ENDOWING UNDERWATER NETWORKS WITH CHANNEL AWARENESS: A DISCUSSION ON COMPUTATIONAL COMPLEXITY AND INFORMATION SIZE ISSUES

Matteo Lazzarin Department of Information Engineering, University of Padova
Paolo Casari via Gradenigo, 6/B — 35131 Padova, Italy
Michele Zorzi {casarip,lazzari2,zorzi}@dei.unipd.it

Abstract

In this paper, we consider the problem of transferring channel state information (CSI) to the nodes of an underwater network. As the CSI is derived by simulating the behavior of the underwater channel via ray tracing, we first discuss a parallel implementation of the ray tracing software obtained using the CUDA architecture, that helps compute the CSI faster. We then consider the problem of making the channel state information compact, and suitable for transmission over underwater acoustic channels. Our results indicate that a feedforward artificial neural network achieves both good accuracy and good compression of the CSI.

1 INTRODUCTION

The detailed simulation of underwater acoustic networks faces the complex task of reproducing the channel behavior in a way that is as accurate as possible. Usually, this task entails the computation of a solution to the equations that model the propagation of sound waves in the water. The latter operation typically leads to a high computational burden, compared to the empirical equations found, e.g., in [1, 2]. Such a complexity is not feasible in the simulation of networking protocols, which usually entails tens of thousands of transmissions and should not be subject to the additional complexity of reproducing the propagation of sound.

A solution that has been considered in several venues incorporates the free ray tracing tool Bellhop [3] in popular network simulators such as ns2-Miracle or ns3. The former has been interfaced to Bellhop via a software library named World Ocean Simulation System (WOSS) [4, 5], whereas the latter is available via an addition to the Underwater Acoustic Networks (UAN) module of ns3 [6]. By knowing the boundary conditions of the propagation environment (i.e., the shape of the bottom and the surface of the ocean, and such environmental parameters as the sound speed profile (SSP) and the type of bottom sediments), Bellhop can trace a number of sound rays (see also [7, Ch. 3]) from a transmission source to any given point in space where the destination is assumed to be present; in turn this allows the computation of the channel power attenuation, which concurs directly to the evaluation of link level-performance via, e.g., the Signal-to-Noise Ratio (SNR) that characterizes the transmission.

In typical scenarios, ray models offer a good tradeoff between accuracy and computational complexity; yet, in simulations involving mobile nodes and time-varying environmental parameters (hence a time-varying outcome of the ray tracing process) the time required to complete the simulation task may be very long, even on the order of several days. In order to shorten this time, in this paper we propose a parallel implementation of Bellhop's ray trajectory computation, designed using the CUDA architecture [8]. Our results show that the parallel implementation achieves a significant speed up, which substantially outperforms a serial computation pattern. As will be discussed in Section 2.1, the computation of the sound pressure involves numerical integration, and thus can not be easily parallelized using the CUDA architecture: therefore, its implementation has been kept serial, akin to the original implementation of Bellhop. Having a fast ray tracing tool available makes it feasible to compute some kind of channel state information (CSI), such as the statistics of the channel power attenuation. In this paper, we will discuss how to achieve this knowledge and how to represent it in a

compact way, suitable to be communicated to the nodes of an underwater network, thus making them aware of the channel conditions in their proximity.

The remainder of this paper is organized as follows. In Section 2 we present the ray tracer implementation in CUDA and compare its performance to that of Bellhop; in Section 3 we discuss how to make the CSI compact and suitable for transmission to the nodes of an underwater network; in Section 4 we discuss the accuracy of such channel information representation methods; in Section 5 we draw some concluding remarks.

2 A RAY TRACER FOR UNDERWATER ACOUSTICS USING CUDA

We start by introducing Bellhop [3], a well known ray tracing tool for simulating the propagation of sound under water using ray models. We refer the reader to [7, Ch. 3] and to the references therein for a detailed introduction to ray models. In the following, we will summarize a few key points of Bellhop. Given the parameters of the environment (SSP, bathymetry, surface wave profile, as well as the geoacoustic parameters of the bottom sediments) Bellhop computes the ray trajectories and the acoustic pressure (assuming a unit power source) at every point of a predefined uniform grid in the (range,depth) plane.

