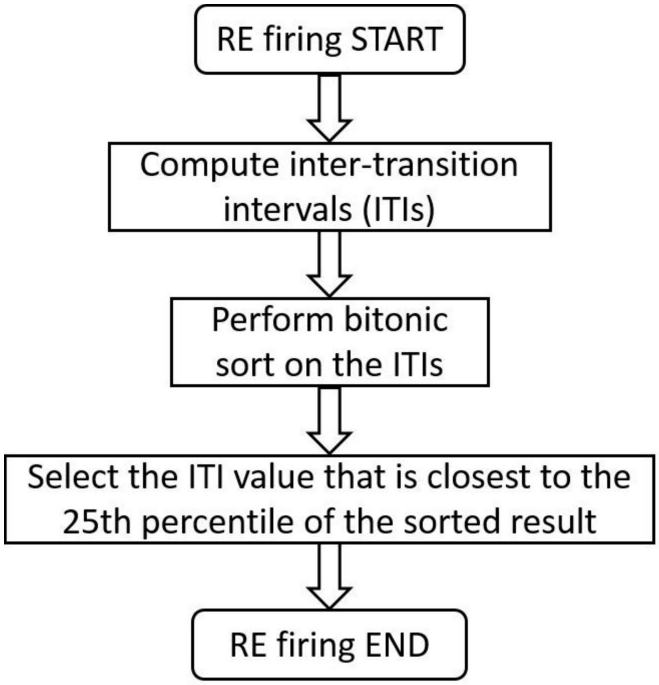

Figure 2.2: Design of the RE actor.

implementation, relatively costly non-local synchronization arises when the data involved in an operation crosses the boundary of a set of operations called a *local work group*. Thus, we structure the summations in the RRE and LFT actors such that they are reduced first at the level of local work groups. This makes maximal use of local memory operations (fast synchronization) during the reduction process.

Figure 2.3 illustrates our application of reduction methods to summation operations. In the example of Figure 2.3, the total data length (number of values to be added) is 16, and the local work group size is 4.

2.4.4 TRT

Another computationally-intensive process in our jitter measurement system is the *stream compaction* [16] of the transition time array in the TRT actor. Stream

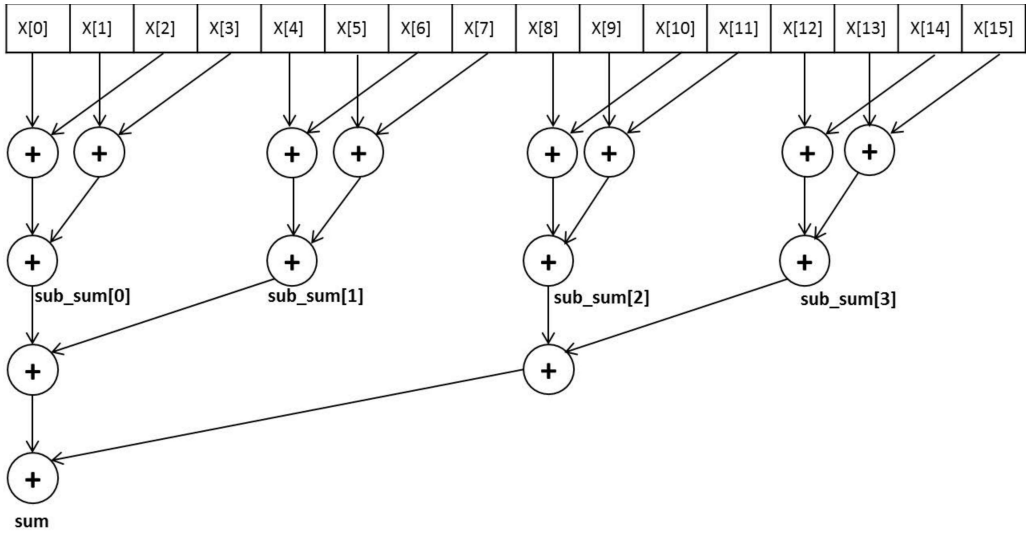

Figure 2.3: Illustration of reduction methods applied to a summation operation.

compaction refers to the process of compressing the storage requirements of a sparse array by removing zero-valued elements from the array. In each jitter measurement window, transitions are detected at switching points between high and low voltage levels, the corresponding transition times are computed, and these transition times are stored by setting array elements to non-zero values at array indices that correspond to the transition time intervals. This approach is efficient for initial computation and storage of the transition times, but it results in a sparse array that is costly in terms of memory.

