Belloch Rodríguez, JA.; Gonzalez, A.; Vidal Maciá, AM.; Cobos Serrano, M. (2015). On the performance of multi-GPU-based expert systems for acoustic localization involving massive microphone array. Expert Systems with Applications. 42(13):5607-5620. doi:10.1016/j.eswa.2015.02.056.

On the Performance of Multi-GPU-Based Expert Systems for Acoustic Localization Involving Massive Microphone Arrays

Jose A. Belloch[a,*], Alberto Gonzalez[a], Antonio M. Vidal[b], Maximo Cobos[c]

[a] *Institute of Telecommunications and Multimedia Applications, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain.*
[b] *DSIC department, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain.*
[c] *Computer Science Department, Universitat de València,Poligon de la Coma s/n, 46100, Valencia, Spain.*

Abstract

Sound source localization is an important topic in expert systems involving microphone arrays, such as automatic camera steering systems, human-machine interaction, video gaming or audio surveillance. The Steered Response Power with Phase Transform (SRP-PHAT) algorithm is a well-known approach for sound source localization due to its robust performance in noisy and reverberant environments. This algorithm analyzes the sound power captured by an acoustic beamformer on a defined spatial grid, estimating the source location as the point that maximizes the output power. Since localization accuracy can be improved by using high-resolution spatial grids and a high number of microphones, accurate acoustic localization systems require high computational power. Graphics Processing Units (GPUs) are highly parallel programmable co-processors that provide massive computation when the needed operations are properly parallelized. Emerging GPUs offer multiple parallelism levels; however, properly managing their computational resources becomes a very challenging task. In fact, management issues become even more difficult when multiple GPUs are

*Corresponding author: Phone Number +34-655436190
Email addresses: jobelrod@iteam.upv.es (Jose A. Belloch), agonzal@dcom.upv.es (Alberto Gonzalez), avidal@dsic.upv.es (Antonio M. Vidal), maximo.cobos@uv.es (Maximo Cobos)

involved, adding one more level of parallelism. In this paper, the performance of an acoustic source localization system using distributed microphones is analyzed over a massive multichannel processing framework in a multi-GPU system. The paper evaluates and points out the influence that the number of microphones and the available computational resources have in the overall system performance. Several acoustic environments are considered to show the impact that noise and reverberation have in the localization accuracy and how the use of massive microphone systems combined with parallelized GPU algorithms can help to mitigate substantially adverse acoustic effects. In this context, the proposed implementation is able to work in real time with high-resolution spatial grids and using up to 48 microphones. These results confirm the advantages of suitable GPU architectures in the development of real-time massive acoustic signal processing systems.

Keywords: Sound Source Localization; Steered Response Power; Microphone Arrays; Graphics Processing Units

1. Introduction

Microphone arrays are commonly employed in many signal processing tasks, such as speech enhancement, acoustic echo cancellation or sound source separation (Brandstein & Ward, 2001). The localization of broadband sound sources under high noise and reverberation is another challenging task in multichannel signal processing, being an active research topic with applications in human-computer interfaces (Kodagoda & Sehestedt, 2014), teleconferencing (Wang et al., 2011) or emergency units (Calderoni et al., 2015). Microphone arrays may follow a given geometry, such as spherical arrays (Huang & Wang, 2014), or may be distributed. Algorithms for sound source localization can be broadly divided into indirect and direct approaches (Madhu & Martin, 2008). Indirect approaches usually follow a two-step procedure: they first estimate the *Time Difference Of Arrival* (TDOA) (Chen et al., 2006) between microphone pairs, and, afterwards, they estimate the source position based on the geometry of

the array and the estimated delays. On the other hand, direct approaches perform TDOA estimation and source localization in one single step by scanning a set of candidate source locations and selecting the most likely position as an estimate of the real source location. Although the computation of TDOAs usually requires time synchronization, new approaches are being developed to avoid this limitation (Xu et al., 2013). Most localization algorithms are based on the *Generalized Cross-Correlation* (GCC) (Knapp & Carter, 1976), which is calculated by using the inverse Fourier transform of the weighted cross-power spectral density of the signals. The *Steered Response Power - Phase Transform* (SRP-PHAT) algorithm is a direct approach that has been shown to be very robust in adverse acoustic environments (DiBiase et al., 2001). The algorithm is usually interpreted as a beamforming-based approach that searches for the candidate position that maximizes the output of a steered delay-and-sum beamformer.

The CUDA platform (CUDA, 2015) provides a computing framework that enables the use of Graphics Processing Units (GPUs) in applications beyond image processing (Liu et al., 2007; Zhao & Lau, 2013). GPUs are high parallel programmable co-processors that provide efficient computation when the needed operations are properly parallelized. Programming a GPU efficiently requires having good knowledge of both the underlying architecture and the mechanisms used by GPUs to distribute their tasks among their processing units. Since the appearance of CUDA programming, many researchers in different areas have made use of it to achieve better performances in their respective fields. For example, well-known computational cores have also been adapted to a GPU computing framework, such as LU factorization (Dazevedo & Hill, 2012), matrix multiplication (Matsumoto et al., 2011) or the Boltzmann equation (Kloss et al., 2010). In audio and acoustics, several works demonstrate the potential of GPUs for carrying out audio processing tasks. For example, the implementation of a multichannel room impulse response reshaping algorithm was carried out in (Mazur et al., 2011), and implementations of adaptive filtering algorithms were presented in (Schneider et al., 2012; Lorente et al., 2012, 2013, 2014). GPU-

based room acoustics simulation was carried out in (Savioja, 2010; Southern et al., 2010; Webb & Bilbao, 2011; Hamilton & Webb, 2013). One of the main contributions within this field was carried out in (Savioja et al., 2011), where improved performances in additive synthesis, Fourier transform and convolution in the frequency domain were presented. A comparison between CPU and GPU performance for a simple crosstalk canceller is presented in (Belloch et al., 2011). Similarly, a binaural audio application with massive audio processing that was fully implemented on a GPU is presented in (Belloch et al., 2013a). GPUs are also used in (Vanek et al., 2012) and in (Bradford et al., 2011) for evaluating the likelihood function in automatic speech recognizers and for sliding phase vocoder, respectively.

The use of GPUs for implementing sound source localization algorithms has also recently been tackled in the literature. The time performances of different localization algorithms implemented on GPU were reported in (Peruffo Minotto et al., 2012) and (Liang et al., 2012). In fact, although different implementations of the SRP-PHAT in the time-domain and frequency-domain are analyzed in (Peruffo Minotto et al., 2012), their results mainly focus on pure computational issues and do not discuss how localization performance is affected by using different numbers of microphones or a finer spatial grid. In (Seewald et al., 2014), the SRP-PHAT algorithm is implemented over two Kinects for performing sound source localization. In the same work, the algorithm only estimates the relative source direction instead of providing the absolute source position and the implementation is evaluated on different GPUs that belong to the old-fashioned Fermi(CUDA, 2015).

One of our previous works (Belloch et al., 2013b) analyzed the performance of a 2-D SRP-PHAT implementation with different Nvidia GPU architectures. The present paper extends that work in various aspects. First, 3-D source localization is considered, leading to a significant increase in the required computational cost. Second, the system considered in this work makes use of multiple GPUs, facing new challenges in parallelization and resource management. Finally, this paper provides a deeper analysis of the influence of the acoustic environment

and the number of microphones in the final performance. As a result, this paper is aimed at demonstrating how localization systems using a high number of microphones distributed within a room can perform sound source localization in real time under adverse acoustic environments by using GPU massive computation resources. Specifically, the well-known SRP-PHAT algorithm is considered here. Note that coarse-to-fine search strategies have been proposed to overcome many of the processing limitations of SRP-PHAT (Do & Silverman, 2007; Said et al., 2013; Marti et al., 2013). However, while these strategies provide more efficient ways to explore the localization search volume, they only provide better performance than the conventional SRP-PHAT when the number of operations is restricted. Thus, the performance of the conventional SRP-PHAT with fine spatial grids is usually considered as an upper bound in these cases.

Relevant parameters that affect the computational cost of the algorithm (number of microphones and spatial resolution) are analyzed, showing their influence on the localization accuracy in different situations. We also discuss the scalability of the algorithm when multi-GPU parallelization issues are considered. This paper highlights the need for massive computation in order to achieve high-accuracy localization in adverse acoustic environments, taking advantage of GPUs to fulfill the computational demand of the system.

In comparison with the implementation presented in (Seewald et al., 2014), we design our application to achieve maximum performance on GPUs making use of the Kepler architecture GK110 (K20, 2014) (See Appendix A for details). This architecture can be found on the Tegra K1 (TK1) systems-on-chip (SoC), embedded in the Jetson development kit (DevKit) (Jetson, 2015), and it is becoming widespread in current mobile devices such as Google's Nexus 9 tablet (Nexus, 2015). Thus, the proposed implementation can be successfully adapted to work properly on GPUs that are currently embedded in mobile devices.

The paper is structured as follows. Section 2 briefly describes the basic SRP-PHAT localization algorithm that will be used throughout this paper. Section 3 presents the implementation of the algorithm on multi-GPU systems. The

proposed acoustic environments for real-time sound source localization are presented in Section 4, describing the experiments conducted for studying the performance of the method in a real application context. The computational performance of the different multi-GPU implementations are also analyzed. Finally, Section 5 provides some concluding remarks. Two Appendixes are provided in order to facilitate the understanding of the parallelization techniques that are used throughout this article.