One of the central components of Bellhop is the *TraceRay* procedure, which numerically integrates the coupled first-order differential ray equations and the dynamic ray equations in order to obtain the trajectory and amplitude of each ray departing from the source. Define r and z as the ray coordinates in the "range" and "depth" dimensions, respectively. Therefore, a sequence of $(r(s), z(s))$ pairs provides a sampled version of the ray trajectory as a function of the parameter s, which is defined as the arclength along the ray. Define also $c(s)$ as the sound speed along the ray path, c_{nn} as the curvature of the sound speed in a direction normal to the ray path, $p(s)$ as the ray amplitude and $q(s)$ as the ray beamwidth. The ray equations can then be written as follows (the auxiliary variables ξ and ζ have been introduced in order to write the coupled ray equations using first-order derivatives) [7, Ch. 3]:

$$\begin{cases} \dfrac{dr}{ds} = c\,\xi(s) \\ \dfrac{dz}{ds} = c\,\zeta(s) \end{cases} \qquad \begin{cases} \dfrac{d\xi}{ds} = -\dfrac{1}{c^2}\dfrac{dc}{dr} \\ \dfrac{d\zeta}{ds} = -\dfrac{1}{c^2}\dfrac{dc}{dz} \end{cases} \qquad , \qquad \begin{cases} \dfrac{dq}{ds} = c(s)p(s) \\ \dfrac{dp}{ds} = -\dfrac{c_{nn}}{c(s)^2}q(s) \end{cases} \tag{1}$$

The solution to these equations (together with initial conditions on the initial ray coordinates and on the ray takeoff angle) provides the trajectory of the ray in the form of its range and depth coordinates r and z. Solving the dynamic ray equations yields the ray amplitude and beamwidth. As can be seen in (1), the knowledge of the speed of sound in the range-depth plane is required in order to solve the ray equations. A common approximation that is usually made in this regard is to assume that the sound speed does not vary with the distance from the source. In any event the SSP, i.e., the function describing the variation of the speed of sound with depth, is still required.

Once the trajectory of a ray has been computed, Bellhop proceeds with the calculation of the related sound field. To this end, the user must specify a function that describes the decay of the sound pressure associated to the ray in a direction normal to the ray trajectory. Among the several available options, we will consider a Gaussian beam shaping in the following.

By repeating the above procedure for several rays leaving the source with different departure angles, Bellhop approximates the sound pressure at any given point in space by taking all rays for which the point falls within the ray beamwidth, and by computing the pressure as a function of the amplitude of these rays. In the following, we consider the "coherent" sound pressure computation, which is performed as in [7, Eq. (3.41)].

2.1 RAY TRAJECTORY COMPUTATION USING CUDA

CUDA [8] is a parallel architecture that allows programmers to leverage on the computing power of modern Graphics Processing Units (GPUs) to carry out computations. The CUDA architecture

bases its floating-point algebra performance on a very high level of parallelization. Therefore, to take advantage from it, an algorithm must support a parallel implementation. The computation of the ray trajectories in a stratified water medium supports such an implementation very well. In fact, by rewriting the ray equations so that the range coordinate r appears as a function of the depth coordinate z we get [7, Eq. (3.71)]

$$\frac{d^2r}{dz^2} = \left[1 + \left(\frac{dr}{dz}\right)^2\right]\left[\frac{1}{c}\left(\frac{dr}{dz}\right)\frac{dc}{dz} - \frac{1}{c}\frac{dc}{dr}\right]. \tag{2}$$

As anticipated in the previous Section, we approximate c as being independent of r. Therefore we get

$$r(z) = r(z_0) + \int_{z_0}^{z} \frac{ac(z')}{\sqrt{1 - a^2c(z')^2}}\, dz' \tag{3}$$

where a is an arbitrary parameter generally set equal to $\cos\theta_0/c(z_0)$, where θ_0 is the ray takeoff angle.