Thus, within our implementation of the TRT actor, we compress the sparse transition time array using a prefix sum [17] technique. This stream compaction process is illustrated in Figure 2.4 with a simple example involving 8 data items.

In OpenCL, a prefix sum computation on a large data set can be implemented in a manner similar to a reduction operation. Using such an approach, we compute the prefix sum for the local work groups in parallel. Then we compute the prefix sum

Figure 2.4: An illustration of stream compaction, which is employed in the TRT actor.

of those partial prefix sums. The result of this intermediate prefix sum operation is called the "summation offset". The final prefix sum result is then computed by adding the summation offset to the results computed on the local work groups.

2.5 Schedule for Dataflow Graph Execution

As is discussed in Section 2.1, a schedule is needed to execute the dataflow graph on a targeted hardware platform. Many possible schedules can be constructed for the dataflow graph of Figure 2.1. The alternative schedules in general have different implementation costs in terms of relevant metrics. For the experiments in Section 4, we derive a *static schedule* that employs a heuristic to decrease the CPU and GPU memory requirements. A static schedule is one in which the assignment and ordering of all actors are fixed before the graph is executed. Static schedules are useful for reducing the run-time overhead of schedule execution, and for improving

the predictability of the implementation [8]. Constant-valued dataflow rates and thresholds allow for the construction of static schedules, and we exploit this property of our dataflow graph model when constructing the schedule.

The static schedule that we employ in our implementation can be expressed as

$$(W_s\ SRC)\ DVL\ STR\ FSM\ TRT\ RE\ RRE$$
$$LFT\ DBS\ DAS,$$

where the parenthesized term $(W_s\ SRC)$ represents the successive execution W_s times of the SRC actor. Recall from Section 2.3 that W_s represents the window size for jitter measurement.

2.6 Real-time versus Accumulated Implementation

We have developed two kinds of implementation for the targeted jitter measurement system. We refer to these as the *accumulated* version, and *real-time* version. In the accumulated version, transition times from the first input window are accumulated, and clock period estimation is based on all transitions found from the first window to the current window. The voltage thresholds are fixed in the first window, and these fixed thresholds are used for all subsequent windows.

On the other hand, in the real-time version, clock estimation in each window is regarded as a complete and independent clock recovery process. That is, results

from the current window are independent of previous windows, and the voltage thresholds are computed separately in each window. In this implementation, the memory requirement is independent of the number of windows processed, which provides great flexibility for continuous operation of the system.

Chapter 3

Optimization on the Jitter Measurement System Design

The real-time system design for jitter measurement that we presented in the last chapter provides a memory-efficient design. When this design is applied to a measurement system that samples data in real time, the system response time for each window of samples should be no slower than the time required to acquire the samples in the window (the reciprocal of the sampling speed for the window). In the targeted real-time measurement application, data is retrieved from a DAQ (data acquisition) board, which collects data from a real-time input source. If the processing speed of the implementation is slower than the DAQ sampling rate, then the data buffer on the DAQ subsystem will overflow as increasing numbers of windows are processed. Therefore, to support unbounded input streams in the measurement application, our system design needs to be optimized based on the constraint described above on the response time.

In this chapter, we will examine our jitter measurement system design as a case study to demonstrate design optimization using LIDE-OCL. We discuss transformations applied to the dataflow graph, and optimization performed on selected actor implementations to help achieve the latency constraint imposed by the sampling rate of data acquisition.

3.1 Dataflow Graph Design Optimization

In the design of the jitter measurement system discussed in the previous chapter, the final clock period is estimated for a single window in each graph iteration. Based on the final clock period, the TIE (time interval error) of each transition in the window is computed as well. The number of TIEs in one window is equal to the number of transitions in the window. The final clock period and the computed TIEs are the output results for the window. The number of TIEs is roughly equal to W_s/CP, where W_s is the window size and CP is the clock period. Therefore, increasing the window size increases the number of TIEs in the window.

