DISTRIBUTED ARRAY OF SYNCHRONIZED SENSORS AND ACTUATORS

D. Budnikov, I. Chikalov, I. Kozintsev and R. Lienhart

Intel Labs, Intel Corporation
dmitry.budnikov(igor.chikalov)(igor.v.kozintsev)(rainer.lienhart)@intel.com

ABSTRACT

We propose to use a set of general purpose computing (GPC), communication and multimedia devices such as laptops, tablets, PDAs, smart phones, audio recorders, and camcorders as a distributed array signal processing system. A novel scheme is developed to synchronize input and output sampling for a network of distributed multi-channel audio sensors and actuators. IEEE 802.11 wireless network is used to deliver the global clock and data streams to distributed GPC platforms, while an interrupt timestamping mechanism is employed to distribute the common clock between I/O devices. Adaptive statistical processing is applied to reduce the effect of timing errors inevitable on real-life platforms. A prototype of distributed array is implemented based on Intel®Centrino™laptops and experimental results demonstrate a precision in A/D and D/A synchronization better than 50 μs (a couple of samples at 48 kHz).

1. INTRODUCTION

Arrays of audio/video sensors and actuators such as microphones, cameras, loudspeakers and displays along with array processing algorithms offer a rich set of new features for emerging applications. Until now, array processing required expensive dedicated multi-channel I/O cards as well as costly high-throughput computing systems due to the requirement to process all channels on a single platform. Recent advances in mobile computing and communication technologies, however, suggest a novel and very attractive platform for implementing array processing algorithms. Students in classrooms and co-workers at meetings are nowadays accompanied by several mobile computing and communication devices with audio and video I/O capabilities onboard such as laptops, PDA's, and tablets. In addition, high-speed wireless network connections, like IEEE 802.11a/b/g, are available to network those devices. Such ad-hoc sensor/actuator networks can enable emerging applications based on synchronized multi-stream audio and video I/O such as smart audio/video conference rooms, meeting recordings, automatic lecture summarization, hands-free voice communication, speech enhancement and object localization. No dedicated infrastructure in terms of the sensors, actuators, multi-channel interface cards and computing power is required. Instead multiple GPC platforms along with their sensors and actuators cooperate on providing transparent synchronized I/O to applications. However, there are several important technical and theoretical problems to be addressed before the idea of using those devices for array DSP algorithms can materialize in real-life applications. One of the most important problems is to provide a common reference time to a network of distributed computers and their I/O devices.

To illustrate the importance of time synchronization we implemented a Blind Source Separation (BSS) algorithm published in [1]. In the simplest setting two sound sources are separated using the input of two microphones, each connected to a different laptop. However, without synchronization of A/Ds the BSS algorithm failed to perform separation. Figure 1 demonstrates how a difference of only a few Hz in audio sampling frequency between two channels (laptops) impacts source separation. On the x-axis the sampling difference in Hz between two audio channels at about 16 kHz is shown against the achieved signal separation gain by BSS on the y-axis. As can be seen in Figure 1, a difference of only 2 Hz at 16 kHz reduces the signal separation gain from 8.5 dB to about 2 dB only. In real life the difference in sampling frequency can be even higher as we illustrate in Table 1. BSS is not the only algorithm that is extremely sensitive to sampling synchronization. Other applications that require similar precision of time synchronization between channels are acoustic beamforming and 3D audio rendering.

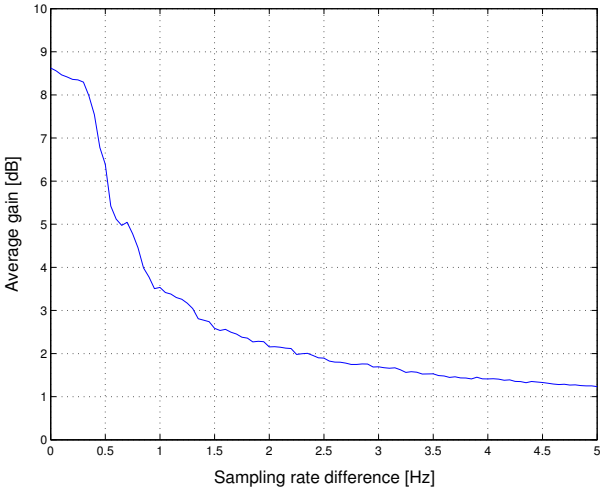


Figure 1: Sensitivity of acoustic source separation performance to small sample rate differences. Channel 1 is assumed to sample at 16 kHz, while channel 2 is assumed to sample at 16000+x Hz. Signal separation gain is calculated for the Blind Source Separation algorithm in [1].