The knowledge of the SSP is instrumental to solving (3). However, the SSP is typically unavailable in the form of a function. More commonly, the SSP is provided in a sampled form and must be interpolated within each pair of subsequent samples. When linear interpolation is employed, the SSP becomes a piece-wise linear function, and the water column can be viewed as a stack of c-linear layers [7, Section 3.6.3]. Call z_i, $i = 1, \ldots, N_c$, the values of the depth where a SSP sample is available, and define a layer as the portion of the watercolumn within any two such depths $[z_i, z_{i+1})$. Within a layer, Eq. (3) yields

$$r(z) = r(z_i) + \left.\frac{\sqrt{1 - a^2c^2}}{ag}\right|_{c(z_i)}^{c(z)}, \tag{4}$$

where $c(z) = c(z_i) + gz$ is a linear function, and g is the slope of the line. This equation can be re-arranged to yield the equation of a circumference of radius $R = 1/(ag)$ and range coordinate of the center $b = \sqrt{1 - a^2(c_0 + gz_i)^2}/(ag)$.

Once the equation of the arc of circumference followed by the ray within a layer is known, the computation of the trajectory inside the layer can be completely parallelized. The code we developed to perform this task acts in parallel both across rays and within one ray. In particular, for each ray, the code takes 32 points at equally spaced depths within every layer, and delegates the computation of the circumference points to 32 separate CUDA threads. With this configuration, every thread derives exactly one ray trajectory point per layer, in order to achieve the fastest computation speed. At a later stage, these points are assembled into the complete ray trajectory. Whenever a new trajectory point is computed, the thread also checks whether bottom or surface interactions occur; in this case, Snell's law is employed to compute the reflection angle and the new equation of the ray trajectory.

2.2 DISCUSSION ON SOUND PRESSURE COMPUTATION IN CUDA

Once the ray trajectories have been derived, the next steps are (i) to compute the amplitude and beamwidth of each ray by integrating the differential equations for p and q in (1) and (ii) to compute the sound pressure in the (r, z) plane, usually on a grid of equally spaced points.

We have tried several patterns to parallelize the pressure computation, but they always resulted in comparable or higher computation time than with the "serial" version of Bellhop. The main reasons for this outcome can be explained as follows. Step (i) requires numerical integration, which is a step-by-step operation, i.e., it serially considers all points $1, 2, \ldots, n, \ldots$ along the ray path and requires the knowledge of the result of the integration at point n in order to compute the integration result at point $n + 1$. This makes parallel implementations subject to many memory transactions, which in turn decrease the effectiveness of the parallelization. Step (ii) requires to compute the distance between every point of the grid and the trajectory of every ray, as this is required to compute the contribution of the sound pressure at every grid point. This operation is subject to a practical shortcoming. In principle, every check requires a memory access to retrieve the coordinates of the grid point. Since

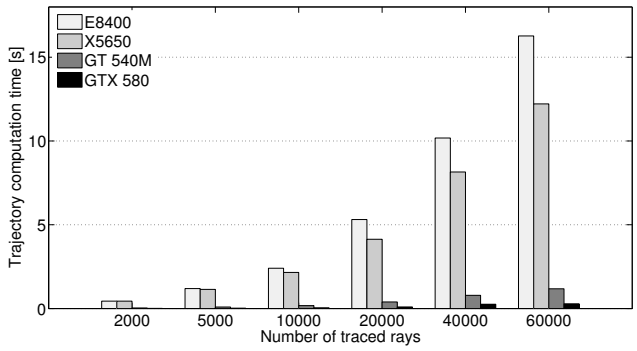


Figure 1: Amount of time required for computing the ray trajectories as a function of the number of rays, for CUDA-enabled (GT 540M, GTX 580) and non-CUDA-enabled (E8400, X5650) systems.