Storing to disk large numbers of TIEs in this way significantly slows down system performance. To address this problem, we have designed an actor `TIESD`, which first computes the standard deviation of all TIEs in a window first, and then outputs this standard deviation (a single number) to the result file rather than outputting all TIEs. Therefore, in the optimized dataflow graph, the results in each iteration are the final estimated clock period and the standard deviation of TIEs in the window. The updated dataflow graph is shown in Figure 3.1. Here, f_{num} (the production and consumption rates on the output edge of the `DAQSRC` actor) is the number samples retrieved from the DAQ board to construct one window of jitter measurement input.

In this optimized dataflow graph, the `DAQSRC` actor retrieves data from the DAQ subsystem, and the `DAQTRF` actor converts the collected data from digit values to floating point format, and transmits the data to GPU memory for GPU-based

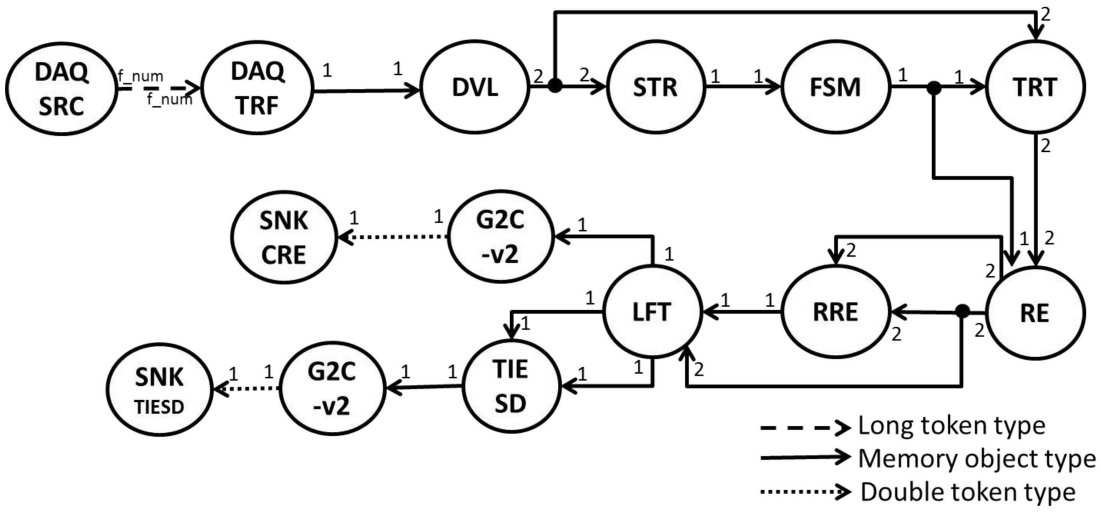

Figure 3.1: Improved dataflow model for our real-time jitter measurement system.

computation with the following actors. The `TIESD` actor computes the standard deviation of the TIEs in the current window. The `SNKCRE` and `SNKTIESD` actors are the sink actors for the final clock period and standard deviation of the TIEs, respectively. The other actors in Figure 3.1 have the same functionality as the corresponding actors in Figure 2.1.

3.2 Optimization on Actor Design

In this section, we discuss how to optimize selected actors in Figure 3.1 to further improve system latency.

In our jitter measurement system, sorting is an important procedure in two of the key actors — the `DVL` and `RE` actors. In the `DVL` actor, the input data is first sorted and then a voltage threshold is derived from the sorted data to quantize each sample to a binary value. In the `RE` actor, sorting is applied to order the transition time intervals so that a rough estimation of the clock period can be determined.

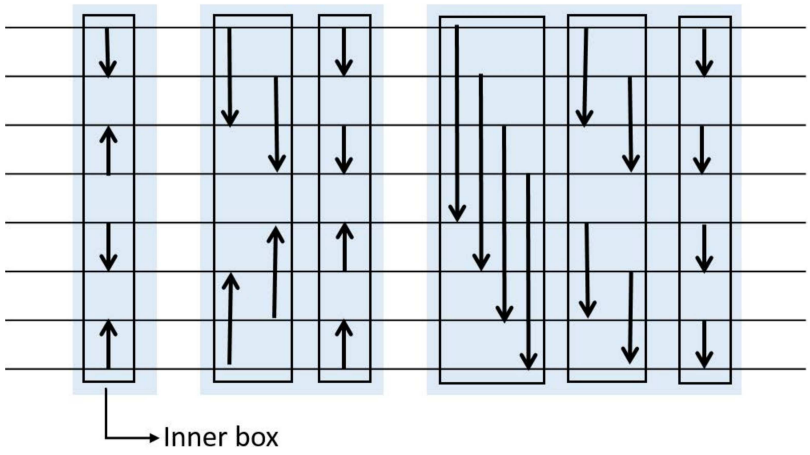

Figure 3.2: An illustration of the bitonic sort algorithm.

