# A GPU Implementation of the SRP-PHAT Sound Source Localization Algorithm

Luiz G. da Silveira Jr., Vicente P. Minotto
Universidade do Vale do Rio dos Sinos
Dept. of Computer Science
Av. Unisinos, 950.
São Leopoldo, RS, 93022-000, Brazil
lgonzaga@unisinos.br, vminotto@gmail.com

Claudio R. Jung
Universidade Federal do Rio Grande do Sul
Institute of Informatics
Av. Bento Gonçalves, 9500
Porto Alegre, RS, 91501-970, Brazil
crjung@inf.ufrgs.br

Bowon Lee
Hewlett-Packard Labs
Mult. Comm. and Net. Lab
1501 Page Mill Road
Palo Alto, CA 94304, USA
bowon.lee@hp.com

***Abstract*—Microphone arrays have been widely used for hands-free speech acquisition systems such as teleconferencing or automatic speech recognition. They typically require sound source localization for subsequent multichannel signal processing algorithms such as beamforming for enhancing speech signals. Steered response power with phase transform (SRP-PHAT) source localization has been the most popular method thanks to its robustness against reverberation. In most cases, the SRP-PHAT requires exhaustive search methods, which makes its real-time implementation a challenging task. In order to address this problem, this paper presents a GPU (Graphic Processor Unit) implementation that takes advantage of several parallel steps involved the the computation of the SRP-PHAT.**

***Index Terms*—Sound source localization, Steered response power, Microphone array, Parallel processing, GPU implementation**

## I. Introduction

Microphone array systems have gained popularity for applications that require robust estimation of speech signals against noise, interfering sources, and reverberation [1]. In most applications, they require sound source localization (SSL) for subsequent multichannel signal processing algorithms such as beamforming, dereverberation, echo cancellation, etc.

Sound source localization algorithms for speech acquisition can be mainly divided into two categories. The first is a two-stage method based on time difference of arrival (TDOA) estimations followed by triangulation [2] and the second is a search space-based method for finding a source location with the maximum steered response power (SRP) [3] or the maximum-likelihood (ML) [4], [5], among all candidate locations. The two-stage method is computationally inexpensive, but is subject to fail due to TDOA estimation errors [6]. Unfortunately, this is often the case in typical acoustic environments for speech acquisition systems. The search space method is robust, but requires a predefined search space resulting in computational complexity proportional to the number of candidate source locations. For capturing speech from meeting participants in a typical conference room, for example, the SRP-PHAT may require much higher computational complexity for real-time applications on currently available processors with moderate processing power.

Zhang et al. [4] showed that we can reduce the complexity of the SRP-PHAT method from $O(M^2)$ to $O(M)$, where $M$ is the number of microphones. Do et al. [7] proposed an iterative hierarchical method called *stochastic region contraction* (SRC) to reduce the effective number of candidate sources for SRP-PHAT. Even with these optimizations, a significant computational load is still needed for each candidate source location.

In this paper, we present a GPU (Graphics Processing Unit) implementation of the SRP-PHAT source localization suitable for real-time applications with a large search space. This paper is organized as follows. In Section II, we describe microphone array signal models and the SRP-PHAT method for SSL. Section III describes our GPGPU implementation of the SRP-PHAT method for real-time applications. Section IV describes experimental results followed by discussions on Section V. Section VI concludes this paper.

## II. Sound Source Localization

For an array of $M$ microphones, the signal $x_m(t)$ captured at the $m^{th}$ microphone can be expressed as follows

$$x_m(t) = s(t - \tau_m^{\mathbf{q}}) + v_m(t), \text{ for } m = 1, 2, \cdots, M \quad (1)$$

where $s(t)$ is the source signal, $v_m(t)$ is a term due to reverberation, interferences, and background noise, and $\tau_m^{\mathbf{q}}$ denotes propagation delay of the signal $s(t)$ from a source location $\mathbf{q}$ to the $m^{th}$ microphone. For simplicity, we assume that the differences in attenuation caused by propagation from the source to the microphones are negligible. With this signal model, we can express a vector $\mathbf{X}_\omega = [X_1(\omega), X_2(\omega), \cdots, X_M(\omega)]^T$ of the microphone signals in the frequency domain as

$$\mathbf{X}_\omega = S(\omega)\mathbf{D}_\omega^{\mathbf{q}} + \mathbf{V}_\omega, \quad (2)$$

where

$$\mathbf{D}_\omega^{\mathbf{q}} = [e^{-j\omega\tau_1^{\mathbf{q}}}, e^{-j\omega\tau_2^{\mathbf{q}}}, \cdots, e^{-j\omega\tau_M^{\mathbf{q}}}]^T$$
$$\mathbf{V}_\omega^{\mathbf{q}} = [V_1(\omega), V_2(\omega), \cdots, V_M(\omega)]^T$$

denote delay and noise vectors respectively. Thus, SSL is a problem of finding a source location $\mathbf{q}$ corresponding to a set of delay vectors $\mathbf{D}_\omega^{\mathbf{q}}$ given observations $\mathbf{X}_\omega$.