2. Sound Source Localization: SRP-PHAT Algorithm

Consider the output from microphone l, $m_l(t)$, in an M microphone system. The Steered Response Power (SRP) at the spatial point $\mathbf{x} = [x, y, z]^T$ for a time frame n of length T_L can then be defined as

$$P_n(\mathbf{x}) \equiv \int_{nT_L}^{(n+1)T_L} \left| \sum_{l=1}^{M} w_l m_l \left(t - \tau(\mathbf{x}, l)\right) \right|^2 dt, \tag{1}$$

where w_l is a weight and $\tau(\mathbf{x}, l)$ is the direct time of travel from location $\mathbf{x}$ to microphone l. DiBiase (DiBiase, 2000) showed that the SRP can be computed by summing up the Generalized Cross-Correlations (GCCs) for all possible pairs of the set of microphones. The GCC for a microphone pair (k, l) is defined as

$$R_{m_k m_l}(\tau) = \int_{-\infty}^{\infty} \Phi_{kl}(\omega) M_k(\omega) M_l^*(\omega) e^{j\omega\tau} d\omega, \tag{2}$$

where τ is the time lag, * denotes complex conjugation, $M_l(\omega)$ is the Fourier transform of the microphone signal $m_l(t)$, and $\Phi_{kl}(\omega)$ is a combined weighting function in the frequency domain. The phase transform (PHAT) (Knapp & Carter, 1976) has been shown to be a suitable GCC weighting for time delay estimation in reverberant environments. The PHAT weighting is expressed as:

$$\Phi_{kl}(\omega) \equiv \frac{1}{|M_k(\omega) M_l^*(\omega)|}. \tag{3}$$

Taking into account the symmetries involved in the computation of Eq.(1) and removing some fixed energy terms (DiBiase, 2000), the part of $P_n(\mathbf{x})$ that

changes with $\mathbf{x}$ can be isolated as

$$P'_n(\mathbf{x}) = \sum_{k=1}^{M} \sum_{l=k+1}^{M} R_{m_k m_l} \left(\tau_{kl}(\mathbf{x})\right), \tag{4}$$

where $\tau_{kl}(\mathbf{x})$ is the Inter-Microphone Time-Delay Function (IMTDF). This function is very important since it represents the theoretical direct path delay for the microphone pair (k, l) resulting from a point source located at $\mathbf{x}$. The IMTDF is mathematically expressed as (Cobos et al., 2011)

$$\tau_{kl}(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{x}_k\| - \|\mathbf{x} - \mathbf{x}_l\|}{c}, \tag{5}$$

where c is the speed of sound ($\approx$ 343 m/s), and $\mathbf{x}_k$ and $\mathbf{x}_l$ are the locations of the microphone pair (k, l).

The SRP-PHAT algorithm consists in evaluating the functional $P'_n(\mathbf{x})$ on a fine grid $\mathcal{G}$ with the aim of finding the point-source location $\mathbf{x}_s$ that provides the maximum value:

$$\mathbf{x}_s = \arg\max_{\mathbf{x} \in \mathcal{G}} P'_n(\mathbf{x}). \tag{6}$$

Figure 1 shows schematically the intuition behind SRP-PHAT localization. In this figure, an anechoic environment is assumed so that the GCC for each microphone pair is a delta function located at the real TDOA. Each TDOA defines a half-hyperboloid of potential source locations. The intersection resulting from all the half-hyperboloids matches the point of the grid having the greatest accumulated value.

2.1. SRP-PHAT Implementation

The SRP-PHAT algorithm is usually implemented on a grid by carrying out the following steps:

1. A spatial grid $\mathcal{G}$ is defined with a given spatial resolution r. The theoretical delays from each point of the grid to each microphone pair are pre-computed using Eq.(5).
2. For each analysis frame, the GCC of each microphone pair is computed as expressed in Eq.(2).

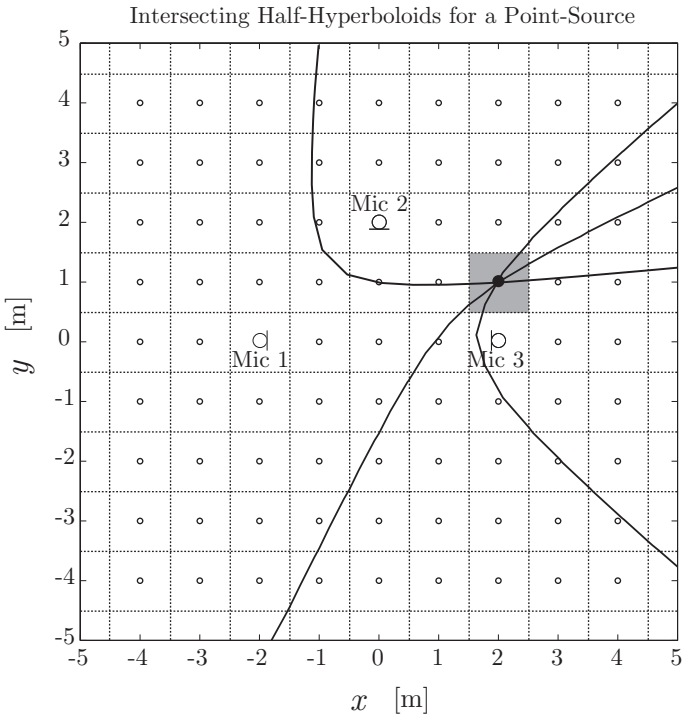


Figure 1: Intersecting half-hyperboloids for $M = 3$ microphones. Each half-hyperboloid corresponds to a TDOA peak in the GCC.

3. For each position of the grid $\mathbf{x} \in \mathcal{G}$, the contribution of the different cross-correlations are accumulated (using delays pre-computed in 1), as in Eq.(4).

4. Finally, the position with the maximum score is selected as in Eq.(6).

The SRP-PHAT localization performance depends on the selected spatial resolution r. Figure 2 illustrates the algorithm performance when considering different spatial grid resolutions. The accumulated SRP-PHAT values for each spatial grid location are shown for a 2-D plane in a 4×6 m room with $N = 6$ microphones. Note how the location of the source is more easily detected when finer spatial resolutions are used, as in the case of $r = 0.01$ m.

2.2. Computational Cost

The SRP-PHAT algorithm is usually implemented by performing a frequency-domain processing of the input microphone signals. Given M microphones, the number of microphone pairs to process is $Q = M(M-1)/2$. For a DFT size of L (equal to the time-window size), an FFT takes $5L\log_2 L$ arithmetic operations that result from $\frac{L}{2}\log_2 L$ complex multiplications and $L\log_2 L$ complex additions. Note that one complex multiplication is equivalent to four real multiplications and one real addition, while a complex addition is equivalent to two real additions. As a result, the signal processing cost for computing the GCC is given by:

- **DFT**: Compute M FFTs, then, $M \times 5L\log_2 L$.
- **Cross-Power Spectrum**: A complex multiplication for L points, resulting in $6L$ operations (4 real multiplications and 2 real additions). This is done for Q microphone pairs, resulting in a cost of $6QL$.
- **Phase Transform**: Magnitude of the L points of the GCC, which costs L operations. This is also done for Q pairs, resulting in QL operations.
- **IDFT**: The IDFT for Q pairs must be performed, which requires $Q5L\log_2 L$ operations.

Moreover, for each functional evaluation, the following parameters must be calculated:

- M Euclidean distances, $\|\mathbf{x}_m\|$, requiring 3 multiplications, 5 additions and 1 square root (≈ 12 operations): $20M$ operations
- Q TDOAs, requiring 2 operations (1 subtraction and 1 division by c) per microphone pair: $2Q$ operations.
- The SRP requires truncating the TDOA values to the closest sample according to the system sampling frequency, multiplying the cross-power spectrum to obtain the phase transform for each microphone pair and adding up all the GCC values: $5Q$ operations.

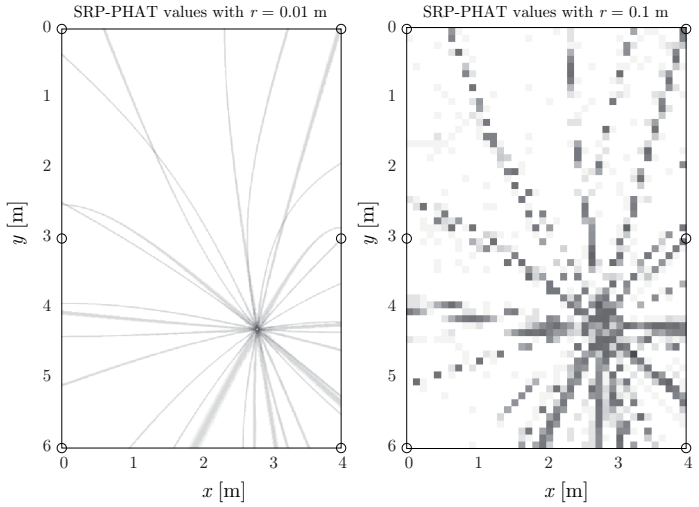


Figure 2: Accumulated SRP-PHAT values for a 2-D spatial grid (4×6 m and $M = 6$ microphones) with different spatial resolutions. (a) $r = 0.01$ m. (b) $r = 0.1$ m.

As a result, the cost of the SRP-PHAT is given by:

$$\begin{aligned} Cost &= \left(\frac{M + M^2}{2}\right) 5L \log_2 L + \\ \frac{7M(M-1)}{2} L &+ \nu \left(20M + \frac{7M(M-1)}{2}\right), \end{aligned} \tag{7}$$

where ν is the total number of functional evaluations. In the conventional full grid-search procedure, ν equals the total number of points of the grid $\mathcal{G}$. Figure 3 shows the computational cost of the algorithm for different spatial resolutions and number of microphones, considering a 3D grid search space with a uniform spatial resolution of r meters.