In both of these actors that employ sorting, bitonic sort is applied to parallelize the sorting process. However, the transformed sorting process is not fully parallelized, which means that increasing amount of data to be sorted leads to an increase in execution time. The performance of the sorting algorithm becomes a bottleneck of the overall jitter measurement system when increasing the window size to improve the system throughput. In Section 3.2.1 and Section 3.2.2, we discuss how to optimize the sorting process for these two actors.

The bitonic sort network used in the applied sorting algorithm is shown in Figure 3.2. In this network, the computation represented by each inner box in Figure 3.2 is fully parallelized, while the different inner boxes are executed sequentially. Therefore, increasing the number of data items for sorting can be expected to result in a linear or almost-linear increase in execution time.

3.2.1 Optimization of the DVL Actor

The `DVL` actor operates by sorting its input data, and selecting two values from the sorted data. From a given period of the periodic input waveform, high and low voltage thresholds can be recorded. As we increase the window size, the number of periods encompassed by the window increases given our assumption that the input signal has a constant frequency.

To improve the performance of the `DVL` actor, we observe that voltage thresholds can be derived with acceptable accuracy without sorting all of the input data in the a given window. To exploit this observation, we apply an optimization that randomly selects a proper subset S of the input data for sorting, and then derive the voltage thresholds using the result of sorting S. To ensure sufficient accuracy, the number samples in S should be sufficiently large. This number should be selected carefully because selecting too large a value (overdesigning for accuracy) will slow down performance.

3.2.2 Optimization of the RE Actor

As previously discussed, the `RE` actor computes transition time intervals between pairs of successive transitions. The maximum vectorization degree for this computation is $(W_s - 1)$, where W_s is the window size. However, using a static vectorization degree based on this maximum value can result in excessive numbers of threads and excessive amounts of synchronization overhead. Thus, to further optimize our GPU-based `RE` actor implementation, we dynamically configure the

vectorization degree of the actor based on the number of transitions that are computed for a given window of data. This ensures that the vectorization degree is matched dynamically to the amount of data parallelism available in a given window of data.

The `RE` actor sorts all of the derived time intervals and then selects the 25^{th} percentile of the sorted intervals as the rough estimated clock period. Experimentally, we find that most of the time intervals are clustered close to this rough estimate. The transitions with significant jitter are the ones that deviate significantly from the rough estimate.

As an additional optimization for our `RE` actor implementation, we apply a random sampling technique similar to the one described in Section 3.2.1. This reduces the amount of data that needs to be sorted in the actor `RE`, thereby reducing the computational load. The specific size of the data subsets selected for sampling is determined experimentally to achieve an effective trade-off between accuracy and computational cost.

Chapter 4

Experimental Verification

In this section, we present an experimental validation of the proposed method. Actual measured data were used: a two-state digital waveform representing bits from a PRBS sequence measured using a deep-memory digital oscilloscope. A fuller description of the waveform and the measurement apparatus used to acquire it appear in [5]. Also in that reference, there were proposed two criteria for judging the correctness of a measurement algorithm that is transformed in order to meet software engineering goals such as increased throughput, decreased latency, or decreased memory consumption:

- Intermediate values obtained using both the old and new algorithms should be comparable, and
- The difference between the final measurement results of the old and new algorithms should be small compared to the total measurement uncertainty in either.

Here, we compare the measurement results of [3] and [5] with those of two variants of the proposed method, which we will refer to as the accumulated and real-time variants. The same actual measured data is used for the comparison.

In the accumulated variant, the transition times from the very first window are accumulated in a buffer and the clock period estimation is based on all transitions

found from the first window to the current window. Voltage thresholds are fixed based on the voltage statistics found in the first window. Thus, the first window needs to contain at least one logic state transition. The accumulated variant needs only to store one array that grows with waveform depth: the transition times. Thus, it has lower memory consumption that the methods of [3] and [5] but nevertheless can run only for some finite time before computer memory is exhausted.