The problem of time synchronization in distributed computing systems has been discussed extensively in the literature in the context of maintaining clock synchrony throughout large geographic areas. Each process exchanges messages with its peers to determine a common clock. Seminal works have been reported in [2] and [3]. However, the results provided there can not be applied directly to our problem, since the precision of time synchronization is too low. NTP, the Network Time Protocol, currently used worldwide for clock synchronization in the best case achieves synchronization in the range of milliseconds - 2 to 3 orders of magnitudes higher than the microsecond resolution needed for our application scenarios. The Global Positioning system (GPS) provides a much higher clock resolution. Its reported time is steered to stay always within one microsecond of UTC (Coordinated Universal Time). GPS, however, only works reliably outdoors and thus does not completely fit our application scenario. There is also some recent work on synchronization in wireless sensor networks. In [4, 5], the reference-broadcast synchronization method is introduced. In this scheme, nodes send reference beacons to their neighbors based on a physical broadcast medium. All nodes record the local time at which they receive the broadcasts (e.g., by using the RDTSC instruction of the Pentium ® processor family; the Read-Time Stamp Counter counts clocktics since the processor was started). Based on the exchange of this information, nodes can translate each other's clock. Although promising, the worst case performance of 150μs reported in [4] is too high for our application scenario. Our system

Laptop	Dell Inspiron 7000	IBM ThinkPad T20	IBM ThinkPad 600E	IBM ThinkPad T23
Sampling rate, Hz	16001.7	16003.6	16001.8	16009.5

Table 1: Audio sampling rates of several laptops.

is similar in spirit but we rely on additional processing to reduce errors in estimation of synchronization parameters.

In general, all clock synchronization algorithms studied in the literature only address the problem of providing a common clock to distributed computing platforms. They do not address how the I/O can be synchronized with the common clock (we proposed one solution for audio input synchronization in [6]). In other words, even under the assumption of a perfect clock on each platform, there is still a mechanism required to link the common clock to the data in the I/O channels [1]. On a GPC this is a challenge by itself and we address this problem in this work. Our paper extends the solution in [7] by introducing the adaptive statistical processing used to provide smooth monotonic and precise distributed clock.

The rest of the paper is organized as follows. Section 2 states the synchronization problem and describes the linear transition model used for inter- and intra-platform synchronization. Section 3 contains a brief description of data flows and timing relationships on a typical GPC platform. Sections 4 and 5 describe the inter- and intra platform synchronization algorithms respectively. In Section 6 we present important details about the practical implementation of the distributed audio sensor/actuator array and experimental results.

2. PROBLEM FORMULATION AND CLOCK MODEL

In the following we assume that each GPC platform contains a local platform clock (e.g., RDTSC) and a number of input and/or output devices. Let t be an absolute physical time and let $\theta^{Ref}(t)$ be the value of reference clock at time t. We use $\tau_{i,j}(t)$ to denote the number of samples since the start of the I/O produced by j-th A/D (or consumed by D/A) converter on i-th GPC at time t. The problem that we address in this paper can be formulated as the problem of finding the estimate $\hat{\theta}_{i,j}^{Ref}(\tau_{i,j})$ (the global time stamp of audio sample τ_i) that minimizes the difference $|\hat{\theta}_{i,j}^{Ref}(t) - \theta^{Ref}(t)|$ on a chosen time interval.

We tackle the problem of distributed I/O synchronization in two steps: (1) (inter-platform) the local CPU clocks of the GPCs are synchronized against a reference clock, and (2) (intra-platform) I/O is synchronized against the local clocks and thus also against the global clock. Note that two different timing models are required since the I/O devices on a typical PC platform have their own internal clocks that is not synchronized to other platform clocks such as the RDTSC.