3. Algorithm Parallelization for real-time GPU implementation

The GPU-based implementation of the SRP-PHAT algorithm is applied to Nvidia hardware devices with Kepler architecture GK110 (K20, 2014). Appendix A provides a detailed description of the GPU-parallelization techniques used throughout this section.

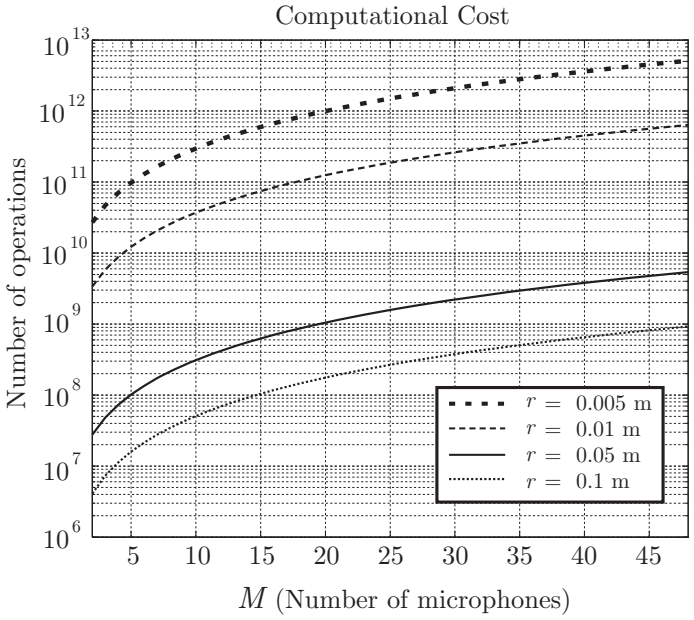


Figure 3: Computational cost when for different number of microphones M and spatial resolutions r.

Since the localization is carried out in three dimensions, three different resolutions r_x, r_y, and r_z define the spatial grid $\mathcal{G}$. Taking a shoe-box-shaped room as a model room with dimensions $l_x \times l_y \times l_z$, the size of the grid is $\nu = P_x \times P_y \times P_z$, where $P_x = \frac{l_x}{r_x}$, $P_y = \frac{l_y}{r_y}$ and $P_z = \frac{l_z}{r_z}$.

The real-time implementation of the SRP-PHAT algorithm uses 50% time-window overlap, with audio sample buffers of size L. These $L \times M$ samples are transferred to the GPU first. A GPU buffer (denoted here as $\mathbf{T}_{GPU}$) stores the audio samples in consecutive memory positions as they arrive to the GPU. One aspect that affects the performance for all audio signal processing applications on GPU is the transfer of audio samples from CPU to GPU. As mentioned in Appendix A.1, streams can be used to parallelize these transfers and overlap them with the computation. Since we use 50% overlap, the processing is carried out in blocks of size $2L$, which are composed of the current audio-sample buffer and the previous one. Thus, a size of $2LM$ is used for $\mathbf{T}_{GPU}$. The SRP-PHAT

GPU implementation carries out the following steps:

1. M streams are created (one stream for each microphone in the system). The streams are launched consecutively in an asynchronous way. Stream l transfers L samples captured by microphone l to the GPU and stores them in $\mathbf{T}_{GPU}$, with $l = 0, \ldots, M-1$. Then, stream l launches Kernel A, which is responsible for grouping $2L$ elements of microphone l (L samples from previous buffers and L samples from current buffers). These $2L$ elements are also weighted using a Hamming window vector. For this purpose, the stream launches a kernel that is composed of 128-size thread blocks in a CUDA grid of dimensions ($\frac{2L}{128} \times 1$) (i.e., it is composed of $2L$ CUDA threads). Each thread computes one element of the $2L$ elements.

   The tasks carried out by Kernel A are simple. Each thread reads one value from *global-memory*, multiplies it by a `float` number (a value of Hamming window vector) and stores it in a different position of *global-memory*. The accesses to *global-memory* are totally coalesced, since audio samples are stored in consecutive memory positions both when reading and when writing (see Fig. 4). Also, L is power of 2 and is always larger than 1024. Thus, each thread block reads and writes in 128 consecutive memory positions. The selection of 128 for the block size was done experimentally among 64, 128, 256 and 512, with 128 being the one that requires less time.

2. Once Kernel A has finished, stream l uses the CUFFT library to perform a $2L$-FFT using these $2L$ elements. As a result of the computation performed by all the streams, M vectors that are composed of $2L$ frequency bins (denoted as $\mathbf{f}_l$, $l = 0, \ldots, M-1$) are obtained. The use of streams allows us to overlap data transfers with computation. For example, while stream 1 is transferring samples from microphone 1, stream 0 can be executing Kernel A. However, the next steps involve operations among different channels. Thus, all the previous operations must finalize before continuing. This implies synchronization among all the streams.

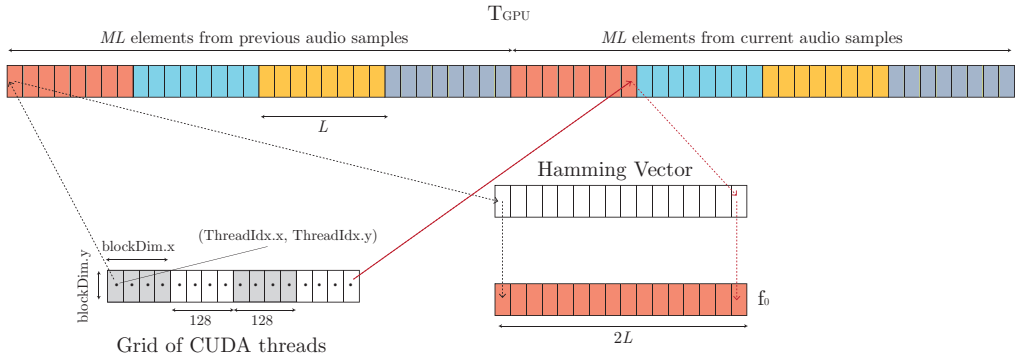


Figure 4: Operations that are carried out by CUDA kernel 21 in case M=4.

The following steps are computed by only one stream.

3. The **GCC** matrix is computed by means of another kernel (Kernel B). In this kernel, a GPU thread takes one value from each of two different $\mathbf{f}_l$ buffers that are at the same vector position. It conjugates one of the values and multiplies it by the corresponding value of the other $\mathbf{f}_l$ buffer. The phase of the complex number obtained by the multiplication is stored in the corresponding position in the **GCC** matrix.

   The accesses to the two $\mathbf{f}_l$ buffers by GPU threads are totally coalesced since consecutive threads access consecutive memory positions (see Fig. 5). Kernel B is limited by the instruction bandwidth since GPU-native functions `cosf`, `sinf`, and `atan2f` are used and all of them require various clock cycles. Kernel B computes $2LQ$ values of the **GCC** matrix, where Q represents the number of microphone pairs. In order to define the size and the number of blocks to launch in Kernel B, different tests were executed. The best performance was achieved by using 128-size thread blocks in a CUDA grid with size 32×16. This implies launching 65536 threads, where each thread is responsible for computing $\frac{2LQ}{65536}$ values of the **GCC** matrix. In this case, increasing the amount of work per thread block in Kernel B is more beneficial than launching more blocks with fewer operations per GPU thread. Thus, the grid configuration applied to Kernel B

achieves maximum occupancy when 512 blocks are launched. This kernel does not require using *shared-memory* but preferably a large number of registers. Thus, we set L1 cache to 48 KB. As described in (K20, 2014), cache L1 is used for register spills, local memory, and stack, which are all private per-thread variables.

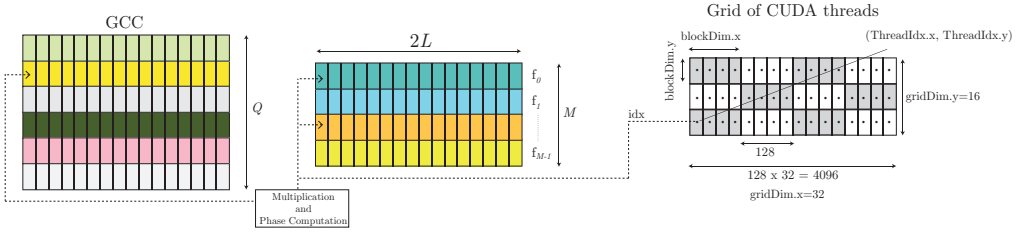


Figure 5: Operations that are carried out by Kernel B.

4. Q inverse FFTs of size $2L$ are then carried out by using the CUFFT library. The **GCC** matrix is now composed of temporal (time delay) values (i.e., $2LQ$ real values).

5. The computation of a tridimensional matrix **SRP** storing the accumulated SRP values is carried out by Kernel C. This kernel also launches thread blocks of size 128 in a tridimensional CUDA grid whose dimension depends on the number of points of the grid $\mathcal{G}$ (ν). In total, ν threads are launched. In this kernel, each GPU thread is devoted to the computation of the total value of the SRP at each point of the grid. To this end, each thread computes and accumulates Q GCC values (it takes a value from each row of the **GCC** matrix and accumulates it). The computation of the SRP requires Q calculations of the IMTDF (see Eq. 5) at each point of the grid. The IMTDF of a pair of microphones specifies the column of the **GCC** matrix that should be selected and then accumulated in the SRP. Figure 6 illustrates these operations.