In the real-time variant, clock recovery in every window is regarded as a complete and independent process. The voltage thresholds, transition locations, clock recovery, and time interval errors within each window are computed independently of all other windows. This variant has memory requirements that are fixed, independently of the amount of time it runs: the amount of time it can run is not memory-limited. By "real-time" we refer to the ability of this method to operate successfully on a temporally unbounded waveform. The term is not to be understood as a claim to be able to measure at the full, many-gigasample per second, sampling rate of contemporary instruments.

Both variants were implemented in LIDE-C and in LIDE-OCL. The two implementations produced identical results to 8 decimal places, and so are identical for all practical purposes. This validates the correctness of the LIDE-OCL implementation. The LIDE-OCL results are used in the following graphs.

The accumulated variant reports the desired statistic about jitter, the standard deviation of time interval error (TIE), only after processing the entire waveform. Table 4.1 shows the value of that statistic obtained using the accumulated variant for various window sizes and reported in [3] and [5] for the same waveform. All of

the results are within 0.02 of a sample time, under a half a percent of relative error.

Validation of the real-time variant is more complicated because it produces a value of standard deviation of TIE for each window. And, as shall be seen below, the statistical behavior of the measurement results varies considerably with the window size. The same measured waveform was provided to the real-time variant, once for each of a number of window sizes. Figure 4.1 summarizes the statistical distribution of the key intermediate computed value, the waveform unit interval (UI). Figure 4.2 shows the standard deviation of TIE found for each window for each window size. Figure 4.3 summarizes with a box plot the statistical distribution of the standard deviations of the previous figure.

These figures show that the UI estimates and standard deviation of TIEs of the real-time variant converge toward those of the accumulated variant and of [3] and [5]. However, there is a significant frequency of high measurement errors (shown with the outlier circles in Figures 4.1 and 4.3) at lower window sizes. One the other hand, the measurement results are visually as accurate with windows of 131072 samples as they are when processing the entire waveform of over 14 million samples—yet the windowed version requires approximately 1% of the memory.

Table 4.1: Standard deviation of TIE: Accumulated variant

window size (samples)	[3]	[5]	8192	16384	32768	65536	131072
std dev of TIE (sample times)	4.650	4.647	4.651	4.649	4.651	4.651	4.648

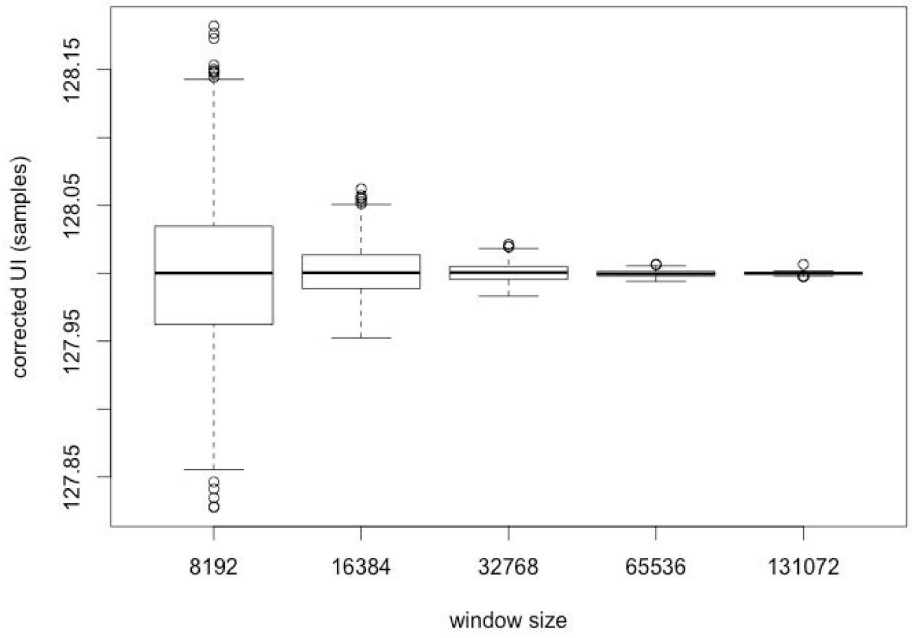

Figure 4.1: Box plot showing distribution of corrected UIs as a function of window size found by the LIDE-OCL implementation of the real-time variant, given the actual measured data described in [5]. The known *a priori* correct value is 128.00. Boxes extend from the 25th to the 75th percentile. The median corrected UI is shown as a thick black horizontal line. Outliers are shown as circles. One extreme outlier was deleted for window size 8192.

