Edge Computing Architecture for Optimized Deep Sea AUV Communication

Dev Saxena
Department of Computer Science
William & Mary
Williamsburg, VA, USA
dsaxena@wm.edu

***Abstract*—Approximately seventy-one percent of the Earth is covered in water. Of that area, ninety-five percent of the ocean has never been explored or mapped. There are several engineering challenges that have prevented the exploration of the deep ocean through human or autonomous means. These challenges include but are not limited to high pressure, cold temperatures, little natural light, corrosion of materials, and communication. Ongoing research has been focused on trying to find optimal and low-cost solutions to effective communication between autonomous underwater vehicles (AUVs), and the surface or air. This paper introduces an architecture that utilizes an edge computing approach to establish a network of devices that could enable further exploration of the deep ocean. For this network, we use existing technologies that have been tested in the field or are nearing the end of their development and compare different approaches.**

***Index Terms*—AUV, ROV, tethered, edge device, mother ship.**

I. Introduction

Exploration of the deep ocean is crucial to study climate change, energy, ocean conservation, and human health. Further knowledge of what lies in the dark depths of the ocean seabed could help discover renewable energy sources, advance natural medicine research, and protect the ocean ecosystem [7].

Before introducing the architecture, there are several constraints that have shaped previous and current implementations of solutions to deep ocean exploration. Many of the underlying ones stem from physical and chemical phenomena, but there are also many other controllable factors such as cost and choice of communication technology. Although this paper does not focus on the physical design aspects of the devices used for exploration, these challenges can help minimize the wear on devices or the cost after each mission; hence why we account for them in our network.

A. Temperature and Pressure

In ocean depths past six thousand meters, the pressure of water reaches approximately 596 atm. This is problematic for most of the electronics that need to be mounted on the AUV such as cameras, lights, and computers. Since most are built to operate above the water, they operate in 1 atm of air. To combat this, the most practical solution is to enclose the technological devices in air-filled housing that will not collapse under such intense pressure.

The temperature at such depths in the ocean can reach to around four degrees Celsius. This poses a problem for dissimilar materials - materials that are difficult to join as a result of their chemical or physical compositions - because they may shrink or expand at various rates when the temperature falls [5].

B. Corrosion and Darkness

On the theme of what materials to use, saltwater is an electric conductor and so will accelerate the corrosion of metals that are immersed in it. In the development of AUVs and remotely operated vehicles (ROVs) are engineered with materials that are low on the Galvanic scale. A material like gold would be perfect but the cost of gold makes titanium a more suited material for construction.

Moving away from the materials aspect and onto the technology, we focus on the sensors and lights on the device that will help it navigate and collect data. There is very little natural light below 200 meters in the ocean and no sunlight whatsoever below 1000 meters. With underwater creatures, terrain, and other organisms to consider, AUVs need to produce ample light. These lights also need to be positioned in such a way that they don't reflect off of marine snow - reflective organic material in the deep ocean - and blind the cameras. Usually, AUVs and ROVs are armed with sonar as well as cameras to determine their position, listen for the seabed below, and detect sea creatures around them [5].

C. Communication

Lastly, we move onto communication methods as that is what the focus of this paper will be. Traditionally, some form of radar or EM waves would be used for autonomous vehicles but radio waves do not travel very far through water and are very susceptible to noise. This leaves solutions that are separated into categories that involve a tethered connection and untethered connections [5].

Tethered connections to vehicles involve cable(s) that are either coaxial, in twisted pairs, or are fiber-optic. Since this is a direct connection to a mothership on the surface, tethered solutions have a high data transfer rate but may have lower ranges. Outside of these considerations, a long cable extending into the ocean faces entaglement, breaking, fraying, and encounters with creatures.