   Since the value of the IMTDF can indicate any position of the column of the **GCC** matrix, coalesced access to the *global-memory* is not guaranteed. In fact, the most probable situation is that the accesses will be

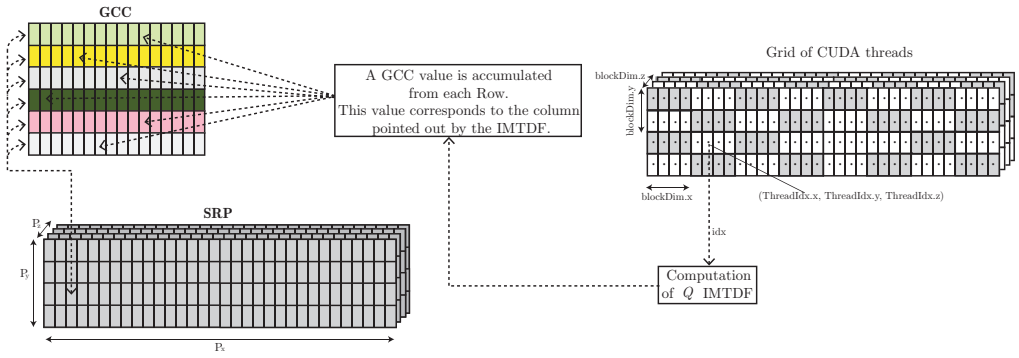


Figure 6: Operations that are carried out by Kernel C.

quite disordered, so that the kernel employs most of its time in memory accessing. However, this limitation can be reduced if we force the compiler to use the Kepler read-only data cache with the **GCC** matrix, since this cache does not require aligned accesses. This read-only cache memory has also been used in recent GPU-based audio research such as (Hamilton & Webb, 2013) and (Bilbao & Webb, 2013). Furthermore, as in Kernel B, we set L1 cache to 48 KB to favor possible register spills. In the accumulation loop of the SRP values, we have set a `#pragma unroll` to accelerate the computation.

6. The grid position corresponding to the maximum SRP value has to be searched. To this end, we launch Kernel D. This kernel exactly follows the reduction example in Harris' implementation (Harris, 2014) that comes with the Nvidia GPU Computing SDK (Software development kit), but it changes the sum operation for a maximum operation. However, even though this code is optimized for finding the maximum value, it does not indicate its position. Thus, after obtaining the maximum, we launch another kernel (Kernel E). This kernel launches as many threads as elements of the **SRP** matrix and only performs a comparison operation with the maximum. If the comparison matches, the thread writes the value of its index in a variable.

3.1. Memory considerations

The computation of the IMTDF could be carried out off-line since the grid resolutions and the microphone locations are static. However, this would imply storing a 4-dimensional data structure composed of $\nu \cdot Q$ elements. If we use a standard room size (such as 6.0 $\times$ 4.0 $\times$ 3.0 m), a resolution of $r = r_x = r_y = r_z = 0.01$ m, and $M = 48$ microphones, this data structure would require using more than eight gigabytes of *global-memory*. This exceeds the *global-memory* size of most available GPU devices. Thus, every IMTDF value is computed for each group of processed buffers.

There are also other variables that are used to compute the values of the **GCC** and **SRP** matrices, such as the room dimensions, the number of microphones and their position. Since all of these read-only variables must be available for all of the threads, they are stored in the *constant memory* (with size 64 KB).

3.2. Multi-GPU Parallelization

Distributing the above processing tasks among different GPUs is not straightforward. The greatest computational load relies on Step 6, which consists in computing the maximum value of the **SRP** matrix. Table 1 shows the elapsed time corresponding to each step for M =48 microphones and a spatial grid resolution of r =0.01 m.

The tasks from Kernels C, D and E can be easily distributed among N_{GPU} GPUs (the number of GPUs present in the system): each GPU computes $\frac{\nu}{N_{GPU}}$ elements of the **SRP** matrix and locates the maximum among its computed elements. To this end, N_{GPU} CPU threads are created at the beginning of a parallel region by means of *openMP* (see Appendix A.2 to know how *openMP* can deal with multiple GPUs). This strategy is only focused on multi-GPU parallelization of the **SRP** matrix.

In Appendix B, there is a description of an alternative strategy that aims at parallelizing both the computation of the **SRP** matrix and the **GCC** matrix.

Table 1: Elapsed Time in each kernel with M=48 and spatial grid resolution r=0.01.

Steps of the algorithm	**Time** [ms]
Transfer + Kernel A + FFT (steps 1 and 2 in Section III)	1.416
Kernel B (step 3: Computation of GCC)	0.015
IFFT of GCC (step 4)	0.006
Kernel C (Computation of SRP matrix)	0.007
Kernel D (Reduction: Computation of Maximum SRP value)	121.267
Kernel E (Localization of the Maximum)	0.009
Total elapsed time	122.720

This strategy uses also the UVA (Unified Virtual Addressing) feature for inter-GPU communication. This strategy requires different synchronization points that significantly penalize their performances, especially when compared to the parallelization presented in this article.

3.3. Basic Implementation using two GPUs

As shown in Section 4, the performance of the SRP-PHAT algorithm is assessed in a system that is composed of two GPUs. Using all the parallelization techniques previously presented, the SRP-PHAT algorithm is implemented on two GPUs as follows:

1. A parallel region is created with two CPU threads. Each CPU thread is bound with a GPU.
2. Since different audio buffers are received in the system, each CPU thread independently and asynchronously sends all audio buffers to its GPU by using stream parallelization. The Kernels A and the FFTs are computed for each channel inside the streams.
3. As in step 2 of Section 3, stream synchronization is addressed. Only one stream is used to compute the rows of the **GCC** matrix.

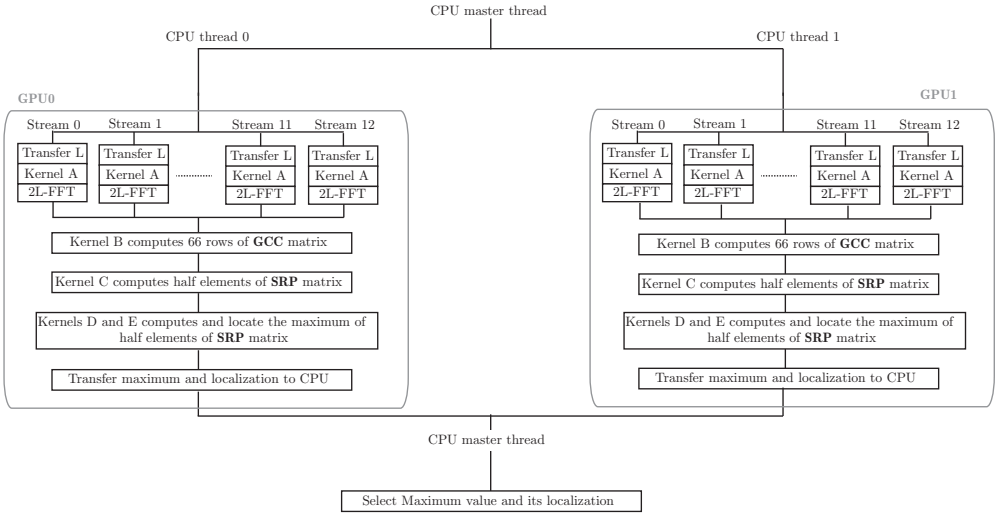


Figure 7: Steps of the GPU-based SRP-PHAT implementation using two GPUs and *openMP*.

4. Since both of them have computed the **GCC** matrix, each GPU computes $\nu/2$ elements of the **SRP** matrix and locates a maximum value among the computed elements.
5. Each GPU transfers back to the CPU its maximum value and its location inside the **SRP** matrix. Then, a synchronization barrier for both CPU threads is set followed by an *openMP* section that is only executed by the master thread. This thread compares the two maximum values and chooses the greatest one, getting its location. This location indicates the sound source position. Figure 7 illustrates the computation of the SRP-PHAT when $M = 12$.

4. Experiments and Performance

To analyze both the computing and localization performance of the above GPU implementations, a set of acoustic simulations using the image-source method (Allen & Berkley, 1979) have been considered. A shoe-box-shaped room with dimensions 4×6×3 m and wall reflection factor ρ (Kuttruff, 2000) was simulated using different numbers of microphones ($M \in \{6, 12, 24, 48\}$).

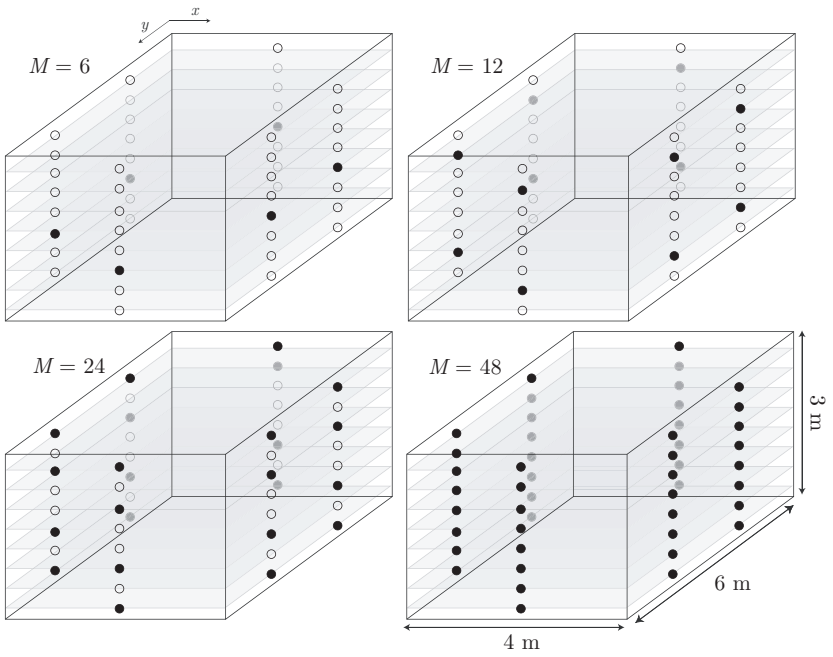


Figure 8: Microphone set-ups for $M = 6$, $M = 12$, $M = 24$ and $M = 48$. The black dots denote the actual active microphones in each configuration.