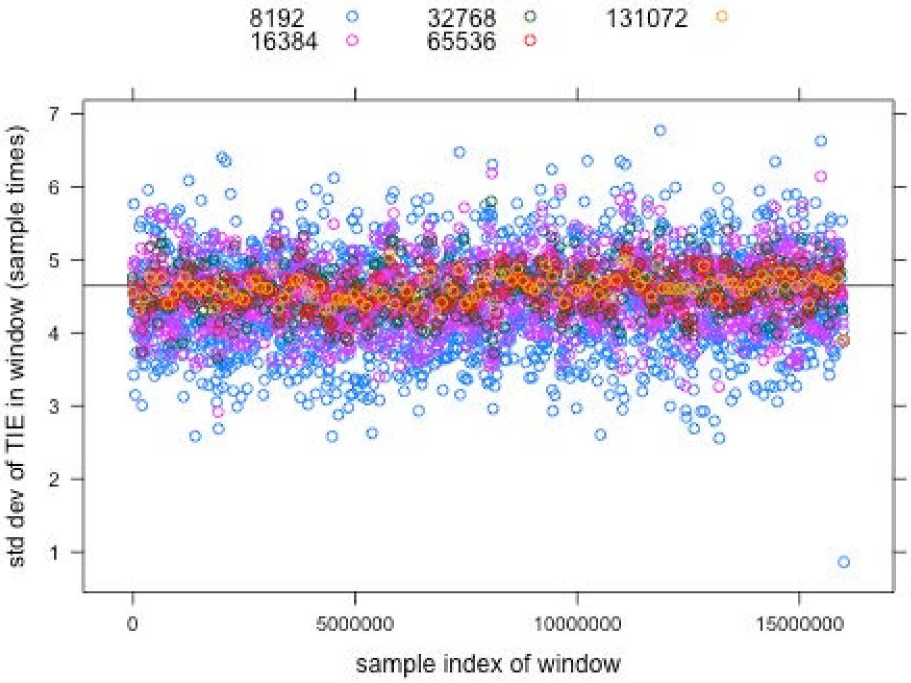

Figure 4.2: Evolution of standard deviation of TIE as windows are processed in the real-time variant. Colors indicate number of samples per window. The x-axis indicates the final sample number of each window. Horizontal black line is at the standard deviation of 4.65 samples obtained by the accumulated variants.

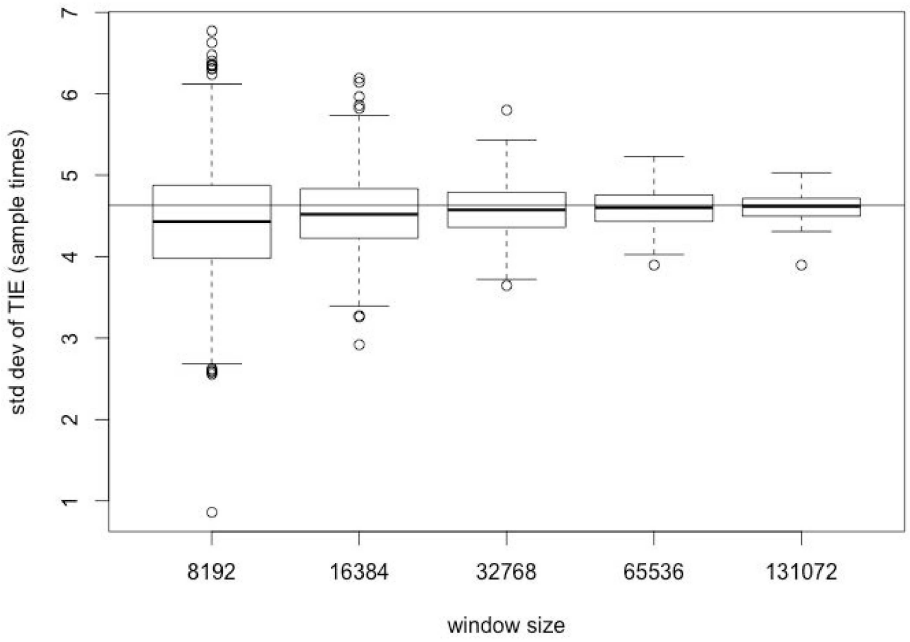

Figure 4.3: Fig. 4.2 summarized as a box plot. The horizontal line passing through the entire figure shows the standard deviation of 4.65 sample times obtained by the accumulated variants.