Let $\theta_i(t)$ [2] be the value of i-th GPC local clock at time t. Inter-platform and intra-platform synchronization require finding estimates $\hat{\theta}_i^{Ref}(\theta_i)$ (convert value of local clock to global time) and $\hat{\theta}_{i,j}(\tau_{i,j})$ (convert sample number to local time stamp), respectively. Estimation process is required to satisfy several important constraints. Firstly, we impose clock monotonicity constraint:

$$\hat{\theta}_i^{Ref}(\theta') \geq \hat{\theta}_i^{Ref}(\theta'') \tag{1}$$

for $\theta' > \theta''$. Additionally, some applications also require smoothness. Therefore, we limit the magnitude of the second derivate to be smaller than a given constant K:

$$|\frac{\partial \hat{\theta}_i^{Ref}(\theta)}{\partial^2 \theta}| < K \tag{2}$$

at any time (but initialization).

We propose piecewise-linear models both for inter- and intra platform synchronization clock models. Therefore we need to find the model parameters a_i and b_i such that

$$\hat{\theta}_i^{Ref}(\theta_i) = a_i(\theta_i)\theta_i + b_i(\theta_i), \tag{3}$$

and $\alpha_{i,j}$ and $\beta_{i,j}$ such that

$$\hat{\theta}_{i,j}(\tau_{i,j}) = \alpha_{i,j}(\tau_{i,j})\tau_{i,j} + \beta_{i,j}(\tau_{i,j}). \tag{4}$$

The dependency of the model parameters on time approximates instabilities in the clock frequency due to temperature variations and other factors. In practice, these instabilities are in the order of 10^{-5}. This time dependency also signifies the need for their periodic update. Note that the smoothness and monotonicity constraints for distributed clocks can be violated at the moment of model update (as it happens for example in IEEE 802.11 time synchronization function - TSF). Therefore is a special model update procedure (both for inter and intra-synchronization) is illustrated in Figure 2. Let a_i, b_i be current model parameters and a_i', b_i' be updated parameters at time θ_i^0. At first, a range in local clock values $[\theta_i', \theta_i'']$, $\theta_i^0 < \theta_i' < \theta_i''$ is chosen for transition from the "old" model to the "new" model. Then, parameters of transition model a_i'', b_i'' are calculated as the solution of the following system of equations:

$$\begin{cases} a_i''\theta_i' + b_i'' = a_i\theta_i' + b_i, \\ a_i''\theta_i'' + b_i'' = a_i'\theta_i'' + b_i'. \end{cases} \tag{5}$$

The resulted clock model therefore becomes as follows:

$$\hat{\theta}_i^{Ref}(\theta_i) = \begin{cases} a_i\theta_i + b_i, \text{if } \theta_i \leq \theta_i', \\ a_i'\theta_i + b_i', \text{if } \theta_i' < \theta_i \leq \theta_i'', \\ a_i''\theta_i + b_i'', \text{if } \theta_i > \theta_i''. \end{cases} \tag{6}$$

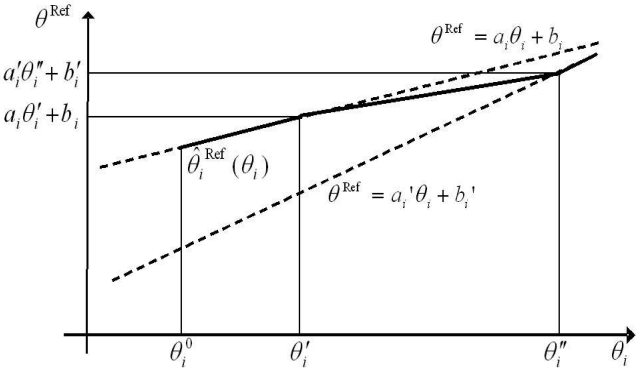


Figure 2: Update rule for clock model parameters that satisfies the monotonicity and smoothness constraints.

3. TIMING RELATIONSHIPS ON GPC PLATFORM

In order to understand parameter estimation technique for the inter and intra platform clock models we briefly describe the operations and timing relationships on a typical GPC platform.

Figure 3 shows a processing diagram of networking and audio I/O. Both I/O operations have a very similar structure that can be described by the following sequence of actions (only input path is described):

[1] In this work we are interested in synchronizing audio I/O, however, our methods are applicable to other I/O devices.