The microphone set-up for the considered systems are shown in Figure 8. Note that the microphones are located on the walls of the room and are placed on eight different planes ($z = \{0.33, 0.66, 1.00, 1.33, 1.66, 2.00, 2.33, 2.66\}$) following hexagon-like shapes. Moreover, different reflection factors ($\rho \in \{0, 0.5, 0.9\}$) were used to take into account different reverberation degrees. In all cases, independent white Gaussian noise was added to each microphone signal in order to simulate different *Signal to Noise Ratios* (SNR $\in \{0, 5, 10, 20\}$) (in dB).

The audio card used in the real-time prototype uses an ASIO (Audio Stream Input/Output) driver to communicate with the CPU and provides 2048 samples per microphone (L=2048) every 46.43 ms at a sample frequency of 44.1 kHz. This time is denoted by t_{buff}. The time employed for the computation is denoted by t_{proc}. This time takes into account all transfer times and measures the time from the first audio sample transferred to the GPU until the final source location

Table 2: Characteristics of the GPU K20c.

Cuda Device	Tesla K20c
Architecture	Kepler
Capability	3.5
Number of SM	13
Total number of cores	2496
Max. dimension of a block	1024 x 1024 x 64
Max. dimension of a grid	2^{31}-1 x 65535 x 65535
Total amount of global memory	4 GB

is estimated (at each time frame). The localization system works in real time as long as $t_{\text{proc}} < t_{\text{buff}}$. Otherwise, microphone samples would be lost and the localization would not be correctly performed. The simulations were carried out in the Nvidia GPU K20c (K20, 2014), which has the characteristics shown in Table 2. Both computational and localization performances have been assessed taking into account three spatial grid resolutions ($r \in \{0.1, 0.05, 0.01\}$) in the XY plane (resolutions r_x and r_y are equal). The resolution r_z is 0.33 m (resulting from dividing the height of the room into eight slots).

4.1. Localization Performance

The source signal used in this study was a 5-second male speech signal with no speech pauses. Pauses were manually suppressed to evaluate localization performance only over frames where speech was present. The processing was carried out by using 50% overlap in time windows of length 4096 samples (size $2L$), with sampling frequency $f_s = 44.1$ kHz. For each frame, a source location $\hat{\mathbf{x}} = [\hat{x}, \hat{y}, \hat{z}]^T$ was estimated. A total number of 107 frames (N_f=107) per 40 different positions (N_p=40) that were uniformly distributed over the room space were performed. Localization accuracy was computed by means of the Mean

Absolute Error, which is given by:

$$\mathrm{MAE} = \frac{1}{N_f}\frac{1}{N_p}\sum_{i=1}^{N_f}\sum_{j=1}^{N_p}|\mathbf{e}_{ij}|_2, \tag{8}$$

where $\mathbf{e}_{ij}$=$\mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}$, with $\mathbf{x}_{ij}$ and $\hat{\mathbf{x}}_{ij}$ being the true and estimated source locations at a given time frame i and source position j. Note that the above MAE was computed for each environmental condition (reflection factor and signal to noise ratio), microphone setup and spatial grid resolution. Figure 9 shows the results for different values of wall reflection factor ρ taking into account different spatial resolutions and number of microphones.

It is important to point out that using a high number of microphones helps to substantially improve localization accuracy under high noise and reverberation. The error decreases as the SNR increases and/or reverberation decreases (lower ρ). It is important to see how the spatial resolution has an impact when there are few microphones. In this case, a coarse spatial grid is not sufficient to correctly find the minimum of the SRP search space, which is more easily detected when the SRP is enhanced by the contributions of additional microphone pairs. In fact, when the number of microphones is 12 or higher, the performance difference between $r = 0.01$ and $r = 0.1$ is almost negligible. Accuracy differences among different values of ρ are noticiable. It should be emphasized that, under favorable acoustic conditions (high SNR and low ρ), the experimental error is always below the maximum expectable error independently of the number of microphones. Note that the maximum error in anechoic conditions is given by the largest diagonal of the cuboids forming the 3D grid (≈ 0.179 m for $r = 0.1$ and ≈ 0.165 m for $r = 0.01$). In all cases, the use of a higher number of microphones significantly helps in reducing this error.

4.2. Computational Performance

The spatial resolutions considered in this paper result in large-scale SRP matrices. Table 3 shows the processing times t_{proc} for different combinations of r and M when using two GPUs. It can be observed that the only that does not obtain a t_{proc} lower than 46.43 ms (t_{buff}) is the configuration composed of

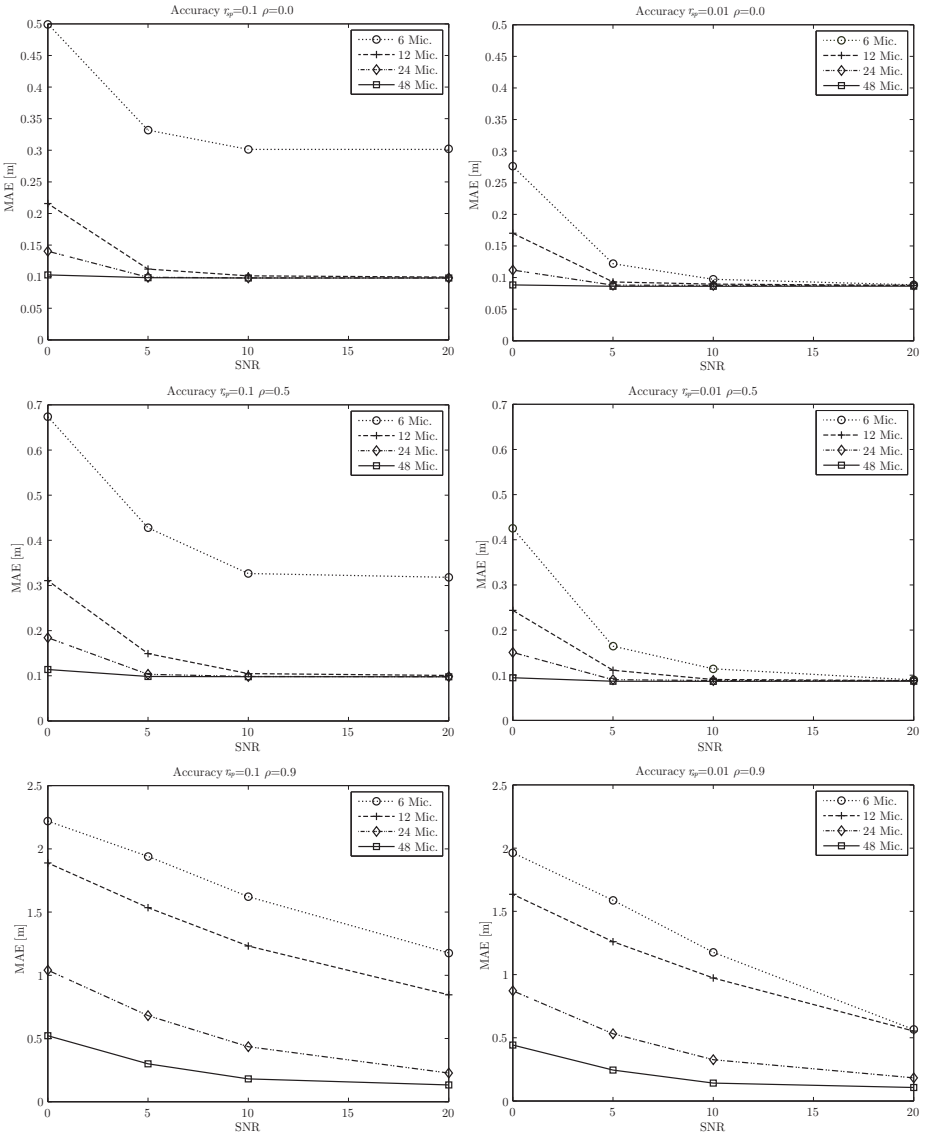


Figure 9: Localization accuracy for different wall reflection factors ($\rho \in \{0, 0.5, 0.9\}$) as a function of the SNR and the number of microphones M. Each column presents results for different spatial resolutions ($r = 0.1$ and $r = 0.01$ m).

$M = 48$ and $r = 0.01$. Thus, real-time processing is not possible in this case. However, by looking at the results shown in Table 4, it is possible to observe

Table 3: Processing time t_{proc} using two GPUs.

r	$M = 6$	$M = 12$	$M = 24$	$M = 48$
0.01	1.031 ms	3.578 ms	15,564 ms	60.108 ms
0.05	0.381 ms	0.758 ms	2.238 ms	6.433 ms
0.1	0.371 ms	0.650 ms	1.588 ms	4.588 ms

Table 4: Processing time t_{proc} using one GPU.

r	M=6	M=12	M=24	M=48
0.01	1.894 ms	6.731 ms	30.145 ms	122.720 ms
0.05	0.564 ms	1.132 ms	3.484 ms	11.203 ms
0.1	0.546 ms	0.926 ms	2.336 ms	7.493 ms

that the influence of the second GPU becomes relevant. In the case of $M = 48$, the processing time is halved for any resolution. Real-time processing would be easily achieved for $M = 48$ and $r = 0.01$ by adding an additional GPU. Figure 10 shows more clearly the time differences among all the configurations by varying the number of microphones and the grid resolutions $r \in \{0.1, 0.05, 0.01\}$. Note that the time t_{buff} is marked by a solid black line.