Untethered connections involve communication from the vehicle to another device via radio waves, acoustic waves, or optical methods such as lasers or LEDs. Having already discussed the drawbacks of radio wave communication, acoustics are widely used because of their long range abilities. However, most acoustic solutions succumb to high noise and low bandwidth in data transfer. The relatively newer idea of optical communications, with some research in space exploration being pioneered by NASA, solves this issue by providing higher bandwidth although within closer distances [1]. Additional factors such as power consumption and latency are also higher for acoustic communication methods than they are for optical methods. Table I and Table II show the comparisons between the communication types [1].

Type	Technology	Data Rate	Range
Tether	Twisted Cables	10 Gb/s	100m
Tether	Optical Fiber	10 Tb/s	10,000m
Wireless	Acoustic	10 Kb/s	1000m
Wireless	Optical	100 Mb/s	100m
Wireless	Radio	10 Mb/s	1m

TABLE I
COMMUNICATION TECHNOLOGIES, THEIR TYPES, THEIR MAXIMUM DATA RATES, AND THEIR MAXIMUM RANGES

Tech	Use	Availability
Twisted Cables	control/nav/video	commercial
Optical Fiber	control/nav/video	commercial
Acoustic	control/nav	commercial
Optical	control/nav/video	research
Radio	control/nav	commercial

TABLE II
COMMUNICATION TECHNOLOGIES, THEIR USES, AND THEIR AVAILABILITY

D. *New Edge-Based Network*

Keeping all these constraints in mind for the development of our new network, we propose a new system that uses tethered devices and untethered devices and involves several devices communicating with each other. The goal is to process data efficiently nearer to the device in the deep ocean and minimize transfer up to the surface or shore. This system will use fiber optic cables, lasers, acoustic modems to communicate all the way from the deep ocean to a mother ship on the surface.

Our system attempts to model this by having 3 levels of devices in the ocean. On the surface, a mother ship stands to receive the gathered data and send any necessary, minimal instructions to an AUV. On the second level, approximately 6000 meters below sea level, we have an intermediary ROV that performs data processing and acts as a translator between the mother ship and the AUV nearer to the seabed. This intermediary ROV will be tethered to the mother ship. Finally, on the third and deepest level, the untethered AUV collects data from the deep ocean and sends it to the intermediary ROV through wireless methods. The full architecture will be discussed in later sections.

II. Related Work

Before introducing the architecture and findings of our system, it is worth looking at similar studies and models that have had a similar idea to ours.

A. *GFOE Deep Discover*

In 2022 the Global Foundation for Ocean Exploration had built one of the most capable ROVs to date called the Deep Discover (D2). The D2 boasted some great specifications with an ability to drop down to 6000 meters below sea level, used 27 LED lights, high definition cameras, a suction sampler, a manipulator arm, coral cutters, and a temperature probe. Its purpose was to collect videos, physical samples, and additional oceanic data for use by scientists.

In terms of the architecture, the D2 was tethered to a camera sled, Seirios. Seirios lit up the area under D2 and allowed D2 to safely crawl the seabed. Seirios was then tethered to a NOAA ship, Okeanos Explorer, with a six-mile-long cable made of steel. This model allowed pilots of the D2 to safely guide the ROV while it explored even the darkest parts of the deep ocean. The data transfer pipeline was through the tethered connections and allowed only the most crucial data to be transferred through the tethers without the D2 needing to surface. [5]

B. *Opensea IQ Edge*

A private company called Greensea Systems made waves in the computing aspect of deep ocean exploration by using untethered, commercially available, ROVs loaded with batteries, an acoustic modem, and revolutionary onboard software. They managed to put a parallelized NVIDIA edge platform on the ROV which allowed the ROV to process sonar and video data while also maintaining its navigation and communication capabilities.

Similar to the D2 from GFOE, Greensea's ROV only sent the most crucial information to human operators and did most of the data processing onboard. This method allowed the use of acoustic modems as the amount and frequency of data was drastically reduced. The company ran a successful test of their system in March 2023. [6]

III. Interpolation Methods

Now equipped with GPU compute capability, AUVs have the potential to process and interpolate bathymetry data faster. The following subsections describe the specific design choices that I made to implement the interpolation functions on the GPU and compare them against their CPU counterparts. For all of these methods, we assume the a cell structure of existing points, and then compute an interpolated point based on that cell. Doing this computation for multiple cells, the aim is to generate points inside a larger grid. This is represented visually in Fig. 1.