CPU / RAM	GPU / GRAM
Intel Core-2 Duo E8400 (3.0 GHz, 6 MB cache), 4 GB DDR2	——
Intel Xeon X5650 (2.66 GHz, 12 MB cache), 16 GB DDR3	——
Intel i7-2670QM (2.2 GHz, 6 MB cache) 8 GB DDR3	Nvidia GT 540M 2 GB
Intel i7-2600 (3.4 GHz, 8 MB cache) 16 GB DDR3	Nvidia GTX 580 3 GB

Figure 2: Systems where the serial and parallelized versions of the ray tracer have been tested.

both the trajectory and the width of each ray evolve as the ray propagates, it is very difficult to foresee a priori which grid points will be involved in the checks. In turn, this issue makes it difficult to combine memory accesses into a single operation, and decreases the speed of the pressure computation. Due to the lack of substantial performance improvements in the parallel version of the pressure computation algorithm, we preferred to keep the pressure computation serial.

2.3 RESULTS ON THE EXECUTION TIMES

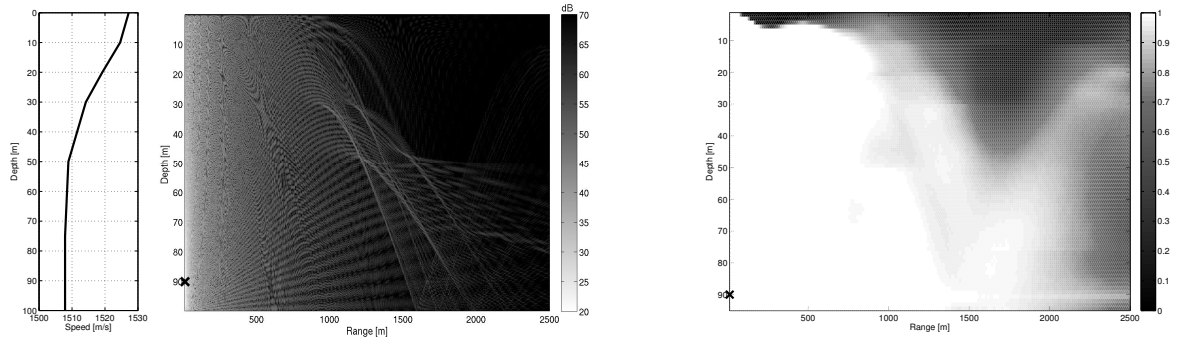
In this section, we present a comparison of the time required to compute the ray trajectories by the serial and the CUDA ray tracer implementations. We stress that the measured execution times include the time required to move data from the RAM of the host computer to the RAM of the CUDA device and back. The comparison is carried out for a shallow water scenario, which is more challenging from a computational point of view because of the large number of seabed and surface interactions incurred by the rays.

We performed several tests for different numbers of rays to be traced using the systems listed in Table 2. The trajectory computation times resulting from the tests are shown in Fig. 1. The results clearly demonstrate the advantages of the CUDA implementation. Notably, even the low-end GT 540M graphics card achieves a significant speedup of about 11 times with respect to a high-end Intel Xeon-based workstation. With the high-end GTX 580 graphics card, the speedup achieved is even higher, around 43 times.

3 TRANSFERRING CHANNEL AWARENESS TO THE NODES

A faster tool for the simulation of underwater sound propagation finds several applications in underwater communications. In this paper, we will consider the transfer of channel state information (CSI) to the nodes of an underwater network. This task encompasses the use of the ray tracer to compute performance metrics related to the channel behavior, as well a compact method to represent these metrics.

In the following, we define the CSI as the probability that the SNR, i.e., the ratio between the received acoustic signal power and the noise power, at any given point in the (r, z) plane exceeds a desired threshold θ. We denote this probability as $p(r, z) = P[\mathrm{SNR} > \theta]$. To compute this probability we proceed as follows. We assume that the network operates at a frequency $f = 20$ kHz for a shallow water



(a) (b) (c)

Figure 3: Example of channel realization and probability map derivation. (a) monthly average of the SSP for the month of May from [9], at $(10.5°\mathrm{E}, 42.5°\mathrm{N})$; (b) corresponding channel power attenuation derived using the ray tracer for $f = 20$ kHz (darker shades of grey correspond to a weaker signal); (c) $p(r, z)$ for $\theta = 10$ dB and a transmit power of 110 dB re μPa (darker shades of grey correspond to a lower probability). In (b) and (c), the position of the transmitter is marked by a black cross.