5. Conclusion

New emerging GPU architectures help to overcome different computational problems in acoustic signal processing algorithms involving many microphone channels. This paper has analyzed the specific case of sound source localization, where very fine spatial resolutions or having a high number of microphones have a deep impact in the performance of real-time applications. In this context, the following contributions have been presented in this paper.

Firstly, we have proposed a scalable multi-GPU implementation of the well-known SRP-PHAT algorithm for source localization in three dimensions. To this end, two parallelization levels have been considered. On the one hand,

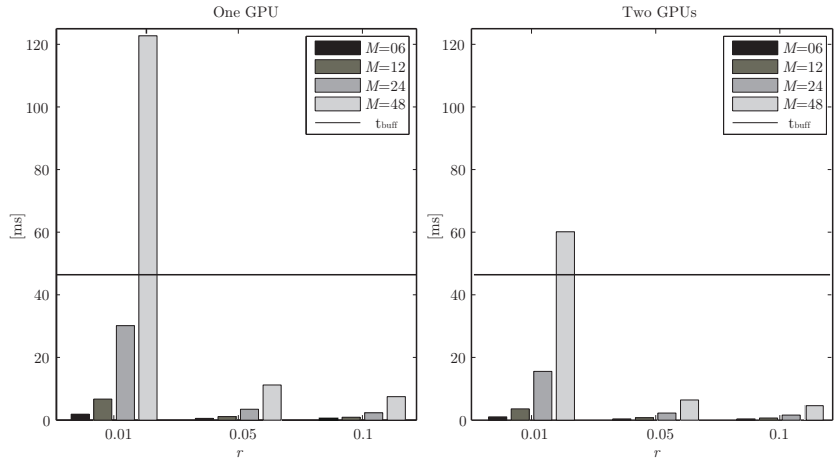


Figure 10: Time t_{proc} for different resolutions and number of microphones.

multiple cores are included in a GPU device. On the other hand, the system is composed of several GPUs. The analyzed computational performance indicates that the algorithm is scalable, so that the time employed to estimate the source location is reduced with the number of GPU devices.

Secondly, we have evaluated the relation existing among localization accuracy, number of microphones and available computational resources. While most works take into account computational issues or performance issues only, we have approached both aspects from a practical implementation perspective. To this end, simulated experiments going from a noiseless anechoic case to a noisy and highly reverberant case have been analyzed, studying the overall performance with a varying number of microphones and spatial resolutions. Results show that thanks to the GPU computational resources, a fine spatial search grid and a high number of microphones can be used to improve the localization accuracy and the robustness of the system.

Finally, the presented implementation exploits the resources of GPUs with Kepler architecture, which can be currently found in new generation mobile devices such as Google's Nexus 9 tablet. Thus, the proposed implementation can be successfully run on embedded mobile devices.

Acknowledgments

This work has been partially funded by the Spanish Ministerio de Economía y Competitividad (TEC2009-13741, TEC2012-38142-C04-01, and TEC2012-37945-C02-02), Generalitat Valenciana PROMETEO 2009/2013, and Universitat Politècnica de València through Programa de Apoyo a la Investigación y Desarrollo (PAID-05-11 and PAID-05-12).

References

Allen, J. B., & Berkley, D. A. (1979). Image method for efficiently simulating small-room acoustics. *J. Acoust. Soc. Am.*, *65*, 943–950.

Belloch, J. A., Ferrer, M., Gonzalez, A., Martinez-Zaldivar, F., & Vidal, A. M. (2013a). Headphone-based virtual spatialization of sound with a GPU accelerator. *J. Audio Eng. Soc*, *61*, 546–561.

Belloch, J. A., Gonzalez, A., Martinez-Zaldivar, F. J., & Vidal, A. M. (2011). A real-time crosstalk canceller on a notebook GPU. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME)* (pp. 1 –4). Barcelona, Spain.

Belloch, J. A., Gonzalez, A., Vidal, A. M., & Cobos, M. (2013b). Real-Time Sound Source Localization on Graphics Processing Units. In *Proc. of the International Conference on Computational Science, ICCS 2013*. Barcelona, Spain.

Bilbao, S., & Webb, C. J. (2013). Physical modeling of timpani drums in 3D on GPGPUs. *J. Audio Eng. Soc*, *61*, 737–748.

Bradford, R., Ffitch, J., & Dobson, R. (2011). Real-time sliding phase vocoder using a commodity GPU. In *Proc. of ICMC 2011*. University of Huddersfield, United Kingdom.

Brandstein, M., & Ward, D. (2001). *Microphone arrays*. Springer.

Calderoni, L., Ferrara, M., Franco, A., & Maio, D. (2015). Indoor localization in a hospital environment using Random Forest classifiers. *Expert Systems with Applications*, *42*, 125 – 134. doi:http://dx.doi.org/10.1016/j.eswa.2014.07.042.

Chen, J., Benesty, J., & Huang, Y. (2006). Time delay estimation in room acoustic environments: an overview. *EURASIP Journal on Applied Signal Processing*, *2006*, 1–19.

Cobos, M., Marti, A., & Lopez, J. J. (2011). A modified SRP-PHAT functional for robust real-time sound source localization with scalable spatial sampling. *IEEE Signal Processing Letters*, *18*, 71–74.

Cook, S. (2013). *A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann.

CUDA (2015). Nvidia CUDA Developer Zone. https://developer.nvidia.com/cuda-downloads. (accessed 2015 February 02).

Dazevedo, E., & Hill, J. (2012). Parallel LU Factorization on GPU Cluster. *Procedia Computer Science*, *9*, 67 – 75. doi:10.1016/j.procs.2012.04.008. Proceedings of the International Conference on Computational Science, ICCS 2012.

DiBiase, J. H. (2000). *A high accuracy, low-latency technique for talker localization in reverberant environments using microphone arrays*. Ph.D. thesis Brown University Providence, RI.

DiBiase, J. H., Silverman, H. F., & Brandstein, M. S. (2001). Robust localization in reverberant rooms. In M. S. Brandstein, & D. Ward (Eds.), *Microphone Arrays: Signal Processing Techniques and Applications* chapter 8. (pp. 157–180). Berlin, Germany: Springer-Verlag.

Do, H., & Silverman, H. F. (2007). A fast microphone array SRP-PHAT source location implementation using coarse-to-fine region contraction (CFRC). In

Proc. of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New Paltz, USA.

Hamilton, B., & Webb, C. J. (2013). Room acoustics modelling using GPU-accelerated finite difference and finite volume methods on a face-centered cubic grid. In *Proc. of Conference on Digital Audio Effects (DAFx-13)*. Maynooth, Ireland.

Harris, M. (2014). Optimizing Parallel Reduction in CUDA NVIDIA. `http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf`. (accessed 2014 August 27).

Huang, Q., & Wang, T. (2014). Acoustic source localization in mixed field using spherical microphone arrays. *EURASIP Journal on Applied Signal Processing*, *90*, 1–16.

Jetson (2015). Mobile GPU: Jetson. `http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf`. (accessed 2014 November 22).

K20 (2014). NVIDIA Kepler Architecture. `http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`. (accessed 2014 August 27).

Kloss, Y., Shuvalov, P., & Tcheremissine, F. (2010). Solving Boltzmann equation on GPU. *Procedia Computer Science*, *1*, 1083 – 1091. Proceedings of the International Conference on Computational Science, ICCS 2010.

Knapp, C. H., & Carter, G. C. (1976). The Generalized Correlation Method for Estimation of Time Delay. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *ASSP-24*, 320–327.

Kodagoda, S., & Sehestedt, S. (2014). Simultaneous people tracking and motion pattern learning. *Expert Systems with Applications*, *41*, 7272 – 7280. doi:`http://dx.doi.org/10.1016/j.eswa.2014.05.019`.

Kuttruff, H. (2000). *Room acoustics*. Abingdon, Oxford, UK: Taylor & Francis. 368 pages.

Liang, Y., Cui, Z., Zhao, S., Rupnow, K., Zhang, Y., Jones, D. L., & Chen, D. (2012). Real-time implementation and performance optimization of 3D sound localization on GPUs. In *DATE'12* (pp. 832–835).

Liu, W., Schmidt, B., Voss, G., & Muller-Wittig, W. (2007). Streaming algorithms for biological sequence alignment on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *18*, 1270–1281. doi:`http://doi.ieeecomputersociety.org/10.1109/TPDS.2007.1069`.

Lorente, J., Ferrer, M., , De Diego, M., & Gonzalez, A. (2014). GPU Implementation of Multichannel Adaptive Algorithms for Local Active Noise Control. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *22*, 1624 – 1635. doi:`10.1109/TASLP.2014.2344852`.

Lorente, J., Ferrer, M., De Diego, M., Belloch, J. A., & Gonzalez, A. (2013). GPU implementation of a frequency-domain modified filtered-X LMS algorithm for multichannel local active noise control. In *Proc. of the 52nd AES Conference*. Guildford, United Kingdom.

Lorente, J., Gonzalez, A., Ferrer, M., Belloch, J. A., De Diego, M., Piñero, G., & Vidal, A. M. (2012). Active noise conrol using Graphics Processing Units. In *Proc of the International Congress on Sound and Vibration*. Vilnius, Lithuania.