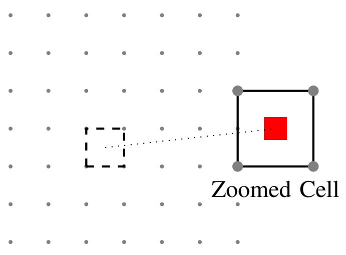

Fig. 1. Zoomed-in view of a single interpolated value inside a larger grid. The red square represents the interpolated value

A. Bilinear Interpolation

For this application, I tested three different interpolation methods commonly used for bathymetric data: Bilinear, Cubic Spline, and Ordinary Kriging. There are many other variants and forms of each method for which the code changes can be easily implemented on top of the base functionality I implemented. I briefly introduce each method below with the specifications used in this research and sources for further reading.

The bilinear interpolation scheme combines two linear interpolations, one along the longitude coordinates and one along the latitude coordinates, to compute an estimate in a 2-dimensional space. Assuming an individual cell in our data that has 4 pre-existing points and their elevation values given by Q_{ij}

$$\begin{gathered}(x_1, y_1), f(x_1, y_1) = Q_{11}\\(x_2, y_1), f(x_2, y_1) = Q_{21}\\(x_1, y_2), f(x_1, y_2) = Q_{12}\\(x_2, y_2), f(x_2, y_2) = Q_{22}\end{gathered}$$

Then, in order to estimate a new (x, y), we use the following formula:

$$\begin{aligned}f(x,y) = &\frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}Q_{11} + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}Q_{21}\\&+ \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}Q_{12} + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}Q_{22}\end{aligned}$$

This method will produce a weighted average of the 4 surrounding points. The closer that a point is to the target, the greater influence it will have on the interpolated result. Doing this repeatedly for several cells inside a larger "grid", we can construct any points we need inside any given grid.

B. Cubic Spline Interpolation (Catmull-Rom)

The Catmull-Rom cubic spline interpolation [2] works similarly but uses 16 neighboring grid points. We interpolate first in x on each of 4 rows, and then in y on each of 4 columns using the following standard cubic interpolation formula:

$$\begin{aligned}C(t) = &\frac{1}{2}(2P_1 + (-P_0 + P_2)t + (2P_0 - 5P_1 + 4P_2 - P_3)t^2\\&+ (-P_0 + 3P_1 - 3P_2 + P_3)t^3)\end{aligned}$$

With more input data, the cubic spline interpolation scheme is expected to perform with better accuracy but also needs to query a greater number of points, increasing memory access times.

C. Ordinary Kriging

Our last interpolation scheme is the most accurate and also very commonly used for bathymetric applications due to its sophisticated weighting ability. Since this is likely the most foreign method to readers, I will go more in depth with its construction.

There are 2 main principles that govern Kriging algorithms:

1) Datum close in geological distance to the unknown should get larger weight
2) Data close together are redundant and should share their weight

To quantify these principles, we first use a variogram [8] to model the correlation in distance and elevation between 2 arbitrary points. Let x be the geographic latitude and longitude of the point and let y be the elevation at that point. Then, the variogram function is given by

$$\gamma(h) = C_0(1 - exp(-\frac{h}{a})$$

where C_0 represents the sill, a represents the range, and h is the separation distance. The rest of the kriging algorithm [3] can then be expressed as

$$\gamma(x_i, x_j)\Lambda = \gamma(x_o, x_i)$$

where Λ is the matrix of weights pertaining to each neighboring point. For multiple points, such as the grid that we need, the following matrix calculation can be solved using Gaussian elimination. Here is what that may look like for a grid consisting of 3 points (our implementation uses a 5x5 grid):