area of depth 100 m at $(10.5°\mathrm{E}, 42.5°\mathrm{N})$, in the Tyrrhenian sea. We approximate the sea surface and bottom as flat. We generate 5000 random realizations of the SSP. To do this, we assume operations in the month of May, and take the corresponding average SSP from the world ocean database (WOD) 2009 [9]. Akin to the discussion in Section 2.1, the SSP is provided as a set of samples, taken in this case at the standard depths of 0, 10, 20, 30, 50, 75 and 100 m. Each random realization of the SSP is generated by applying a displacement (uniformly drawn in the interval $[-4, +4]$ m/s) to each SSP sample. We perform a separate run of the ray tracer for each SSP sample, and derive the channel power attenuation at a set of points arranged in a grid in the (r, z) plane, where r spans the ranges from 12.5 m up to 2500 m in steps of 12.5 m, whereas z spans the depths from 1.25 m up to 95 m in steps of 1.25 m. By assuming that the source power level is 110 dB re μPa and using the empirical equations for the noise power spectral density in [2] with a shipping factor of 0.5 and a wind speed of 0 m/s, we compute the SNR for each pair (r, z) and for every realization of the SSP. Finally, for each (r, z) pair, we compute the probability $p(r, z)$ that the SNR exceeds θ. Fig. 3 shows an example of the derivation of $p(r, z)$. In particular, Fig. 3(a) shows the average May SSP taken from WOD 2009 [9], and Fig. 3(b) shows the corresponding channel power attenuation yielded by the ray tracer. Fig. 3(c) shows a pseudocolor plot of $p(r, z)$ for $\theta = 10$ dB.

We stress that such probability map can be employed for several networking purposes. For example, a routing protocol may employ the map to choose the relays located in those areas where $p(r, z)$ is highest. However, it is infeasible to assume that every node in an underwater network can compute its own $p(r, z)$. A more likely scenario is that a control center or gateway node with sufficient computational power computes $p(r, z)$ and transmits it to the nodes using some form of broadcast. It is therefore of interest to find a compact representation of the map, suitable to be transmitted to the nodes. This is the subject of the next subsection.

3.1 REPRESENTATION OF THE CHANNEL STATE INFORMATION

As introduced in Section 3, the map in Fig. 3(c) is obtained by finely sampling the (r, z) plane, using 200 points in the range dimension and 76 points in the depth dimension. Assuming one double-precision floating point number (8 Bytes) is employed to represent each probability value, the total size of the probability map is $S = 121.6$ kBytes. In the following we consider two different options to reduce the size of the map: (i) we subsample the map by a factor M in both the range and the depth dimensions: this reduces the map size to $L_s = S/M^2$; (ii) we train a Feedforward Artificial Neural Network (FANN) to approximate $p(r, z)$.

FANNs are a well-known way to approximate functions (possibly with multiple inputs and outputs)

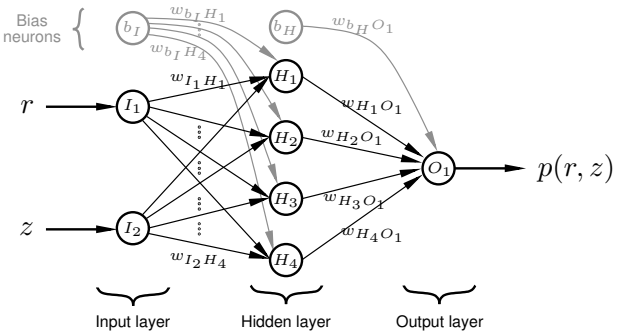


Figure 4: Example of FANN with one hidden layer for the approximation of $p(r, z)$. The input variables are the range r and the depth z of the point where $p(r, z)$ is to be computed.