Madhu, N., & Martin, R. (2008). Advances in digital speech transmission. chapter Acoustic Source Localization with Microphone Arrays. (pp. 135–166). New York, NY, USA: Wiley.

Marti, A., Cobos, M., & Lopez, J. J. (2013). A steered response power iterative method for high-accuracy acoustic source location. *Journal of the Acoustical Society of America*, *134*.

Matsumoto, K., Nakasato, N., Sakai, T., Yahagi, H., & Sedukhin, S. G. (2011). Multi-level Optimization of Matrix Multiplication for GPU-equipped Systems. *Procedia Computer Science*, *4*, 342 – 351. Proceedings of the International Conference on Computational Science, ICCS 2011.

Mazur, R., Jungmann, J., & Mertins, A. (2011). On CUDA implementation of a multichannel room impulse response reshaping algorithm based on p-norm optimization. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)* (pp. 305 –308). doi:10.1109/ASPAA.2011.6082310.

Nexus (2015). Google's nexus 9. http://blogs.nvidia.com/blog/2014/10/17/nvidia-tegra-k1-google-nexus-9/. (accessed 2015 January 11).

openMP (2014). openMP API Specifications. http://www.openmp.org. (accessed 2014 June 05).

Peruffo Minotto, V., Rosito Jung, C., Gonzaga da Silveira, L., & Lee, B. (2012). GPU-based approaches for real-time sound source localization using the SRP-PHAT algorithm. *International Journal of High Performance Computing Applications*, . doi:10.1177/1094342012452166.

Said, A., Lee, B., & Kalker, T. (2013). *Fast steered response power computation in 3D spatial regions*. Technical Report HPL-2013-40 HP Labs Palo Alto, USA.

Savioja, L. (2010). Real-time 3D finite-difference time-domain simulation of low- and mid-frequency room acoustics. In *Proc. of the Int. Conf. Digital Audio Effects*. Graz, Austria.

Savioja, L., Välimäki, V., & Smith, J. O. (2011). Audio Signal Processing using Graphics Processing Units. *J. Audio Eng. Soc*, *59*, 3–19.

Schneider, M., Schuh, F., & Kellermann, W. (2012). The Generalized Frequency-Domain Adaptive Filtering Algorithm Implemented on a GPU for

Large-Scale Multichannel Acoustic Echo Cancellation. *Speech Communication; 10. ITG Symposium; Proceedings of*, (pp. 1 –4).

Seewald, L. A., Gonzaga, L., Veronez, M. R., Minotto, V. P., & Jung, C. R. (2014). Combining SRP-PHAT and two Kinects for 3D Sound Source Localization. *Expert Systems with Applications*, *41*, 7106 – 7113. doi:http://dx.doi.org/10.1016/j.eswa.2014.05.033.

Southern, A., Murphy, D., Campos, G., & Dias, P. (2010). Finite difference room acoustic modelling on a General Purpose Graphics Processing Unit. In *Proc. of the 128th AES Convention*. London, United Kingdom.

Vanek, J., Trmal, J., Psutka, J., & Psutka, J. (2012). Optimized Acoustic Likelihoods Computation for NVIDIA and ATI/AMD Graphics Processors. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*, 1818–1828. doi:10.1109/TASL.2012.2190928.

Wang, R., Wang, X., & Kim, M. J. (2011). Motivated learning agent model for distributed collaborative systems. *Expert Systems with Applications*, *38*, 1079 – 1088. doi:http://dx.doi.org/10.1016/j.eswa.2010.05.003. Intelligent Collaboration and Design.

Webb, C. J., & Bilbao, S. (2011). Computing room acoustics with CUDA - 3D FDTD schemes with boundary losses and viscosity. In *Proc of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Prague, Czech Republic.

Xu, B., Sun, G., Yu, R., & Yang, Z. (2013). High-Accuracy TDOA-Based Localization without Time Synchronization. *IEEE Transactions on Parallel and Distributed Systems*, *24*, 1567–1576. doi:10.1109/TPDS.2012.248.

Zhao, Y., & Lau, F. C. (2013). Implementation of Decoders for LDPC Block Codes and LDPC Convolutional Codes Based on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *99*, 1. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.52.

Appendix A. GPU and CUDA

GPUs are wellknown for their potential in highly parallel data processing. A GPU is composed by multiple Stream Multiprocessors (SM) where, for 3.5 capability (Kepler architecture (K20, 2014)), there are 192 pipelined cores per SM[1]. In the CUDA model, the programmer defines the kernel function where the code to be executed on the GPU is written. A grid configuration, which defines the number of threads and how they are distributed and grouped, must be built into the main code (threads are grouped into thread blocks, and thread blocks configure a grid that is organized in three dimensions, denoted as `BlockIdx.x`, `BlockIdx.y`, and `BlockIdx.z`). The total number of threads launched in a kernel by means of thread blocks can exceed the number of physical cores. At runtime, the kernel distributes all the thread blocks among SMs. Each SM can host up to 16 thread blocks. If the number of blocks exceeds the resources of the GPU, these blocks wait until other blocks finish in order to be hosted later.

A GPU device has a large amount of off-chip device memory (*global-memory*) and a fast on-chip memory (*shared-memory*, *registers*). As its name indicates, the *shared-memory* is normally used when multiple threads must share data. There are also read-only cached memories called *constant-memory* and *texture-memory*. The first memory is optimized for broadcast (i.e., when all the threads read the same memory location), while the second one is more oriented to graphics. Figure A.1 shows how the GPU architecture is organized. Advanced GPU devices (beyond 2.x capability) come with an L1/L2 cache hierarchy that is used to cache *global-memory*. Cache L1 uses the same on-chip memory as *shared-memory*; how much of the on-chip memory is dedicated to L1 is set for each kernel call.

[1] At the time this paper was written, the most advanced GPU device was K20c with Kepler architecture which is the one considered throughout this work

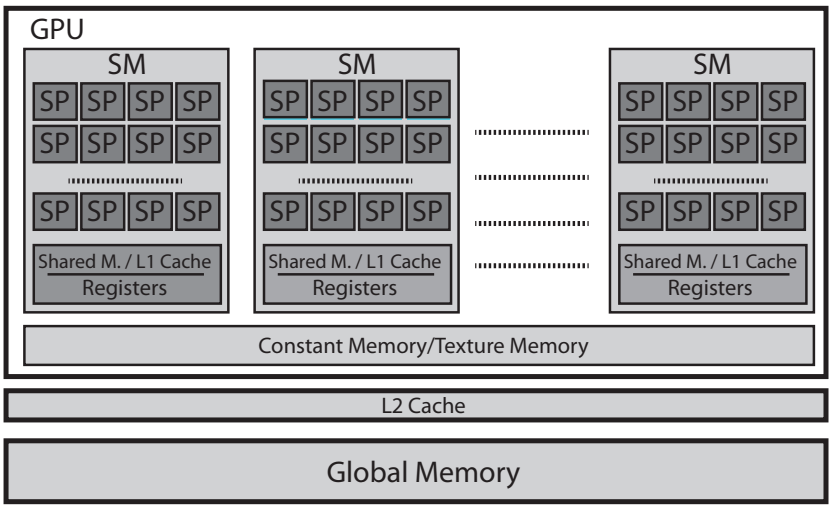


Figure A.1: The GPU is configured by 16 Stream Multiprocessors (SMs), each of which has 192 pipelined cores (SP).

Appendix A.1. Streams on GPU

Streams are virtual work queues on the GPU. They are used for asynchronous operation, (i.e, the control of the program returns to the CPU immediately). Operations assigned to the same stream are executed in order and sequentially. Multiple streams can be defined on CUDA programming; however, up to 32 streams are available to be independently run on the GPU thanks to the *Hyper-Q* technology that is presented in hardware with 3.5 capability (Cook, 2013).

Different streams may execute their assigned operations out of order with respect to one another or concurrently. Thus, when a launched kernel does not require all the GPU resources, these could be used for another kernel that was launched from a different stream. Hence, streams allow multiple kernels to be launched concurrently. Following this idea, data transfer between CPU and GPU can also be overlapped with kernel computations and other transfers whenever they are carried out in different streams. If the data transfers are not assigned to any stream queue, they are executed synchronously and in an isolated way, (i.e., the CPU waits until all the previous operations have finished). GPU kernels are always launched asynchronously by the CPU (regardless of

whether or not they are scheduled on a stream queue). Thus, data transfers are usually used as a synchronization barrier.

Figure A.2 illustrates the parallelization obtained using streams when $M = 4$ (number of microphones) for steps 1,2, and 3 from Section 3 in the main article. Note that, in Fig.A.2, the alternative Kernel A' would have a CUDA grid with dimensions ($\frac{2L}{128} \times M$).

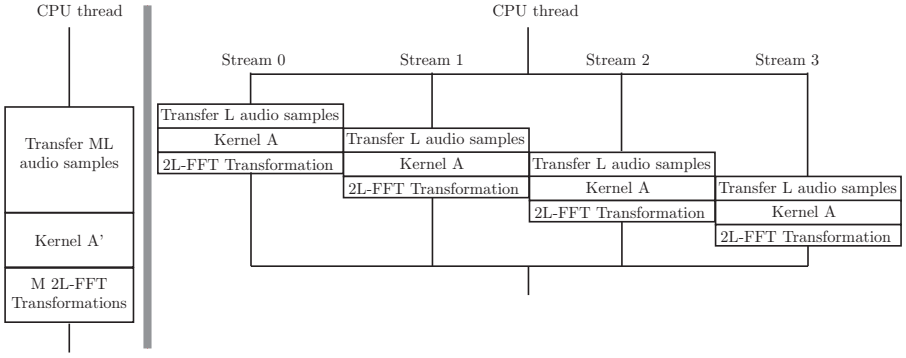


Figure A.2: Parallelization obtained with streams when M=4.

