Noname manuscript No.
(will be inserted by the editor)

Accelerating the SRP-PHAT algorithm on Multi and Many-core platforms using OpenCL

Jose M. Badía · Jose A. Belloch · Maximo Cobos · Francisco D. Igual · Enrique S. Quintana-Ortí

Received: date / Accepted: date

Abstract The Steered Response Power with Phase Transform (SRP-PHAT) algorithm is a well-known method for sound-source localization due to its robust performance in noisy and reverberant environments. This algorithm is used in a large number of acoustic applications such as automatic camera steering systems, human-machine interaction, video gaming and audio surveillance. SPR-PHAT implementations require to handle a high number of signals coming from a microphone array and a huge search grid that influences the localization accuracy of the system. In this context, high performance in the localization process can only be achieved using massively parallel computational resources. Different types of multi-core machines based either on multiple CPUs or GPUs are commonly employed in diverse fields of science for accelerating a number of applications, mainly using OpenMP and CUDA as programming frameworks, respectively. This implies the development of multiple source codes which limits the portability and application possibilities. On the contrary, OpenCL has emerged as an open standard for parallel programming that is nowadays supported by a wide range of architectures. In this work, we evaluate an OpenCL-based implementations of the SRP-PHAT algorithm in two state-of-the-art CPU and GPU platforms. Results demon-

Jose M. Badía, Enrique S. Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I de Castelló, Spain
E-mail: {badia,quintana}@uji.es

Jose A. Belloch
Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Spain
E-mail: jbelloc@ing.uc3m.es

Maximo Cobos
Computer Science Department, Universitat de València, Spain
E-mail: maximo.cobos@uv.es

Francisco D. Igual
Depto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Spain
E-mail: figual@pdi.ucm.es

strate that OpenCL achieves close-to-CUDA performance in GPU (considered as upper bound), and outperforms in most of the CPU configurations based on OpenMP.

Keywords SRP-PHAT, OpenCL, multi-core CPUs, GPUs.

1 Introduction

The Steered Response Power with Phase Transform (SRP-PHAT) algorithm is a well-known method for sound-source localization due to its robust performance in noisy and reverberant environments. Applications such as acoustic-based surveillance, gaming, spatial sound and virtual reality can be greatly enhanced with a location-aware system [1].

In order to locate a sound source, it is necessary to process the input signals captured by a set of microphones in real time. The microphones can be arranged either in a distributed configuration or following a specific geometry. The processing is usually based on the computation of the *Generalized Cross-Correlation* (GCC) functions [2,3] of all the microphone pairs in the system. GCCs are usually obtained from the inverse Fourier transform of the cross-power spectral density of the microphone signals, multiplied by a proper spectral weighting function.

The SRP-PHAT method [4] exhibits a massive fine-grain parallelism with the same operations performed over many sets of data. Usually, these data sets correspond to the audio samples of the different audio channels involved in the system. Basically, the SRP-PHAT algorithm evaluates a functional over a fine spatial grid and accepts its maximum value as the most likely source position.

Source localization applications may involve different needs in terms of number of microphones and spatial resolution. In this context, distributed microphone systems may involve a high number of microphones, increasing the computational requirements required to perform localization tasks. Also, the computational complexity of the system may be affected by the total number of candidate locations explored by the algorithm, which may depend on the size of the localization space or the desired spatial resolution. Thus, a scalable and computationally efficient system is of high interest.

Different types of multi-core machines built either from multiple CPUs or GPUs are commonly employed in diverse fields of science for accelerating a number of applications, usually relying on OpenMP and CUDA as programming frameworks, respectively. This implies the development of distinct source codes which limits the portability and application possibilities.

OpenCL consists of an API (Application Programming Interface) for coordinating parallel computation across different devices on a heterogeneous platform; and a cross-platform C-like programming language to program each device. Different vendors provide specific compilers to extract high performance from their architectures. This provides a high level of versatility since an OpenCL-based implementation can be ported and run in a large number of different processors.

Developing an OpenCL-based implementation widens the applications of an implementation. Moreover, the code will work on GPUs that are embedded in system-on-chips (SoCs) that are nowadays working as acoustic sensors in the Internet-of-Things [5]. Finally, besides multi-core CPUs and GPUs, the OpenCL-based implementation will also work in other kind of accelerators, such as Field Programmable Gate Array (FPGA) [6].