[2] In the rest of the paper we will sometimes omit explicit time dependency to simplify our notations.

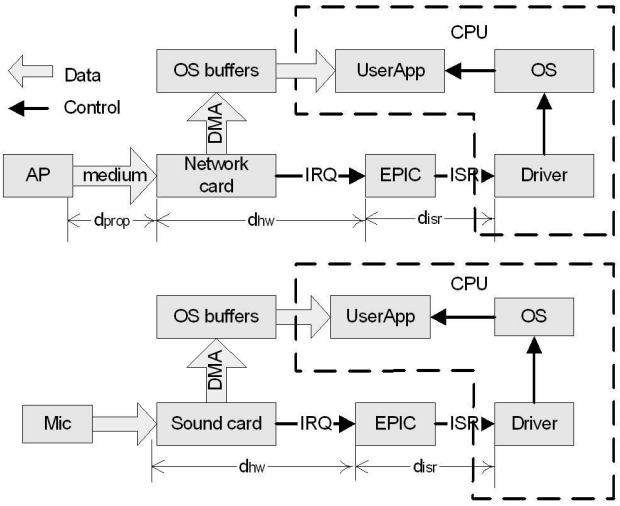


Figure 3: Network (top part) and audio (bottom part) data and control flows on on a typical GPC platform.

1. Incoming data is received and processed by a hardware device, and eventually is put into a Direct Memory Access (DMA) buffer. This is modeled in Figure 3 by the delay d_{hw}, which is approximately constant for similar hardware.
2. The DMA controller transfers the data to a memory block allocated by the system and signals this event to the CPU by an Interrupt ReQuest (IRQ). This stage introduces variable delay due to memory bus arbitration between different agents (i.e., CPU, graphics adapter, other DMA's).
3. The interrupt controller (APIC) queues the interrupt and schedules a time slot for handling. Because APIC is handling requests from multiple I/O devices this stage introduces variable delay with standard deviation of around 6 *ms* and the maximum deviation of 30 *ms*. Both previous stages are modeled by d_{isr} in Figure 3.
4. The Interrupt Service Routine (ISR) of the device driver is called, and the driver sends notification to the Operating System (OS).
5. The OS delivers a notification and data to the user application(s). This stage has to be executed in a multitasking software environment and this leads to significant variable delays that depend on CPU utilization and many other factors.

In summary, data traverses multiple hardware and software stages in order to travel from an I/O device to the CPU and back. The delay introduced by the various stages is highly variable making the problem of providing a global clock to the GPCs and distributing it to I/O devices very challenging. It is advantageous to perform synchronization as close to hardware as possible, therefore our solution is implemented at the driver level (during ISR) thus avoiding additional errors due to OS processing.

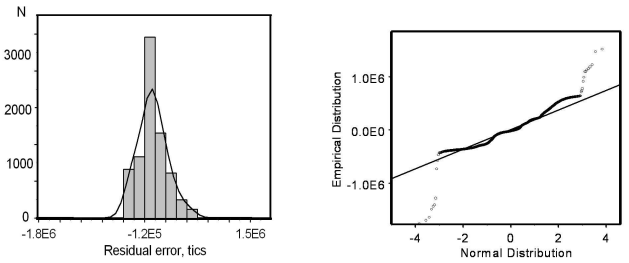
4. TIME SYNCHRONIZATION OF PLATFORM CLOCKS (INTER-PLATFORM SYNCHRONIZATION)

For the synchronization of CPU/platform clocks over a wireless network we propose to use a series of arrival times of multicast packets sent by a wireless access point (AP). In our current approach we implement a pairwise time synchronization with one node chosen as the master (say $\theta^{Ref} = \theta_0$). All other nodes (clients) are required to synchronize their clocks to the master [3]. A similar approach was also suggested in [4, 5]. Our solution, however, extends it by introducing additional constraints on the timing model. In addition we propose to use adaptive statistical processing method to achieve the desired synchronization precision.

The algorithm consists of the following steps:

1. AP sends next beacon packet.
2. Master node records its local time of packet arrival and distributes it to all other nodes.
3. Client nodes record both their local times of arrival of beacon packets from AP, and the corresponding times received from the master.
4. Clients update local timing models based on the set of local timestamps and corresponding master timestamps.