### A. Steered Response Power Method

The *steered response power* (SRP) method computes the output power of a filter-and-sum beamformer for all possible source locations and then selects one with the maximum power [3]. In the frequency domain, the power of a candidate source location $\mathbf{q}$ can be expressed as

$$P(\mathbf{q}) = \sum_{m=1}^{M} \sum_{l=1}^{M} \int \Psi_{ml}(\omega) X_m(\omega) X_l^*(\omega) e^{j\omega(\tau_m^{\mathbf{q}} - \tau_l^{\mathbf{q}})} d\omega, \tag{3}$$

where $\Psi_{ml}(\omega)$ is the frequency weighting for more accurate source location estimate. One of the most popular weighting is the *phase transform* (PHAT) [8]

$$\Psi_{ml}(\omega) = \frac{1}{|X_m(\omega) X_l^*(\omega)|}, \tag{4}$$

and Zhang et al. [4] showed that Eq. (3) with the PHAT weighting is equivalent to

$$P(\mathbf{q}) = \int \left| \sum_{m=1}^{M} \frac{X_m(\omega)}{|X_m(\omega)|} e^{j\omega\tau_m^{\mathbf{q}}} \right|^2 d\omega, \tag{5}$$

which reduces the number of computations by a factor of $M$.

After computing $P(\mathbf{q})$, we can find the source location as

$$\hat{\mathbf{q}} = \arg\max_{\mathbf{q} \in \mathcal{Q}} P(\mathbf{q}), \tag{6}$$

where $\mathcal{Q}$ denotes the search space of all potential source locations.

## III. GPU Processing of the SRP-PHAT Computation

### A. Common approaches for GPU implementation of generic algorithms

While the SRP-PHAT method requires exhaustive computational processing, GPU can provide massively multithreaded manycore chips with hundreds of scalar processors, running tens of thousands of threads, and giving more than one TFLOP peak performance [9]. GPU vendors usually offer parallel programming languages with a corresponding API (Application Programming Interface) that allow users to develop generic purpose applications running on their GPU units. An interesting property of such programming languages is that the user does not need to have knowledge about the traditional graphics pipeline model and its API.

Nowadays, the main GPU computing platform is nVIDIA's CUDA (Compute Unified Device Architecture) [10]. Besides CUDA, there is an open standard for GPU Computing, called OpenCL (Open Computing Language), which is distributed freely and provides compatibility to the main GPU manufacturers nVIDIA and ATI. However, CUDA has been more stable and also more extensively explored for high performance computing on GPUs, while OpenCL focuses on portability and heterogeneous platforms (CPU and GPU) jointly for increased performance. In fact, OpenCL implementation is still incipient and cannot offer high performance results when compared with CUDA.

CUDA is scalable parallel programming model and accompanied by an API for GPU (and other parallel processors), which allows direct programming in C/C++ language by approaching data-parallel aspects and workload. In practice, we can consider that CUDA extends C/C++ languages, providing all support to exploit massively the parallelism for non-graphics applications.

However, to successfully achieve high performance using CUDA, the problem data must be initially parallelized, preferably with all data fully transfered to device memory (GPU memory), to avoid significant data transfers from the main memory to the device memory (which are costly and may be the bottleneck of GPU implementations of genetic algorithms). Then, the application can be scaled to hundred of processor cores and thousand of concurrently threads.

### B. SRP-PHAT Algorithm on CUDA

The computational cost of the SRP-PHAT algorithm depends on several parameters, such as the size of the microphone array (i.e. the number of microphones), the size of the discretized search space $\mathcal{Q}$, and the sampling frequency of the data acquisition device. In this work, we consider that $\mathcal{Q}$ consists of a $N \times N$ 2D grid of equally spaced points, and that we have $M + 1$ microphones in the array.

Before providing specific details of our implementation, it is important to mention that given two discretized positions $\mathbf{q}_1$ and $\mathbf{q}_2$ in the search space $\mathcal{Q}$, the values $P(\mathbf{q}_1)$ and $P(\mathbf{q}_2)$ in Equation (5) can be done in a completely independent manner, which allows a highly parallel implementation of the SRP-PHAT algorithm.