In conclusion, taking as a reference a common audio application that deals with the sound-source localization, the objective of this work is to compare the performance of an OpenCL-based implementation on powerful CPUs and GPUs with OpenMP-based and CUDA-based implementations, which are considered to exploit efficiently both architectures, respectively.

The main contributions of this paper are as follows:

- A portable OpenCL implementation of the well-known SRP-PHAT algorithm suited to different types of parallel computing platforms, allowing real-time performance with a large number of microphone channels and fine spatial resolution.
- A set of kernel alternatives that optimize parallelism granularity and memory access costs.
- A practical solution for low power platforms, suitable for IoT-like applications.
- A comparison of OpenCL, CUDA and OpenMP equivalent implementations on state-of-the-art CPU and GPU platforms.

Next, we briefly discuss some related work. In Section 2, we describe the SRP-PHAT algorithm. Section 3 is devoted to implementation issues and section 4 to the performance analysis. Finally, Section 5 provides a few conclusion remarks.

1.1 Related Work

There exist plenty of CUDA-based implementations in the field of audio signal processing [7–9], even approaching sound-source localization algorithms [10, 11], which are constrained to use NVIDIA platforms.

In [12], the SRP-PHAT algorithm is applied on an specific scenario using two Kinects to perform sound source localization. It also includes two preliminary parallel versions of the algorithm, a multi-threaded and an OpenCL implementation.

OpenMP-based implementations were used in [13] for multi-core platforms audio systems based on Beamforming and Wave Field Synthesis. Our previous work [14] also employs OpenMP in order to accelerate the SRP-PHAT algorithm.

While most published approaches are aimed at describing implementations considering specific localization setups, our work focuses on evaluating source localization performance from a computational point of view taking into account two basic parameters: number of microphones and spatial resolution.

2 The SRP-PHAT Algorithm

Consider the output from a microphone l, $m_l(t)$, in a system composed of S microphones. Then, the SRP at the spatial point $\mathbf{x} = [x, y, z]^T$ for a time frame n of length T is defined as

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

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 [4] showed that the SRP can be computed by summing up the GCCs for all possible pairs of the set of microphones in the system. In particular, the GCC for a microphone pair (k, l) is computed 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) [2] has been demonstrated to be a suitable GCC weighting for time delay estimation in reverberant environments:

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

Taking into account the symmetries involved in the computation of Eq. (1), and removing some fixed energy terms [4], the part of $P_n(\mathbf{x})$ that changes with $\mathbf{x}$ is isolated as

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

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 [15]

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

where c is the speed of sound, and $\mathbf{x}_k$ and $\mathbf{x}_l$ are the microphone location vectors.

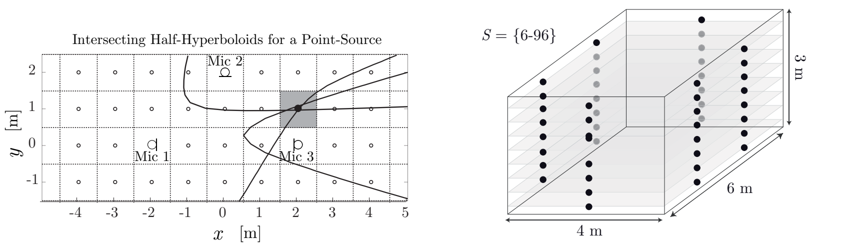

Fig. 1 (a) Intersecting half-hyperboloids for $S = 3$ microphones. Each half-hyperboloid corresponds to a TDOA peak in the GCC. (b) Microphone set-up for {6-96} microphones.

All in all, the SRP-PHAT algorithm consists in evaluating the functional $P'_n(\mathbf{x})$ on a fine grid G with the aim of finding the point-source location $\mathbf{x}_s$ that provides the maximum value [16]:

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

Figure 1(a) 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 (Time Difference of Arrival). 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 largest accumulated value.