Chapter 5

Conclusion and Future Work

We have developed a novel jitter measurement system with applications to design and evaluation of digital communication systems. By integrating methods in window-based jitter analysis algorithms, dataflow-based system-level modeling, OpenCL programming, and GPU implementation, we have demonstrated that our measurement system achieves memory efficient, real-time operation. We have also exposed and investigated system-level design trade-offs among computation accuracy, memory requirements and latency in jitter measurement. Finally, we have validated our new jitter measurement system's consistency with related previous systems by verifying that it produces both intermediate computed values and final measurement results that converge to within sub-percent measurement errors compared to two previous systems. Useful directions for future work include optimization of the dataflow graph actors and the schedule to further improve the execution speed of the implementation.

Chapter 6

Personal Report

I have studied in Salzburg University of Applied Sciences for four months. During the four-month study, I continued my research on the model-based design of the measurement system application. Faculty, staff and students in this university have impressed me a lot for their expertise and enthusiasm on the research. I strongly recommend graduate students in America apply for the MPS scholarship and enjoy the exchanging experience in the Salzburg University of Applied Sciences.

6.1 Overall Impression of the Research Period

During my stay in Salzburg University of Applied Sciences, I have been impressed a lot for my research experience in this University.

6.1.1 Support from the International Office

During my application for the MPS scholarship, the staff in the international office helped me a lot on how to complete the application including answering the questions I had when applying and providing effective information on the scholarship application. I appreciate a lot for their friendly help and timely assistance.

After I received the offer from MPS foundation, they positively contacted me on my preparation for coming and joining the university this year. Moreover,

they created a Facebook group for the incoming students and volunteers from the university so that all incoming students had the opportunity to communicate with current students in the university and better know about Salzburg and Austria before arriving. Before the beginning of the semester, I was invited to attend the welcome week for all incoming students. Plenty of events are provided in this week to help us be familiar with the Salzburg city and the university itself. This effectively help students adapted to the new life in Salzburg faster.

During the study and research in the Salzburg University of Applied Sciences, the international office has provided a series of events such as visiting a famous salt mine in Salzburg and hiking. This enriched our life in Austria a lot. The service and instructions provided by the international office guided us know Salzburg and studied in Salzburg better. They did make a lot of efforts to let us understand and learn the culture in Austria.

6.1.2 Support and Communication with IT Department

I am a graduate student majoring in electrical and computer engineering. Faculty in Department of Information Technology & Systems Management is my advisor during the exchanging study. All staff and faculty in the department are very friendly and helpful. Due to my research project requirement, some equipment and software are necessary for my research project. When I discussed my project with them before my arrival, they supported a lot on preparing the equipment and software facilities for my research study. After my arrival, staff in the department

introduced the whole university and department to me with enthusiasm. To help me more involved in the department events, I was invited to join the regular meeting of the department and several important public events this semester. These events let me have a better understanding for the department culture and research projects it has. I was also invited to participate in the regular meeting of one research project. Discussion for the project in that semester expand horizons on different research areas.

6.2 Quality of the Research Institution

The overall quality of Salzburg University of Applied Sciences is pretty good. Although the university doesn't cover majors for all different subjects, the majors it has are very practical and largely required in the job market.

Campus of Salzburg University of Applied Sciences is perfectly formed though small. Facilities like library, printing service and canteen are provided as well as those outdoor facilities such as a mini beach volleyball court.

Although most of the departments in the university are in the same building, Department of Information Technology & Systems Management is able to provide several labs for different experiments' requirements. Besides, there are lab assistants coordinating lab-related preparation including hardware and software facilities. Lounge and discussion areas make it more convenient for students and faculty to discuss and talk about the research.

All in all, the quality of the research institution is satisfactory and full of

surprise.

6.3 Integration in the research institution/organization

During my study in Salzburg University of Applied Sciences, I have attended most events invited by the IT department. The following is what I have done to be integrated to the research in the IT department.