There are different possibilities to tackle the CUDA implementation of the SRP-PHAT algorithm. In this paper, we have considered one thread by each candidate sound location in the 2-D grid in the CUDA configuration. Then, the output of each thread is one calculation of $P(\mathbf{q})$ as given in Equation (5), where $\mathbf{q}$ corresponds to an element in $\mathcal{Q}$. As consequence, the scalability of the algorithm running on GPU highly depends on size of $\mathcal{Q}$, which grows quadratically with $N$. The load of each thread depends on the size of the audio buffer (which is used to compute the FFT and approximate in integral in Equation (5), and also on the number of microphones in the array. Assuming a constant buffer size, the load of each thread grows linearly with $M$.

Hence, in the proposed CUDA implementation of the SRP-PHAT algorithm, increasing the size of $\mathcal{Q}$ should not affect significantly the overall cost, as long as there are enough parallel threads running on the GPU. On the other hand, increasing the number of microphones should increase linearly the overall cost of the implementation.

## IV. Experiments

We have planned a set of experiments, in order to evaluate and validate the GPU-based SRP-PHAT implementation. The prototype has been tested considering different hardware setup, varying mainly the GPUs. The measurements of elapsed times

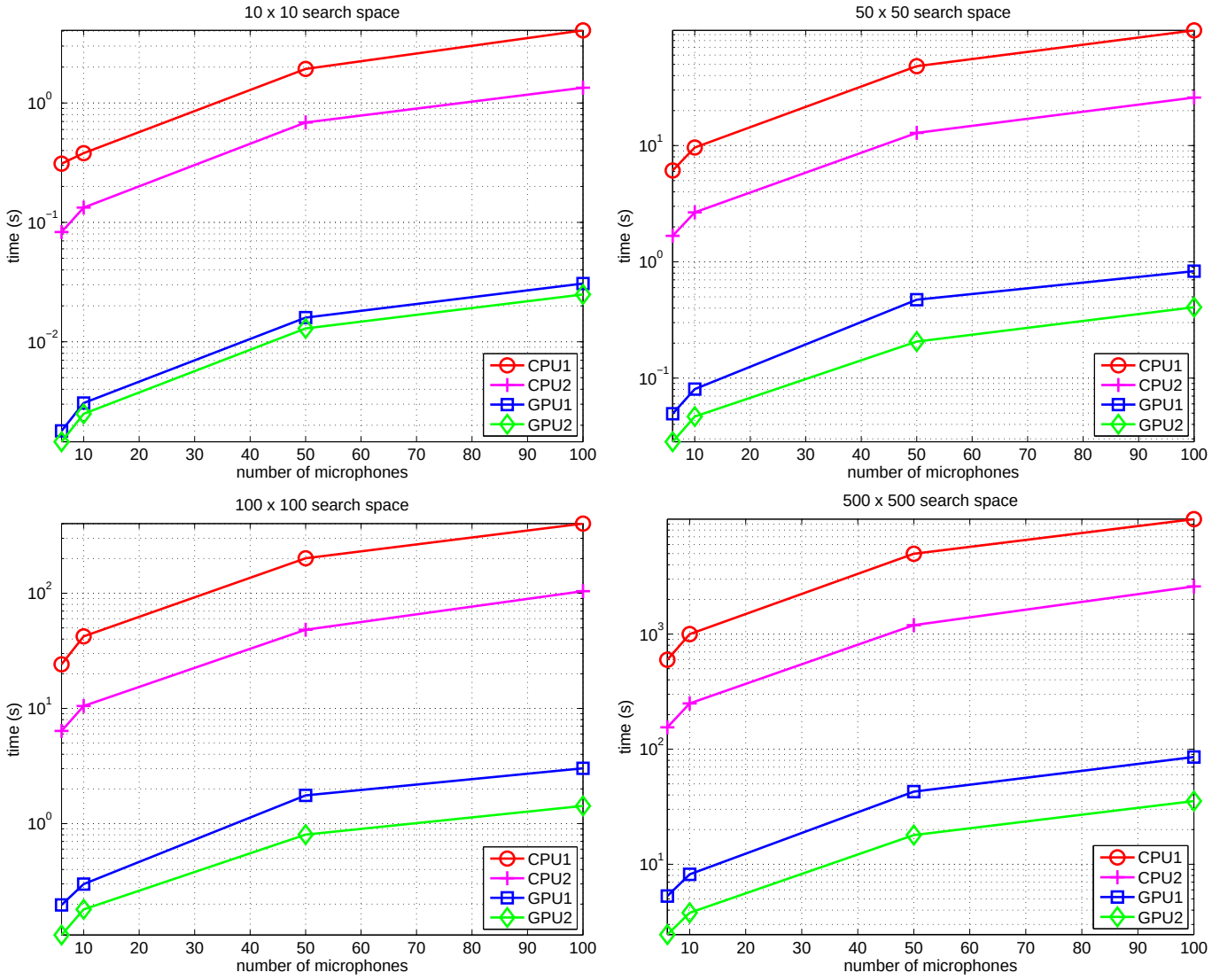


Fig. 1. Execution times varying the number of microphones and the size of the search space.

for different combinations of number of microphones ($M$)[1] and grid size ($N \times N$), assuming an acquisition rate of 44 kHz and a buffer size of 4096 samples (so that real time execution should process 10 SRP-PHATs per second).

For the CPU experiments, we used a C++ implementation of the SRP-PHAT algorithm running on a notebook with an Intel Core i7-720QM processor (4 1.6 GHz cores, 8 threads), and the proposed CUDA implementation running on a GeForce 9600GT GPU with 512MB dedicated memory, and a GeForce 9800GTX GPU, with 768MB dedicated memory. For the CPU tests, we used a single-threaded experiment (called CPU1 in the experiments) and a multi-threaded implementation with eight threads (CPU2). The experiments with the GeForce 9600GT and 9800GTX cards are called, respectively, GPU1 and GPU2.