2.1 Sequential method

The SRP-PHAT algorithm is usually implemented on a 3D spatial grid with three different resolutions: r_x, r_y and r_z. 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 $P_x = \frac{l_x}{r_x}$, $P_y = \frac{l_y}{r_y}$, $P_z = \frac{l_z}{r_z}$. The real-time implementation of the SRP-PHAT algorithm works with sample buffers of size L. The main steps carried out by the algorithm together with their computational cost are:

1. For each of the S microphones we weight the L samples of all input buffers by a Hamming window vector. This involves SL multiplications, that is, SL floating-point operations (flops).
2. For each of the S microphones we perform an L-FFT resulting in S vectors, each containing L frequency bins. The computation of S FFTs, requires $5SL\log_2 L$ flops that result from $\frac{L}{2}\log_2 L$ complex multiplications and $L\log_2 L$ complex additions.
3. The **GCC** matrix of size $Q \times L$ is computed, where $Q = S(S-1)/2$ represents the number of microphone pairs. For each pair of microphones (i, j), the element `GCC[i, j]` is obtained by combining the ith frequency bin of both microphones. The complex value of the first bin is conjugated

and multiplied by the value of the second. The matrix stores the complex phase resulting from those multiplications.

A complex multiplication for L points results in $6L$ flops (4 real multiplications and 2 real additions). This is done for Q microphone pairs, yielding in a cost of $6QL$ flops. The computation of the phase of the complex element involves $5L$ operations. Thus, this requires $5QL$ additional flops.

4. Q inverse L-FFT are then carried out with the rows of matrix **GCC**. After this processing, the matrix **GCC** stores the temporal values (time delays). This requires $5QL\log_2 L$ operations.
5. A tridimensional **SRP** matrix is computed. Its dimensions are given by the number of points of the grid $G(\nu)$. Each element of this matrix stores the value $P'_n(\mathbf{x})$ of one spatial point $\mathbf{x}$ (see Eq. 4). Each element of the matrix is computed by accumulating Q GCC values, one per row of the matrix. The selected column in each row depends on the IMTDF (see Eq. 5), i.e, from point $\mathbf{x}$ to the pair of microphones associated to the row.

   The values of the inter-microphone time-delays are independent of the values of the audio samples. Therefore, as we will show in section 3, we have tested the possibility of reducing the cost of processing the audio frames by precomputing those values and storing them in a four-dimensional matrix. Each element `IMTDF[i, j, k, l]` contains the time-delay from the point of the grid G with coordinates $\{i, j, k\}$ to the lth pair of microphones.
6. Finally, the algorithm computes the position of the maximum SRP value, which represents the estimated sound source location.

Steps 5 and 6 require the evaluation of the following parameters for each point of the grid:

– S Euclidean distances, $\|\mathbf{x}_k\|$, requiring 3 multiplications, 9 additions and 1 square root.
– Q TDOAs, requiring 2 flops (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: $3Q$ flops.

In total, the cost of the SRP-PHAT is given by:

$$Cost = \left(\frac{S+S^2}{2}\right) 5L\log_2 L + \frac{11S^2-9S}{2}L + \nu\left(13S + \frac{5S(S-1)}{2}\right) \text{ flops,} \tag{7}$$

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 G.

3 Parallel Implementations

The SRP-PATH algorithm presents many opportunities of massive data parallelization. Its main steps described in the previous section must be performed sequentially as each of them depends on the results of the previous one. However each step consists of regular and independent operations over different elements of vectors and matrices and so they can be computed in parallel. In [11] we introduced a parallel implementation of the algorithm using CUDA. Specifically we described the different kernels used to approach each of the main steps of the method described of section 2.1. In the following we will briefly outline the OpenCL implementation whose kernels are mainly based on the CUDA kernels.