$$\begin{pmatrix}\sigma_{11} & \sigma_{12} & \sigma_{13}\\\sigma_{21} & \sigma_{22} & \sigma_{23}\\\sigma_{31} & \sigma_{32} & \sigma_{33}\end{pmatrix}\begin{pmatrix}\lambda_{11} & \lambda_{12} & \lambda_{13}\\\lambda_{21} & \lambda_{22} & \lambda_{23}\\\lambda_{31} & \lambda_{32} & \lambda_{33}\end{pmatrix} = \begin{pmatrix}\sigma_{10} & \sigma_{10} & \sigma_{10}\\\sigma_{20} & \sigma_{20} & \sigma_{20}\\\sigma_{30} & \sigma_{30} & \sigma_{30}\end{pmatrix}$$

Having solved for the weights, the estimated elevation is given by

$$\hat{z}(x_0) = \hat{\lambda_1}z(x_1) + \hat{\lambda_2}z(x_2) + ... + \hat{\lambda_n}z(x_n)$$

IV. Computational Design

With our mathematical methods laid out, we now move to the computation part of this project. The structural design comprising each point and grid is universal across the CPU and GPU implementations. Each of the interpolation methods then has a CPU implementation in C++ and a GPU implementation in C++/CUDA.

A main control file will read in an existing grid of data obtained. Then, the CPU and GPU functions are called and timed to measure the efficiency and accuracy of the results [4].

Some testing results will be discussed in section V. In a real-time environment, the interpolation will be performed solely on the GPU and only control functions will require the CPU. This design choice will minimize sequential processing while preserving as much memory as possible. Although there are many different ways to implement the interpolation methods on the GPU, the code in this project takes advantage of memory coalescing but assumes no shared memory exists on the device.

V. Experimental Findings

We now move onto the results of implementing these interpolation methods on the CPU and GPU, the testing design, and the evaluation metrics.

A. Device Specifications

Before we see the results and talk about the testing setup, it is important to note that performance metrics depend heavily on the GPU device's capabilities and will have different outcomes should the device change. The device used for these tests has the following specifications:

NVIDIA GeForce MX550	
Total Global Mem	2048 Mb
Cores	1024 CUDA Cores
Max Threads per Multi-processor	1024
Max Dim Size of thread block	(1024, 1024, 64)

B. Testing Design

For this study, the layout of the original data grid is very important in demonstrating how well the interpolation schemes perform on both the CPU and the GPU. The first case, also the most ”ideal”, considers a grid where each latitude is fully populated with elevation values for each corresponding longitude. The purpose of interpolating on such a grid, which we will reference in the future as Grid Type A, is to fill in entire missing latitudes between any two successive existing ones. This would be useful in real-time applications where data frequency can be artificially increased due to hardware or geographical limitations.

The second case considers an original grid with missing longitude values randomly distributed throughout. This case is to fill in gaps where any noise or hardware limitations may have prevented accurate data collection. We reference this test case as Grid Type B.

C. Grid Type A Results

Figures 2, 3, and 4 show the results of the tests performed on grid type A. For smaller batches of randomly generated interpolation points, the performance of CPU and GPU code seems to match pretty well across the three interpolation methods. However, as we increase the number of interpolation points to 100,000 and greater, the graphs start to diverge more clearly and the runtime for the CPU drastically increases for the cubic splines and ordinary Kriging cases.

One notable observation is the minimal difference in the performance of the bilinear interpolation method on the CPU and GPU across batch sizes. This is likely due to the schemes being implemented in almost the same way on both the CPU and the GPU. However, with a more capable device that has shared memory and by taking advantage of warp sizes, the GPU methods can attain much better performance not just on the bilinear method, but also the cubic splines and ordinary Kriging methods.