Fig. 1 shows the obtained execution times for the four tested algorithms (CPU1, CPU2, GPU1 and GPU2), varying the size of the search size. Each Figure illustrates the evolution of the execution times as the number of microphones is increased, and the vertical axis (time) is shown in logarithmic scale for a better visualization of the differences. Fig. 2 shows a comparison of the execution times of the four analyzed implementations fixing the number of microphones (50), and varying the size of the search space. An analysis of this results is presented in the next section.

[1]Since we have a 6-microphone array, results with $M > 6$ were simulated.

## V. Discussions

It can be observed in Fig. 1 that the execution times obtained with the CUDA implementation running on both a lower-end GPU (GeForce 9600GT) and a higher-end GPU (GeForce 9800GTX) were similar for a small search region ($10 \times 10$), but the difference increases as the search space gets larger. In fact, results obtained with the 9800GTX card are approximately 1.2 times faster than the 9600GT card for a $10 \times 10$ search

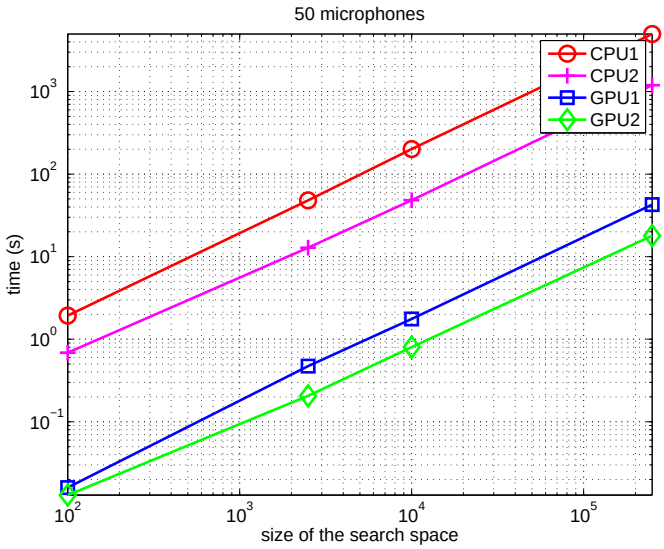


Fig. 2. Execution times for a 50-microphone array, varying the size of the search region.

| | Size of the search space | | | |
|---|---|---|---|---|
| | $10 \times 10$ | $50 \times 50$ | $100 \times 100$ | $500 \times 500$ |
| CPU2 | 3.10 | 3.82 | 3.95 | 3.95 |
| GPU1 | 138.22 | 120.20 | 129.50 | 120.50 |
| GPU2 | 170.47 | 242.25 | 261.00 | 273.64 |

TABLE I
SPEED-UPS OF CPU2, GPU1 AND GPU2 COMPARED TO CPU1 FOR DIFFERENT SEARCH SPACES

region, and approximately 2.3 times faster in the $500 \times 500$ search region. It should be noted that the 9800GTX card is equipped with 128 cores [11] compared to 64 cores that the 9600GT is equipped with [12], which clearly indicates that having a larger number of cores is beneficial for computing the SRP-PHAT with large search spaces.