1. Kernel `kHamming` uses a 1D workspace of size L to apply the Hamming weighting to the samples in parallel. This weighting could be applied in parallel to each individual sample. However, we obtain better results if we increase the granularity of the parallelism and each of the L work-items weights one element of each of the S microphones. The access to the vector of samples is coalesced improving the performance of the kernel.
2. A parallel method can be applied to perform each of the S FFT of size L. For example, we can use the multithreaded implementation included in the FFTW library [17], the CUDA routines included in the cuFFT library [18], or the OpenCL routines included in clFFT library [19].
3. Kernel `kGCC` also uses a 1D workspace of size L to compute in parallel matrix **GCC**. Every element of that matrix could be computed in parallel. However, it is more efficient that each of the L work-items computes the Q elements of one of the columns of **GCC**. As we store the elements of the matrix in a row-major order the access to memory is also coalesced.
4. A parallel method can be applied to perform each of the Q inverse-FFT of size L, using for example the libraries cited in Step 2.
5. Kernel `kSRP` uses a 3D workspace of size $P_x \times P_y \times P_z$ to compute in parallel every element of matrix **SRP**, corresponding to one of the points of the grid. Every work-item has to access Q elements of matrix **GCC**, one of every row. However, the access to matrix **GCC** is not coalesced, as the column indexes depend on the IMTD to the grid point associated to the work-item.
6. In order to compute in parallel the maximum SRP value we have used an iterative tree reduction scheme as described in [20]. The kernel `kRedMax` chooses the best work-group size to fully exploit the local memory of the compute units. The position of the maximum SRP value is obtained in parallel using kernel `kPosMax`. Every of the ν launched work-items deals with one SRP value. If the work-item owns the maximum SRP value, returns its position.

The OpenMP parallel implementation of the method is quite straightforward, because the main steps of the sequential algorithm are based on loops that perform similar operations on different elements of the matrices and vectors involved. We have implemented each step of the sequential method so that we can parallelize it by using the `omp parallel for` pragma on its outermost loop. On step 1 we have parallelized the loop that iterates over the S microphones and on step 3 the loop that iterates over the L samples. We have combined the codes that perform steps 5 and 6, so that we compute the SRP values and its maximum at the same time without having to store the matrix. In this case we have parallelized the loop that iterates over the ν points of the grid. Every OpenMP thread computes a local maximum and its position by means of a `reduction` clause. Afterwards, the master thread computes the global maximum and its position. Finally, to carry out the FFTs of steps 2 and 4 of the method we have used the multithreaded routines included on the FFTW library. We have tested different scheduling strategies of the loops. The best results are obtained using the `auto` scheduling option and are quite similar to the ones obtained by using the `guided` option.

4 Experimental analysis

In order to carry out the tests, we select two high performance parallel architectures: a multicore CPU platform (named as **seb**) and a many-core GPU (named as **p100**):

– **seb** is a server with two Intel Xeon E5-2695 v4 CPUs at 2.10GHz. Each processor contains 18 cores. Therefore, the node features a total of 36 physical cores (72 logical cores if we activate hyper-threading). The platform also includes 132 GB of main memory. Up to 8,192 OpenCL work-items can be launched in parallel on this platform on each of its 72 compute units.
– **p100** is a NVIDIA GPU Tesla P100 accelerator implementing the Pascal micro-architecture. It includes 16 GB of global memory and 3,584 CUDA cores that can provide up to 9.3 TeraFLOPS using single-precision floating point arithmetics. Up to 1,024 OpenCL work-items can be launched in parallel on this platform on each of its 56 compute units.

We have performed all our experiments using microphone setups as in Figure 1 (b) containing from 6 to 96 microphones and using synthetic audio samples. The algorithms have been tested with sample buffers of size $L = 4096$ for each microphone. For a sample frequency $f_s = 44.1$ KHz, if we want to locate the source in real time, the processing time of the algorithm must be less than $t_p = 92.88$ ms.

4.1 Results on a multicore platform

We will first show some of the results obtained with the sequential version of the algorithm in one of the cores of **seb**. As we increase the grid resolution most of the time of the algorithm is devoted to compute the SRP matrix. Therefore, we have implemented several versions of this step:

– **pairs.** For each element of the matrix, corresponding to one point of the grid, this version computes the distance to each microphone of each pair.

- **priv.** For each element of the matrix, we compute and store its distance to each microphone. Then we can reuse those distances to compute the inter-microphone time delays to each pair of microphones.
- **imtd.** This version precomputes the distances from each point to each pair of microphones and stores these values in a four-dimensional matrix. The algorithm can then reuse these values on each iteration.
- **mtd.** In order to reduce the spatial cost of previous version, we only precompute and store the distance from each point to each microphone in a three-dimensional matrix.