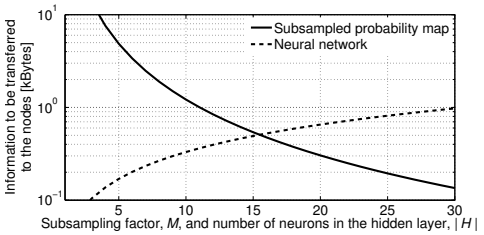


Figure 5: Amount of CSI (in kBytes) to be transferred to the network nodes in order to provide channel awareness.

that are too complex to be modeled using some closed-form equation. For a comprehensive survey of FANNs we refer the reader to [10]. For the present discussion, suffice it to say that FANNs can be described using nodes (the neurons) and edges (the connections between neurons) as in Fig. 4. Typically, a FANN always has at least two layers: the input layer, containing as many nodes as input variables, and one output layer (with as many nodes as output values). In addition, a FANN typically has one or more hidden layers, depending on the complexity of the function to be approximated. For the probability map approximation considered in this paper we will consider FANNs with one hidden layer. With the exception of input nodes, which only replicate their respective input on all output edges,[1] every node computes its output as a non-linear function of the weighted sum of its inputs plus a bias, that is usually modeled as an input from a separate neuron from the previous level (see Fig. 4). Namely, call j a node and $\mathcal{P}_j$ the set of all nodes that provide an input to j. The output y_j of j is computed as $y_j = f\left(\beta_j \sum_{i\in\mathcal{P}} w_{ij} x_i\right)$, where the x_is, $i \in \mathcal{P}_j$ are the inputs to node j. We note that the input from a bias neuron is always 1. The function f is called the activation function. In our implementation we take $f(\cdot) = \tanh(\cdot)$. The parameter β_j is employed to set the steepness of the activation function, and for every node it takes one among a finite set of values. In this paper we consider four values, hence the steepness can be represented using 2 bits, or 0.25 Bytes. Call $|\mathcal{I}|$ and $|\mathcal{H}|$ the cardinality of the input and hidden neuron sets, respectively (the cardinality of the output set is 1). The size of the information to be transferred to the nodes in Bytes is $L_n = 8\left((|\mathcal{I}|+1)|\mathcal{H}| + (|\mathcal{H}|+1)\right) + 0.25\,(|\mathcal{H}|+1) = 8\,(4|\mathcal{H}|+1) + 0.25\,(|\mathcal{H}|+1)$, as $|\mathcal{I}| = 2$. We remark that the $(|\mathcal{I}|+1)$ and $(|\mathcal{H}|+1)$ terms account for the bias neuron in the respective layers.

Fig. 5 shows the amount of information (in kBytes) to be transferred to the network nodes in order to make them channel-aware. The abscissa represents the subsampling factor in case a subsampled probability map is transmitted, or the number of neurons in the hidden FANN layer in case the weights of the FANN are transmitted. We observe that for $M = |\mathcal{H}| = 15$, we have $L_s \approx L_n \approx 500$ Bytes.

For reference, in Fig. 6 we show the original $p(r, z)$ map (a), and compare it to the approximate maps that result from subsampling by a factor $M = 8$ (b), and from training a FANN with $|\mathcal{H}| = 30$ hidden neurons (c). For these values the cases (b) and (c) achieve approximately the same map size of 1 kByte. We observe that the FANN approximates better the ridges and troughs of the map with respect to plain subsampling. In the following Section we will detail the comparison between the subsampling and FANN methods to approximate the SNR probability map.

4 PERFORMANCE OF THE REPRESENTATION METHODS

We now proceed by comparing the approximation to $p(r, z)$ for the subsampling and FANN methods described in Section 3.1. The comparison will be carried out in terms of the mean square error (MSE) of the approximation, the probability of false positive and the probability of false negative. The

[1] For this reason, the input layer is usually excluded from the layer count, i.e., the network in Fig. 4 is said to have two layers.