Let us assume that in Figure 3 the packet j arrives to multiple platforms approximately at the same time t^j ($d_{prop} \approx 0$). The set of observations available on the platforms consist of pairs of timestamps $(\tilde{\theta}_0^j, \tilde{\theta}_i^j)$. From Figure 3 we have $\tilde{\theta}^j = \theta(t^j) + d_{hw} + d_{isr}$ (we omitted dependency on i) that we further model as $\tilde{\theta}^j = \theta(t^j) + d + n$. In this approximation d models all constant delay component and n represents the stochastic component. Given the set of observations $(\tilde{\theta}_0^j, \tilde{\theta}_i^j)$ we are required to estimate the timing model parameters a_i and b_i for all slave platforms. In our experiments a window of 3 minutes is used to estimate current values of a_i and b_i using the least trimmed squares (LTS) regression [8]. LTS is equivalent to performing least squares fit, trimming the observations that correspond to the largest residuals (defined as the distance of the observed value to the linear fit), and then computing a least squares regression model for the remaining observations. Figure 4(a) plots the histogram of residuals and Figure 4(b) shows comparison of quantiles of residuals with quantiles of normal distribution. The distribution appears to be close to Gaussian except for the presence of a few outliers (see Figure 4(b)) that do not fit into a normal distribution. The trimming step is specifically targeted to remove those outliers.



(a) (b)

Figure 4: (a) Histogram and empirical density function of residuals. (b) Comparison quantiles of residuals with quantiles of the normal distribution.

5. SYNCHRONIZING I/O TO PLATFORM CLOCK (INTRA-PLATFORM SYNCHRONIZATION)

In order to synchronize the audio clock to the CPU clock we use a similar approach as the one presented in the previous section. The ISR of the audio driver is modified to timestamp the samples in the OS buffer using the CPU clock to form a set of observation pairs $(\tilde{\theta}_i^j, \tau_{i,k}^j)$, where j now represents the index of an audio data packet and k is the index of A/D (D/A) on i-th GPC. Following our model in Figure 3 we have $\tilde{\theta}^j = \theta(t^j) + d_{hw} + d_{isr}$ (we omitted dependency on i and k) that we further represent as $\tilde{\theta}^j = \theta(t^j) + d + n$. Except for the fact that the τ^j are available without any noise (it is simply the number of samples processed!) we are back to the problem of determining the linear fit parameters for pairs of observations that we solved in the previous section using the LTS method. In addition d_{isr} component (that is responsible for most of variability in delay)

[3] A more complex approach of performing joint timing synchronization is potentially more accurate.

is likely to have the same statistics as the similar parameter in the network packet receiving scenario.

In summary, by using LTS procedure twice both local and global synchronization problems are solved and the audio samples can be precisely synchronized on the distributed GPCs. As an example consider audio recording procedure using a network of distributed sensors. All GPCs receive command to start recording at time $\theta^{Ref}(t_0)$. Each GPC starts recording ahead of time to collect statistics for estimating parameters of clock models. When model parameters are estimated it becomes possible to reconstruct samples of "ideal stream" (virtual stream that starts exactly at t_0 and has specified sampling frequency) from the samples of recorded streams. Let τ_i be the offset in real stream corresponding to n_i that is i-th sample of ideal stream. Then the value of n_i is found by performing interpolation of "real samples". In our implementation we use a windowed *sinc* filter to perform interpolation, however, other methods (e.g., splines) performed well in practice.

6. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The distributed test system was implemented with several off-the-shelf laptops implementing Intel®Centrino™Mobile Technology using the following software components (see also Figure 5):

(a) A *modified WLAN card driver* timestamps each interrupt, parses incoming packets in order to find all master beacon frames, and stores their timestamp values in a cyclic shared memory buffer. The timestamp values as well as the corresponding message IDs are further accessible through the standard driver I/O interface.

(b) A *modified AC97 audio driver* timestamps ISRs and calculates the number of samples transmitted since the beginning of the audio capture/playback. The (timestamp,sample number) pair is placed into a cyclic shared memory buffer.