Our experiments show that the best version of the algorithm depends on the platform and on the programming technology. In some cases it even depends on the number of microphones and grid resolution. Figure 2 shows that the best sequential version for all grid resolutions is the one that precomputes matrix **IMTD**. The **priv** and **mtd** versions obtain similar results and recomputing all pairs of distances produces worse results as we increase the resolution. The behavior is similar with more than 24 microphones, but as we increase the number of microphones we cannot store matrix **IMTD** and then, we use matrix **MTD**.

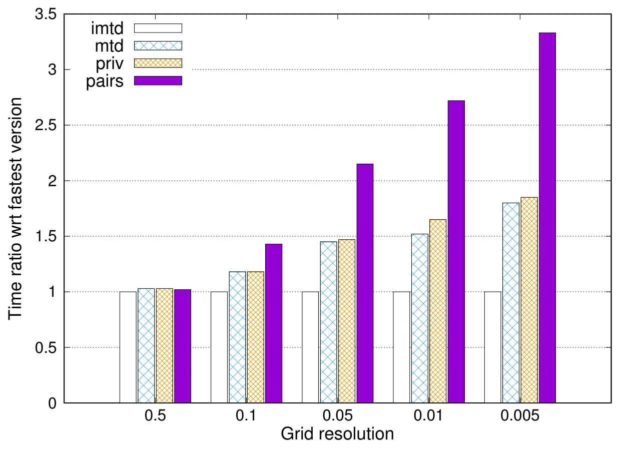

Fig. 2 Time ratio of the different sequential versions of the sequential algorithm with respect to the best version precomputing IMTD matrix. Results are shown for 24 microphones and using different grid resolutions.

Table 1 shows that the computational time of the best version of the sequential algorithm quickly increases with the number of microphones and grid resolution. This version of the algorithm can only perform the localization in real time with few microphones and small resolutions (marked in bold in Table 1).

For the OpenMP version, the best results are obtained exploiting the hyper-threading capability of the architecture and launching up to 72 threads in

Table 1 Computational time (in milliseconds) of the best sequential version.

Res.	Microphones								
	6	12	24	36	48	60	72	84	96
0.5	**3.4**	**6.6**	**25.9**	**56.8**	99.1	164.8	240.9	333.0	439.5
0.1	**3.3**	**8.9**	**35.7**	**77.0**	138.1	235.6	383.5	568.3	757.2
0.05	**3.5**	**15.2**	**65.5**	144.3	262.7	461.4	782.5	1,139.0	1,604.6
0.01	**46.0**	202.5	1,008.5	2,406.8	4,239.3	6,890.0	10,692.7	15,698.24	21,362.4
0.005	182.1	763.4	4,009.5	9,475.3	16,862.7	26,692.7	40,610.6	57,217.8	76,770.7

parallel. In this case the fastest results are obtained precomputing the **mtd** matrix. Figure 3 shows that, for 24 microphones, the algorithm only scales with more than 12 threads if we use high resolutions and provide enough computation to every thread. We can also see the effect of using hyper-threading with more than 36 threads.

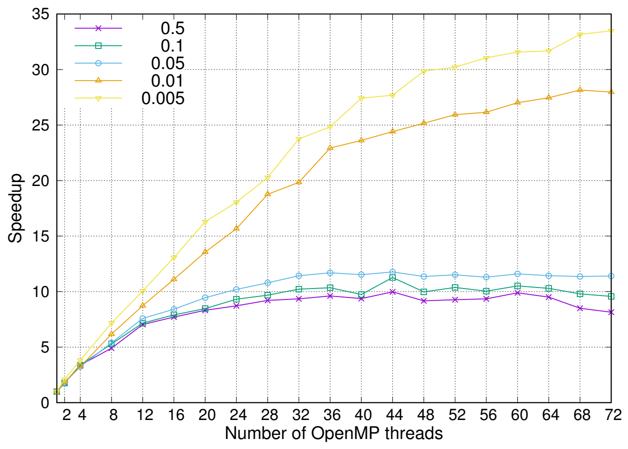

Fig. 3 Speedup of the OpenMP version of the algorithm using 24 microphones.