There are a lot of research projects in the IT department. My research advisor who is in charge of several projects invited me to join in the regular meeting of one project every week. The discussion in the meeting promoted the understanding and communication with each other. I have benefited a lot from these meetings. Besides that, the department regular meetings I joined were given me a picture of working and research style in Europe. During the semester, I was lucky to attend a seminar about MATLAB hosted by IT department and joined in several events for propagating the research projects and showing the research projects in the departments to the potential students and people from the industries. Furthermore, I was also invited to do a presentation introducing what I have done during the study in Salzburg. I first introduced my research lab in home university and software tools developed by my research group. After that I introduced and discussed about my research topic and what I have done in these four month study. Such presentation and further discussion did help us know about the research with each other much better and make me involved in the department better.

6.4 Recommendations for future MPS scholars/fellows

First of all, I strongly recommend future potential students apply for the MPS scholarship. Furthermore I also encourage students to choose Salzburg University of Applied Sciences as their host university.

Austria locates at the center of Europe. Therefore it is much easier for students to visit other Europe countries. On the other side, culture and history in Austria could represent the Europe historical process to some extent. Salzburg, an important city in Austria will give a basic picture of Austria history as well.

Another aspect is the education system. Education system in Austria is quite different from what USA has. Experience of studying in Austria helps students expand their horizons and know more about the whole world. Students will benefits a lot from the education system in Austria.

Last but not least, there are opportunities that students could participate in research projects in the university. One feature of the research project in Salzburg University of Applied Sciences is that they frequently collaborate with companies in the industry. Students could practice their skills in developing such research project and discuss with people in the company which is also helpful for their future career plan.

6.5 Contact information (email address) after the research period

Contact information after the research period: yzliu@umd.edu

Bibliography

[1] D. Murray, "Using RF recording techniques to resolve interference problems," in *Proceedings of AUTOTESTCON*, 2013, pp. 1–6.

[2] Wolfgang Maichen, *Digital Timing Measurement: From Scopes and Probes to Timing and Jitter*, chapter 9.3, Springer, 2006.

[3] Torbjorn Loken, Lee Barford, and Frederick C Harris, "Massively parallel jitter measurement from deep memory digital waveforms," in *Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International*. IEEE, 2013, pp. 1744–1749.

[4] Intel, "Intel 64 and ia-32 architectures software developers manual volume 3 (3a, 3b & 3c):system programming guide," pp. 1–68, 2015.

[5] Y. Liu, L. Barford, and S.S. Bhattacharyya, "Constant-rate clock recovery and jitter measurement on deep memory waveforms using dataflow," in *Instrumentation and Measurement Technology Conference (I2MTC), 2015 IEEE International*, May 2015, pp. 1590–1595.

[6] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[7] Khronos OpenCL Working Group et al., "The opencl specification," *version*, vol. 1, no. 29, pp. 8, 2008.

[8] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, Springer, second edition, 2013, ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[9] Y. Liu, L. Barford, and S. S. Bhattacharyya, "Jitter measurement on deep waveforms with constant memory," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference*, Taipei, Taiwan, May 2016, pp. 1161–1166.

[10] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, pp. 773–799, May 1995.

[11] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[12] R. M. Karp and R. E. Miller, "Properties of a model for parallel computations: Determinacy, termination, queuing," *SIAM Journal of Applied Math*, vol. 14, no. 6, November 1966.

[13] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya, "A lightweight dataflow approach for design and implementation of SDR systems," in *Proceedings of the Wireless Innovation Conference and Product Exposition*, Washington DC, USA, November 2010, pp. 640–645.

[14] Kenneth E Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. ACM, 1968, pp. 307–314.

[15] Yan Shengen, Zhang Yunquan, Long Guoping, et al., "Reduction algorithm optimization based on the opencl," *Journal of Software*, vol. 22, no. S2, pp. 163–171, 2011.

[16] Markus Billeter, Ola Olsson, and Ulf Assarsson, "Efficient stream compaction on wide simd many-core architectures," in *Proceedings of the conference on high performance graphics 2009*. ACM, 2009, pp. 159–166.

[17] Mark Harris, Shubhabrata Sengupta, and John D Owens, "Parallel prefix sum (scan) with cuda," *GPU gems*, vol. 3, no. 39, pp. 851–876, 2007.