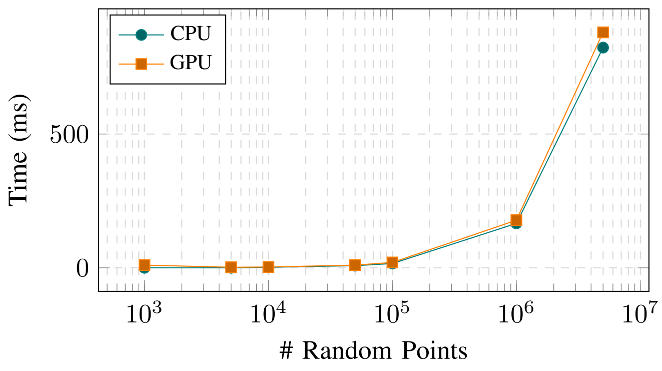

Fig. 2. Log-Log Plot of bilinear interpolation: CPU vs GPU.

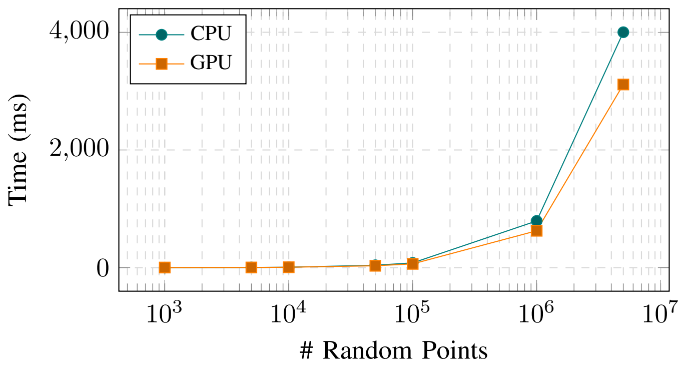

Fig. 3. Log-log plot of cubic interpolation: CPU vs GPU.

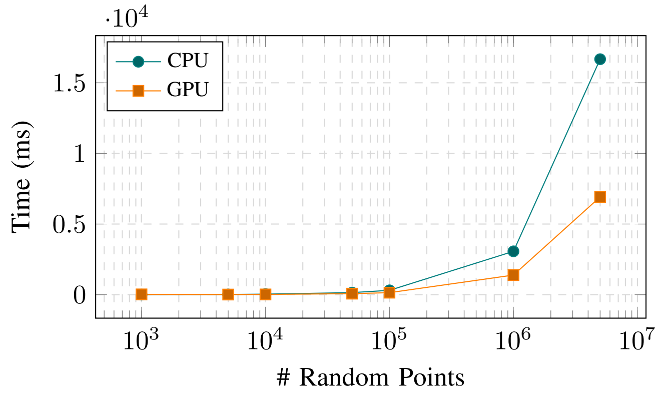

Fig. 4. Log-log plot of Kriging interpolation: CPU vs GPU.

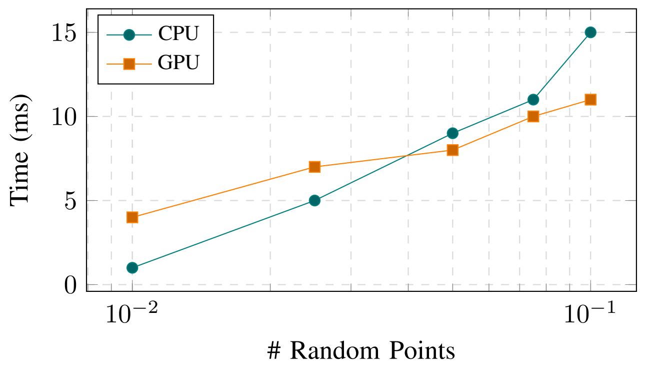

Fig. 5. Log-Log Plot of bilinear interpolation: CPU vs GPU.