Figure 4 shows the number of microphones that can be used in real time to locate the source with different grid resolutions both in OpenCL and in OpenMP. The OpenCL version can use more than 20 microphones in real time even with very large resolutions, $r = 0.005$. Both parallel algorithms can handle more than 96 microphones in real time with small resolutions, $r > 0.1$. The OpenMP implementation is only better in the configurations composed by low resolutions and low and high number of microphones (from 6 to 12 and from 60 to 96).

Figure 5 allows us to compare the speedups of both parallel versions of the algorithm. These acceleration factors are computed with respect to the sequential version executed in one core. The OpenCL version clearly overcomes the OpenMP version. This behaviour is due to the fact that the OpenCL version leverages better the vector units of the cores than the sequential and OpenMP

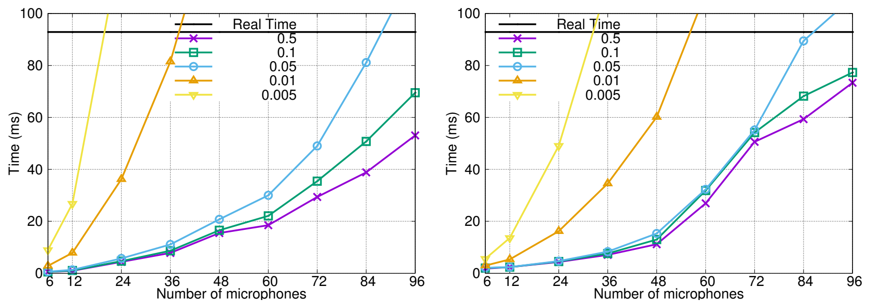

Fig. 4 Time in milliseconds of the OpenMP (left) and OpenCL (right) parallel versions of the algorithm with respect to the Real Time localization (horizontal line)

versions, specially during the most costly step of the algorithm, computing the SRP matrix. This allows the OpenCL version more than double the speedup of the OpenMP version rendering speedups higher than 72 when we increase the grid resolution. We can also see that with high resolutions the speedups of both versions reach a maximum with 48 microphones and they worsen when we increase the number of microphones. This behaviour is mainly due to the non-coalesced access to matrix GCC during the computation of matrix SRP. The size of matrix GCC increases quadratically with the number of microphones and the irregular access to its elements in every thread saturates the cache levels quickly increasing the number of L3 cache misses.

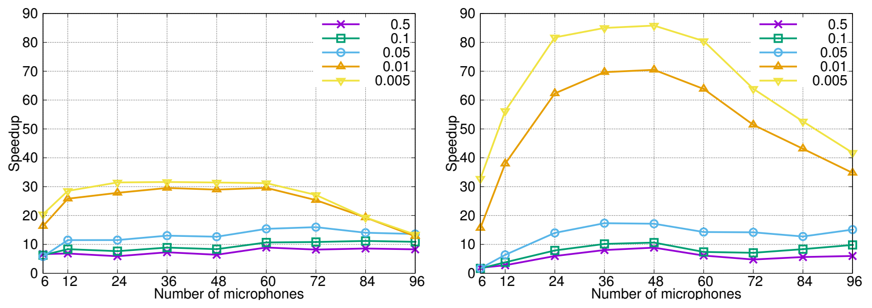

Fig. 5 Speedups of the OpenMP (left) and OpenCL (right) parallel algorithm.

4.2 Results on a many-core GPU

Regarding the Tesla P100 GPU platform, we have used CUDA and OpenCL to implement two parallel versions of the SRP-PHAT algorithm. We will compare the experimental results of both implementations using the **pairs** version of the kernels that compute the SRP matrix, as it is the only solution that allows us to deal with up to 96 microphones and very large grid resolutions, $r = 0.005$. Besides, precomputing the delays reduces only slightly the execution time.

Figure 6 shows that we can locate the source in real time using both parallel algorithms with a large number of microphones and very high grid resolutions. The CUDA implementation overcomes the OpenCL implementation (as is expected), but the gap between both versions is rather short, specially with high resolution grids, where the CUDA implementation is at most 35% faster than the OpenCL implementation.