Fig. 1 also shows that our CUDA implementation presented lower execution times when compared to a single-threaded implementation on a high-end CPU (Intel Core i7-720QM processor) in all tested configurations (variations in the size of the microphone array and search size). The multi-threaded CPU implementation presented an average speed-up of 3.8 times when compared to the single-threaded implementation, but still considerably lower than both GPUs, particularly when compared to the GeForce 9800GTX results.

Fig. 2 shows a comparison of execution times for a 50-mic array varying the size of the search space, and Table I shows the average speed-ups of CPU2, GPU1 and GPU2 compared to CPU1 for different search spaces. As it can be observed, the execution time difference between the best GPU result (GPU2) and the best CPU result (CPU2) is smaller for small search spaces, and it increases as the search size grows. Considering all evaluated search spaces, GPU2 is in average 63 times faster than CPU2 (the speed-up is around 55 for the $10 \times 10$ search region, and around 69 for the $500 \times 500$ search region).

It is important to mention that our CUDA implementation running on both GPUs can handle real-time performance (i.e. processing time $< 0.1$s) for arrays with 10 microphones, and search regions having up to $50 \times 50$ points (for a $100 \times 100$ region, the running time of the 9800GTX card is $\approx 0.17$s). The single-threaded CPU implementation could not handle real-time processing in any of the tested configurations. Even the multi-threaded approach could only provide real-time performance for a 6-mic array with a $10 \times 10$ search size with the direct implementation of the SRP-PHAT algorithm.

## VI. CONCLUSION

In this paper, we presented a GPU implementation of the SRP-PHAT SSL algorithm. In the proposed implementation, the computation of the SRP-PHAT for each position of the discretized space is related to a GPU thread, which are executed concurrently (up to the maximum number of threads allowed by the graphic hardware). The experimental results showed that GPU implementations on widely available graphic cards may present significant speed-ups when compared to single or multi-threaded implementations on recent (and powerful) CPU configurations.

## ACKNOWLEDGMENT

This work was developed in cooperation with Hewlett-Packard Brasil Ltda. using incentives of Brazilian Informatics Law (Law n 8.2.48 of 1991).

## REFERENCES

[1] M. S. Brandstein and D. B. Ward, *Microphone Arrays: Signal Processing Techniques and Applications*. Berlin, Germany: Springer-Verlag, 2001.
[2] M. Brandstein, J. Adcock, and H. Silverman, "A closed-form location estimator for use with room environment microphone arrays," *IEEE Trans. Speech and Audio Process.*, vol. 5, pp. 45–50, 1997.
[3] J. DiBiase, "A high-accuracy, low-latency technique for talker localization in reverberant environments," Ph.D. dissertation, Brown University, Providence, RI, May 2000.
[4] C. Zhang, Z. Zhang, and D. Florêncio, "Maximum likelihood sound source localization for multiple directional microphones," in *Proc. Int. Conf. Acoust., Speech, and Signal Process.*, vol. I, 2007, pp. 125–128.
[5] B. Lee, T. Kalker, and R. W. Schafer, "Maximum-likelihood sound source localization with a multivariate complex Laplacian distribution," in *Proc. Int. Workshop. Acoustic Echo and Noise Control*, 2008.
[6] S. Bédard, B. Champagne, and A. Stéphenne, "Effects of room reverberation on time-delay estimation performance," in *Proc. Int. Conf. Acoust., Speech, and Signal Process.*, vol. II, 1994, pp. 261–264.
[7] H. Do, H. F. Silverman, and Y. Yu, "A real-time SRP-PHAT source location implementation using stochastic region contraction (SRC) on a large-aperture microphone array," in *Proc. Int. Conf. Acoust., Speech, and Signal Process.*, vol. I, 2007, pp. 121–124.
[8] C. H. Knapp and G. C. Carter, "The generalized correlation method for estimation of time-delay," *IEEE Trans. Acoust., Speech and Audio Process.*, vol. ASSP-24, no. 4, pp. 320–327, 1976.
[9] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.
[10] *CUDA Programming Guide*, NVIDIA Corporation, 2 2010, http://developer.nvidia.com.
[11] *GeForce 9800 GTX*, NVIDIA Corporation, http://www.nvidia.com/object/product_geforce_9800_gtx_us.html.
[12] *GeForce 9600 GT*, NVIDIA Corporation, http://www.nvidia.com/object/product_geforce_9600gt_us.html.