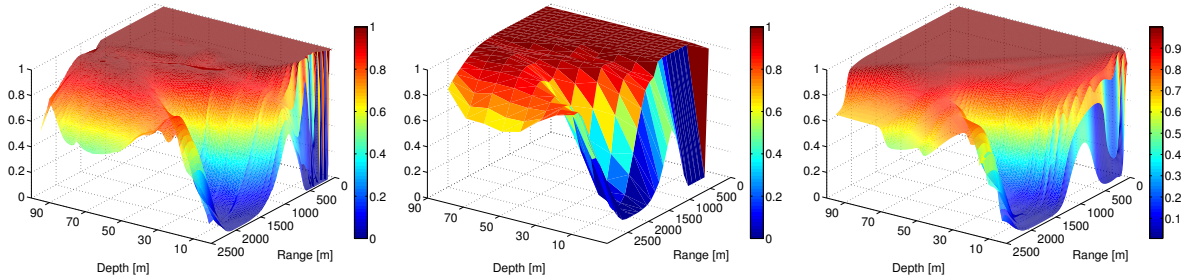


Figure 6: Original and approximated versions of $p(r, z)$ for $\theta = 10$ dB. (a) original map (the same as in Fig. 3(c), shown in 3D and seen from the upper right corner); (b) output obtained by subsampling the original map, $M = 8$; (c) output of the FANN with $|\mathcal{H}| = 30$.

false positive and false negative events are defined as follows. Assume that, in order to achieve reliable transmissions, we want $p(r, z)$ to be higher than some desirable value $\alpha = 0.75$. For each representation method, we say that a false positive occurs if the method predicts $p(r, z) \geq \alpha$, whereas the actual $p(r, z) < \alpha$. A false negative occurs in the opposite case.

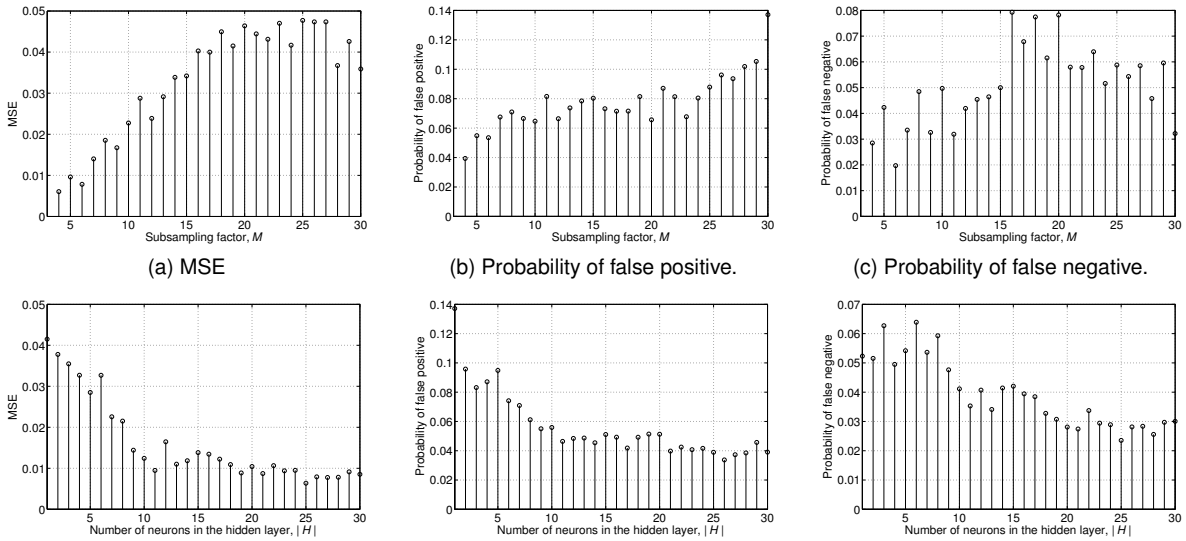
Fig. 7 shows (a) the MSE, (b) the probability of false positive and (c) the probability of false negative for the subsampling and FANN representation methods (top and bottom row, respectively). For the subsampling method, the metrics are shown as a function of the subsampling factor M (hence a more precise map is obtained for a lower value of M), whereas for the FANN method the metrics are shown as a function of the number of neurons in the hidden layer, $|\mathcal{H}|$ (hence the map is generally better for larger values of $|\mathcal{H}|$). The figures provide quantitative evidence that the FANN approximates the map better than what achieved by subsampling. In particular, let us consider $M = |\mathcal{H}| = 15$, which yields the same map size of about 500 Bytes for both methods as per Fig. 5. In this case, the MSE achieved by the FANN is about 0.150, against 0.340 achieved by subsampling, the probability of false positive is 0.045 against 0.070 and the probability of false negative is 0.042 against 0.050. We also note that a lower number of neurons in the hidden layer (e.g., $|\mathcal{H}| = 10$) still makes the FANN achieve a MSE, a probability of false positive and a probability of false negative that are almost the same as in the $|\mathcal{H}| = 15$ case, but with an even smaller map size, amounting to about 320 Bytes. We therefore conclude that FANNs are a suitable method for compressing the channel state information (in this case, the SNR probability map) in order to convey it through an underwater network.