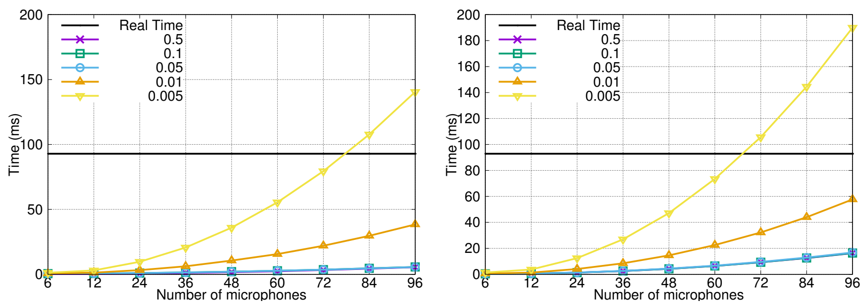

Fig. 6 Time in milliseconds of the CUDA (left) and OpenCL (right) versions of the algorithm with respect to the Real time localization (horizontal line).

4.3 Comparison of platforms and programming technologies

Figure 7 compares the results obtained on both platforms using different APIs varying the number of microphones and the grid resolution. It is selected 24 microphones and a grid resolution of 0.05 because these configurations achieve real time on almost most of the cases. We can first note that the P100 GPU is always faster than the CPU and the performance gap between both platforms grows as we increase the cost of the algorithm by using more microphones or larger resolutions. These results confirm that a state-of-the-art GPU platform is especially appropriate to deal with massive data parallel problems, even when compared with a high performance multicore CPU. We can dismiss the influence of the API as OpenCL allows us to implement the algorithm in both platforms and we can observe that the comparative behavior of both platforms is the same using this technology.

5 Conclusions

In this paper we show that current multi-core CPUs and many-core GPUs are ideal platforms to deal with problems that involve regular computations with

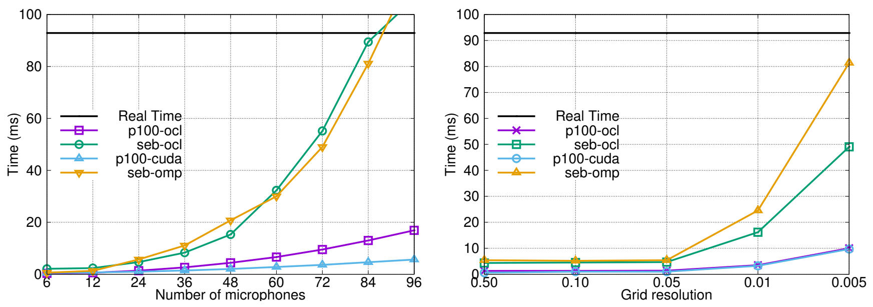

Fig. 7 Comparison of the parallel versions of the algorithm using both platforms and different programming technologies. Left hand side figure uses a grid resolution r=0.05 varying the number of microphones. Right hand side figure uses 24 microphones varying the grid resolution.

huge data matrices and vectors. A good example of this kind of application is the sound source localization using the SRP-PHAT algorithm. Using massive data parallelism on this kind of applications greatly improve their performance as the number of microphones and grid-search size increase.

We have compared the results of using some of the most extended parallel programming technologies for this kind of platforms, namely OpenMP, CUDA and OpenCL, on two state-of-the-art platforms: a 36 core CPU server and a Tesla P100 GPU. Our experimental results show that we can use in real-time a large number of microphones ($S > 70$) to perform the sound source localizations on high resolution grids ($r = 0.005$).

In this paper, we propose OpenCL as an efficient alternative for carrying out applications of this kind so that we can develop an implementation that can be portable and adaptable to different types of hardware. Results show that our implementation overcomes OpenMP in most of the CPU configurations as it better leverages its multiple cores. OpenMP only improves OpenCL in configurations that use low resolution grid-search sizes, which are not significant since they produce poor accuracy in the localization of the sound. On the other hand, the use of OpenCL instead of CUDA does not produce high differences in performance when both run applications of this kind on many-core GPUs. Both implementations obtain similar performances, especially when high spatial resolution parameters are considered.

The fact that OpenCL approaches and occasionally improves ad-hoc designed frameworks indicates that OpenCL is a serious candidate to carry out implementations that require portability and adaptability to different types of parallel architectures.