Appendix A.2. Multi-GPU programming with multicore

One of the standards that allows for multicore processing is openMP (openMP, 2014). This standard works by using a *fork/join* pattern, that is, parallel regions are specified by the programmer. The CPU code runs sequentially and at some point hits a section where work can be distributed into several processors that perform the computations (CPU core spans several CPU threads). Afterwards, when all the computations are completed, all the CPU threads converge to a single thread again, which is called the *master* thread.

If a machine has a multicore processor and several GPUs, the parallelization can be achieved by defining a number of threads in the parallel region equal to the number of GPUs. In this sense, each CPU thread deals with a GPU. This is very important since a CPU thread is bound with a GPU context. Thus, all subsequent CUDA calls (e.g. `cudaMalloc`) allocate memory only on its corresponding GPU (Cook, 2013).

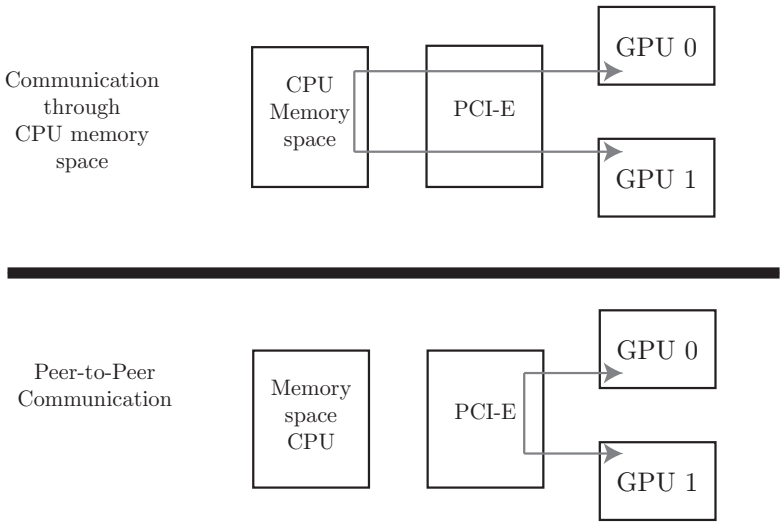


Figure A.3: The UVA feature reduces data-transfer time among GPUs by using peer-to-peer communication (bottom).

Recent CUDA releases (beyond 2.x capability and CUDA SDK 4.x) allow the time employed in data transfers among GPUs to be reduced by using the UVA (Unified Virtual Addressing) feature. That means that inter-GPU communication (peer-to-peer, P2P) can also be performed without routing the data through the CPU, saving PCI-E bandwidth. Before the appearance of these recent features, communication among GPUs had to be carried out through memory space in the CPU, as shown in Figure A.3.

Appendix B. Multi-GPU Parallelization strategy involving GCC and SRP matrices

The challenge of this strategy consists in parallelizing the computation of the **GCC** matrix. Initially, all the GPUs must have access to this matrix since each point of the **SRP** matrix requires a contribution from each pair of microphones (each row of the **GCC** matrix).

The strategy that we present aims at achieving a good trade-off between the total operations carried out in each GPU and the number of transferred audio

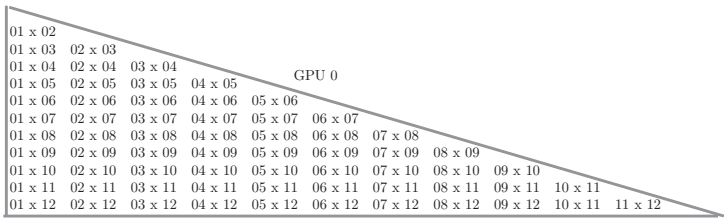
buffers. For example, if the number of microphones is $M = 12$, the number of pairs to compute in **GCC** matrix is $Q = 66$. These pairs are distributed among the N_{GPU} in a pseudo-triangular way. Figure B.1 shows the distribution of the computation and audio buffers among 2, 3 and 4 GPUs. The notation 01 x 05, indicates the element-wise multiplication of vector 1 and vector 5 of all computed vectors $\mathbf{f}_l$, $l = 0, \ldots, M-1$ (see step 2 of Section 3). Note that the GPU that performs more multiplications deals with less audio buffers, minimizing the data transfers between CPU and GPU. This triangular structure can be considered independently of the number of microphones.

Finally, after the distributed computation of the **GCC** matrix, all GPUs need all of the rows of the **GCC** matrix in order to compute their corresponding ν/N_{GPU} elements of the **SRP** matrix. The use of UVA (see Appendix A.2) allows each GPU to access other GPU via peer-to-peer over the PCI-E bus rather than copying data back to the host and then to another GPU. Thus, each GPU transparently accesses the memories of other GPUs by just referencing a memory location.

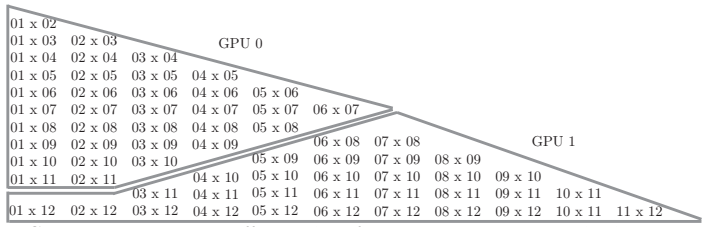
Appendix B.1. Basic Implementation using two GPUs

Using all the parallelization techniques presented in Appendix A, the SRP-PHAT algorithm is implemented on two GPUs as follows:

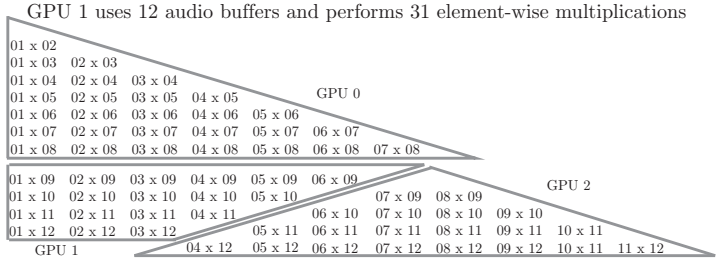
1. A parallel region is created with two CPU threads. Each CPU thread is bound with a GPU.
2. Since different audio buffers are received in the system, each CPU thread independently and asynchronously sends its corresponding audio buffers to its GPU by using stream parallelization. The Kernels A and the FFTs are computed for each channel inside the streams.
3. As in step 2 of Section 3, stream synchronization is addressed. Only one stream is used to compute the rows of the **GCC** matrix. According to Figure B.1, in the case of $M = 12$, one GPU would compute 35 vectors and the other one would compute 31 vectors.



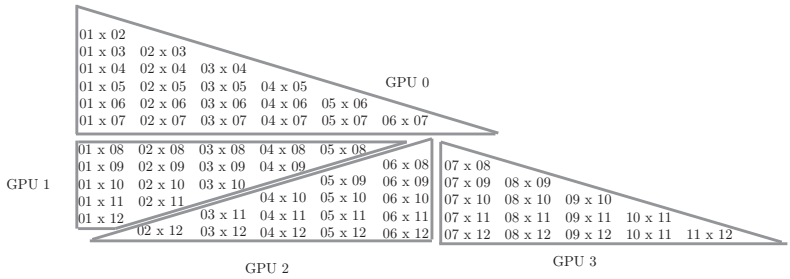
GPU 0 uses 12 audio buffers and performs 66 element-wise multiplications



GPU 0 uses 11 audio buffers and performs 35 element-wise multiplications
GPU 1 uses 12 audio buffers and performs 31 element-wise multiplications



GPU 0 uses 08 audio buffers and performs 28 element-wise multiplications
GPU 1 uses 10 audio buffers and performs 18 element-wise multiplications
GPU 2 uses 09 audio buffers and performs 20 element-wise multiplications



GPU 0 uses 07 audio buffers and performs 21 element-wise multiplications
GPU 1 uses 09 audio buffers and performs 15 element-wise multiplications
GPU 2 uses 10 audio buffers and performs 15 element-wise multiplications
GPU 3 uses 06 audio buffers and performs 15 element-wise multiplications

Figure B.1: Distribution of the audio buffers in order to compute the rows of the **GCC** matrix when N_{GPU} is 1,2,3 and 4.

Table B.1: Speed up between strategies.

r_x,r_y	$M = 6$	$M = 12$	$M = 24$	$M = 48$
0.01	30.097	35.443	36.968	31.649
0.05	12.259	24.043	31.291	43.313
0.1	4.815	9.861	15.310	21.249

4. By using UVA, each GPU has access to the whole **GCC** matrix in order to compute $\nu/2$ elements of the **SRP** matrix and locates a maximum value among the computed elements.
5. Each GPU transfers back to the CPU its maximum value and its location inside the **SRP** matrix. Then, a synchronization barrier for both CPU threads is set followed by an *openMP* section that is only executed by the master thread. This thread compares the two maximum values and chooses the greatest one, getting its location. This location indicates the sound source position.

Appendix B.2. Comparison between strategies

Table B.1 shows the speed up that the implementation strategy presented in section 3.3 achieves with respect to the strategy presented in Appendix B. Two important aspects significantly penalize the performance of this strategy in comparison with the strategy in section 3.3. First, since each GPU does not contain the whole **GCC** matrix, each GPU must access the *global-memory* of the other GPU in order to compute the **SRP** matrix; second, after computing the corresponding elements of the **GCC** matrix, both GPUs must be synchronized.