(c) The *synchronization agents* are responsible for synchronizing the distributed system. We have three types of agents: the multicast server (MCS), the master synchronization agent (SAM) and the slave synchronization agent (SAS). The MCS periodically broadcasts beacon packets (short packets with unique ID as the payload). The SAM and SASs use the modified WLAN driver to detect the beacons and their respective IDs. The SAM periodically broadcasts its recorded timestamps of beacon arrivals to the SAS devices. Based on SASs' recorded timestamps and the corresponding SAM timestamps, each SAS calculates the clock parameter to convert between the platform clock and the global clock. The clock parameters are placed in shared memory for use by other applications.

(d) The *Synchronization API* allows user applications to retrieve the local clock value, access the clock parameters, and convert between the platform and global clock.

(e) The *audio API* allows user applications to retrieve pairs of local timestamps and sample numbers, as well as to convert global timestamp values to sample numbers and vice versa. It also provides transparent synchronized capture and playback.

Based on these components a distributed audio rendering system was implemented with three laptops (see Figure 5). The first laptop was used as the MCS. Modified AC97 and WLAN drivers were installed on the other two laptops. SAM was started on the second laptop, while SAS were started on the third laptop. The distributed system was instructed through the audio API to synchronously playback a Maximum Length Sequence (MLS) signal on the two synchronized laptops. The line-out signals of both laptops were recorded by a multichannel sound card. The measured inter-GPC offset was at most 2 samples at 48 kHz ($\leq 42\ \mu$s).

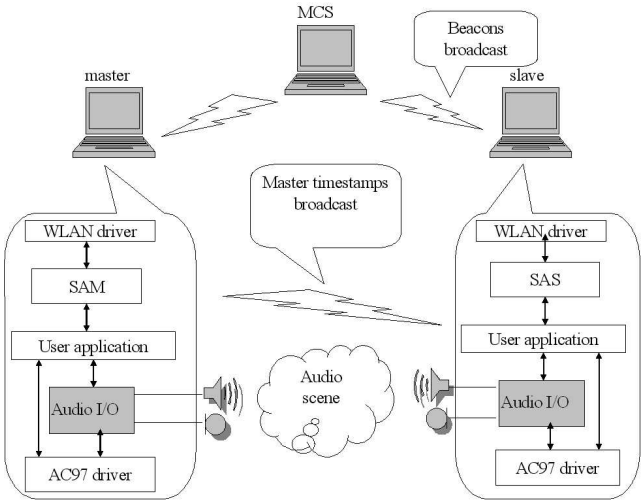


Figure 5: Distributed audio rendering/capturing system setup

7. SUMMARY

We have proposed a general two-step solution to provide a common I/O clock for distributed platforms. In the first step, the CPU clocks of the distributed GPCs are synchronized against each other by using the wireless network. In the second step, each GPC is modeled as a distributed system with multiple I/O devices interconnected to the main memory and CPU via shared buses and a similar synchronization principle is applied. Our results show that at any time the audio I/O offset between different GPCs can be kept below 50 μs that enables usage of *distributed GPCs* for array signal processing of audio and video.

REFERENCES

[1] C. Fancourt and L. Parra, "The coherence function in blind source separation of convolutive mixtures of non-stationary signals," in *Proc IEEE Workshop on Neural Networks for Signal Processing*, 2001, pp. 303–312.

[2] L. Lamport and P.M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *JACM*, vol. 32, no. 1, pp. 52–78, 1985.

[3] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Tran Comm*, vol. 39, no. 10, pp. 1482–1493, 1991.

[4] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Clock synchronization for wireless local area networks," in *IEEE 12th Euromicro Conference on Real-Time Systems (Euromicro RTS 2000)*, 2000, pp. 183–189.

[5] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *5th Symposium on OS Design and Implementation*, Dec 2002.

[6] R. Lienhart, I. Kozintsev, and S. Wehr, "Universal synchronization scheme for distributed audio-video capture on heterogeneous computing platforms," in *Proc 11th ACM Conf on Multimedia*, 2003, pp. 263–266.

[7] D. Budnikov et al., "Providing common i/o clock for wireless distributed platforms," in *Proc IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, 2004, p. (to appear).

[8] P. J. Rousseeuw, "Least median-of-squares regression," *JACM*, vol. 79, pp. 871–880, 1984.