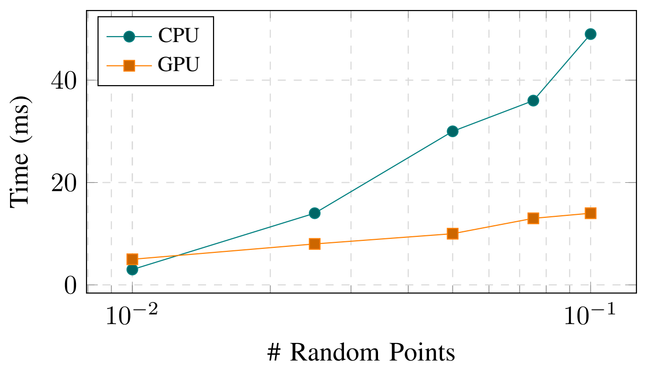

Fig. 6. Log-log plot of cubic interpolation: CPU vs GPU.

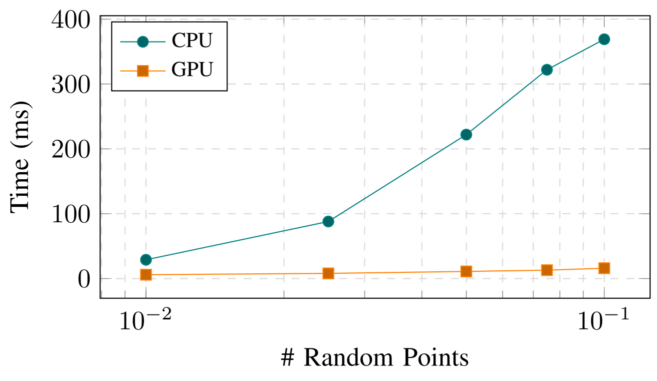

Fig. 7. Log-log plot of Kriging interpolation: CPU vs GPU.

D. Grid Type B Results

Firgures 5, 6, and 7 show results for the tests performed on grid type B, where some random values need to be filled in. Similar to the grid A cases, the GPU code's ability to outperform the CPU code increases with the batch size of points needing to be interpolated.

One main difference between these results for grid B and the earlier results for grid B is the better GPU interpolation performance even for smaller batch sizes. In fact, for the ordinary kriging tests, the GPU performance started out below the CPU performance even for a small number of interpolation points.

References

[1] Alberto Quattrini Li, C. J. Carver, Q. Shao, X. Zhou, and Srihari Nelakuditi, "Communication for Underwater Robots: Recent Trends," vol. 4, no. 2, pp. 13–22, Jun. 2023, doi: https://doi.org/10.1007/s43154-023-00100-4.

[2] Christopher Twigg, "Catmull-Rom splines", Mar. 2003, https://www.cs.cmu.edu/ fp/courses/graphics/asst5/catmullRom.pdf

[3] Cressie N (1988). Spatial prediction and ordinary kriging. (International Association for Mathematical Geology) Mathematical Geology, Vol. 20.

[4] Devsaxena (n.d.). GitHub - devsaxena974/AUV-Real-Time-Interpolation. Retrieved from https://github.com/devsaxena974/AUV-Real-Time-Interpolation

[5] M. Ryan, K. McLetchie, and P. Hoffman, "OYLA Articles: Ocean Exploration Technology: How Robots Are Uncovering the Mysteries of the Deep," oceanexplorer.noaa.gov, Aug. 2022. https://oceanexplorer.noaa.gov/explainers/technology.html

[6] "OPENSEA Edge Delivers Untethered Autonomous Operation to Commercially Available ROVs - Greensea IQ," Greensea IQ, Nov. 18, 2024. https://greenseaiq.com/news/opensea-edge-delivers-untethered-autonomous-operation-to-commercially-available-rovs/ (accessed Feb. 19, 2025).

[7] Schmidt Ocean Institute, "Why do we explore the deep Ocean?," 2021. [Online]. Available: https://oceanexplorer.noaa.gov/edu/materials/why-do-we-explore-fact-sheet.pdf

[8] Understanding a semivariogram: The range, sill, and nugget—ArcGIS Pro — Documentation (n.d.). Retrieved from https://pro.arcgis.com/en/pro-app/latest/help/analysis/geostatistical-analyst/understanding-a-semivariogram-the-range-sill-and-nugget.htm