While the results in this section are presented for a specific shallow-water scenario and for specific system parameters (e.g., the frequency f and the SNR threshold θ), the same conclusions are expected to hold in other scenarios. Further results (not shown here due to lack of space) for different values of f and θ, and for different depths of the transmitter support this statement.

5 CONCLUSIONS

In this paper, we have considered a specific definition of CSI for underwater acoustic communications (i.e., the probability that the SNR at a given depth and range from the source of a transmission is greater than a desired threshold). We then discussed a parallel implementation of a ray tracer based on the NVidia CUDA libraries, which helps simulate the underwater channel behavior faster than with other free tools currently available. In turn, this also helps compute the CSI faster given the parameters of the underwater scenario under consideration.

We finally argued that a fast tool for computing the CSI makes it possible to endow the nodes of an underwater network with channel awareness, provided that the CSI can be suitably compressed. To this end, we showed that neural networks make up a suitable tool to accurately approximate the CSI, while keeping the amount of information to be transferred to the underwater network limited.



(a) MSE (b) Probability of false positive. (c) Probability of false negative.

Figure 7: MSE (a), probability of false positive (b) and probability of false negative (c) for the subsampling method (top row), as a function of the subsampling factor M, and for the FANN (bottom row), as a function of the number of neurons in the hidden layer, $|\mathcal{H}|$.

ACKNOWLEDGEMENT

This work has been supported in part by the Italian Institute of Technology within the Project SEED framework (NAUTILUS project) and by the European Commission under the 7th Framework Programme (grant agreement no. 258359 – CLAM).
The authors are grateful to Prof. Michele Rossi for several useful discussions on FANNs.

REFERENCES

1. R. Urick. *Principles of Underwater Sound*. McGraw-Hill, New York, 1983.
2. M. Stojanovic. On the relationship between capacity and distance in an underwater acoustic communication channel. *ACM Mobile Comput. and Commun. Review*, 11(4): pp. 34–43, October 2007.
3. M. Porter *et al.* Bellhop code. `http://oalib.hlsresearch.com/Rays/index.html`.
4. F. Guerra, P. Casari, and M. Zorzi. World Ocean Simulation System (WOSS): a simulation tool for underwater networks with realistic propagation modeling. In *Proc. of ACM WUWNet 2009*, Berkeley, CA, November 2009.
5. N. Baldo, M. Miozzo, F. Guerra, M. Rossi, and M. Zorzi. MIRACLE: The Multi-Interface Cross-Layer Extension of ns2. *EURASIP Journal on Wireless Communications and Networking*, January 2010. `http://www.hindawi.com/journals/wcn/2010/761792/cta/`.
6. NS-3: UAN models. `http://www.nsnam.org/docs/release/3.10/doxygen/group__u_a_n.html`.
7. F. Jensen, W. Kuperman, M. Porter, and H. Schmidt. *Computational Ocean Acoustics*. Springer-Verlag, New York, 2 edition, 1984. 2nd printing 2000.
8. NVidia CUDA architecture. `http://developer.nvidia.com/what-cuda`.
9. World ocean atlas. `www.nodc.noaa.gov/OC5/WOA05/pr_woa05.html`.
10. R. D. Reed and R. J. Marks II. *Neural Smithing*. MIT Press, Cambridge, MA.