Acknowledgements This work has been supported by the postdoctoral fellowship from Generalitat Valenciana APOSTD/2016/069, the Spanish Government through TIN2014-53495-R, TIN2015-65277-R and BIA2016-76957-C3-1-R, and the Universidad Jaume I project UJI-B2016-20.

References

1. M. Brandstein and D. Ward, *Microphone arrays*, B. Verlag, Ed. Springer, 2001.
2. C. H. Knapp and G. C. Carter, “The generalized correlation method for estimation of time delay,” *Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-24, pp. 320–327, 1976.
3. M. Cobos, F. Antonacci, A. Alexandridis, A. Mouchtaris, and B. Lee, “A survey of sound source localization methods in wireless acoustic sensor networks,” *Wireless Communications and Mobile Computing*, vol. 2017, 2017, article ID 3956282.
4. J. H. DiBiase, “A high accuracy, low-latency technique for talker localization in reverberant environments using microphone arrays,” Ph.D. dissertation, Brown University, Providence, RI, May 2000.
5. C. H. Lee, “Location-aware speakers for the virtual reality environments,” *IEEE Access*, vol. 5, pp. 2636–2640, 2017.
6. “Implementing FPGA design with the OpenCL standard,” https://www.altera.com/en_US/pdfs/literature/wp/wp-01173-opencl.pdf, (accessed 2017 August 04).
7. L. Savioja, V. Välimäki, and J. O. Smith, “Audio signal processing using graphics processing units,” *J. Audio Eng. Soc*, vol. 59, no. 1-2, pp. 3–19, 2011.
8. J. A. Belloch, A. Gonzalez, F. J. Martínez-Zaldívar, and A. M. Vidal, “Real-time massive convolution for audio applications on GPU,” *Journal of Supercomputing*, vol. 58, no. 3, pp. 449–457, December 2011.
9. J. A. Belloch, A. Gonzalez, E. S. Quintana-Ortí, M. Ferrer, and V. Välimäki, “GPU-based dynamic wave field synthesis using fractional delay filters and room compensation,” *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 25, no. 2, pp. 435–447, Feb 2017.
10. V. Peruffo Minotto, C. Rosito Jung, L. Gonzaga da Silveira, and B. Lee, “GPU-based approaches for real-time sound source localization using the SRP-PHAT algorithm,” *International Journal of High Performance Computing Applications*, 2012.
11. J. A. Belloch, A. Gonzalez, A. M. Vidal, and M. Cobos, “On the performance of multi-gpu-based expert systems for acoustic localization involving massive microphone arrays,” *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5607–5620, 2015.
12. L. C. Seewald, L. Gonzaga, M. R. Veronez, V. P. Minotto, and C. R. Jung, “Combining srp-phat and two kinects for 3d sound source localization,” *Expert Syst. Appl.*, vol. 41, no. 16, pp. 0957–4174, 2014.
13. D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, “Multi-core platforms for beamforming and Wave Field Synthesis,” *IEEE Transactions on Multimedia*, vol. 3, no. 2, pp. 235–245, April 2011.
14. J. A. Belloch, M. J. Badia, F. D. Igual, E. Quintana-Ortí, and M. Cobos, “Evaluating sound source localization on multi and many-core platforms,” in *Proceedings of the 17th International Conference on Computational and Mathematical Methods in Science and Engineering*, vol. 1, Rota, Spain, July 2017, pp. 279–286.
15. M. Cobos, A. Marti, and J. J. Lopez, “A modified SRP-PHAT functional for robust real-time sound source localization with scalable spatial sampling,” *IEEE Signal Processing Letters*, vol. 18, no. 1, pp. 71–74, January 2011.
16. A. Marti, M. Cobos, and J. J. Lopez, “A steered response power iterative method for high-accuracy acoustic source location,” *Journal of the Acoustical Society of America*, vol. 134, no. 4, 2013.
17. M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on “Program Generation, Optimization, and Platform Adaptation”.
18. “NVIDIA Library CUFFT,” *online at: http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUFFT_Library.pdf.*
19. “OpenCL Fast Fourier Transforms,” http://clmathlibraries.github.io/clFFT, (accessed 2017 July).
20. M. Scarpino, *OpenCL in Action: How to Accelerate Graphics and Computation.* Manning, 2012.