High-Performance Programming and Execution of a Coral Biodiversity Mapping Algorithm Using Chapel

Scott Bachman
bachman@ucar.edu
National Center for Atmospheric Research
Boulder, Colorado, USA

Rebecca Green
rgreen@coral.org
The Coral Reef Alliance
San Francisco, California, USA

Anna Bakker
annabakker@earth.miami.edu
University of Miami
Coral Gables, Florida, USA

Helen Fox
hfox@coral.org
The Coral Reef Alliance
San Francisco, California, USA

Sam Purkis*
spurkis@earth.miami.edu
University of Miami
Coral Gables, Florida, USA

Ben Harshbarger
ben.james.harshbarger@hpe.com
Hewlett Packard Enterprise
Spring, Texas, USA

ABSTRACT

This paper will demonstrate how the parallelism and expressiveness of the Chapel programming language are used to achieve an enormous improvement in computational speed for a problem related to coral reef conservation. Chapel's concise syntax and versatile data structures enable this problem to be solved in under 300 lines of code, while reducing the time to solution from days down to the order of seconds. This improvement is so substantial that it represents a paradigm shift in the way biodiversity can be measured at scale, providing a wealth of novel information for marine ecosystem managers and opening up brand new avenues for scientific inquiry. This paper will review the solution strategy and data structures in Chapel that allowed these improvements to be realized, and will preview future extensions of this work that have been made possible by this drastic speedup.

CCS CONCEPTS

• **Applied computing** → **Environmental sciences**; • **Computing methodologies** → *Parallel programming languages.*

KEYWORDS

Chapel, coral reef, habitat heterogeneity, Rao's Q, ecosystem

1 INTRODUCTION

Coral reef ecosystems are in rapid decline worldwide, and their biodiversity loss is negatively impacting their ecological function, sustainability, and provision of ecosystem services. Efforts are increasingly being made to protect the remaining reefs through the creation of marine protected areas (MPAs), which are often placed to maximize species richness in a given area. Developing effective strategies for locating and managing MPAs remains challenging, however, due to the tendency for coral reefs to be both sparse and widely distributed, and because it is often very difficult and expensive to measure species diversity *in situ*. Such measurements are also subject to nuanced technical questions pertaining to sampling design, what constitutes a sufficient number of samples, and sometimes ambiguous definitions about how to define a species community [3].

These issues motivate the search for ways to assess species diversity using remote sensing. Remote sensing is the standout technology for mapping the Earth at scale and is increasingly being leveraged for assessing biodiversity of tropical coral reefs [5, 21, 32], taking advantage of its relatively cheap and repeatable sampling, essentially global spatial coverage, and extremely high resolution [e.g. 9, 20]. Remotely-sensed images are used to assess environmental heterogeneity as a proxy for biological diversity under the "habitat heterogeneity hypothesis" [15, 19], which postulates that areas of high environmental heterogeneity (i.e. habitat type) contain a larger number of ecological niches for species to fill [6, 18, 27]. Though there are inherent uncertainties associated with extrapolating biodiversity from habitat type, and habitat type from visible-spectrum imagery [e.g. 10], this overall approach has been successfully used to assess biodiversity across multiple reef sites [4, 11, 13, 14, 33].

Several metrics exist that attempt to quantify biodiversity [23]. A metric known as beta diversity is often employed because it measures the variability in species composition and population within a given area. However, it is not possible to quantify the different species in a particular area without taking *in situ* measurements, making beta diversity unsuitable for remote sensing measurements. An alternative approach is to measuring the variability in habitat types rather than species composition, resulting in a metric known as *habitat heterogeneity*. Larger values indicate frequent transitions between multiple habitats [e.g. 8], a condition that is well-correlated with species richness [e.g. 31]. Hence, measuring habitat heterogeneity using habitat data fits naturally with the hypothesis that habitat diversity is correlated with species diversity [e.g. 8, 17], and motivates using this approach for assessing potential MPA sites. Fortunately, coral reefs are very well-suited for measuring habitat heterogeneity using habitat maps, since they typically feature clear and shallow waters permitting high light penetration and differentiation of habitat types [7].

The Chapel program described in this paper calculates habitat heterogeneity using habitat maps derived from the Allen Coral Atlas (hereafter ACA), a global map and monitoring tool for the world's shallow coral reefs at < 5 meter pixel resolution. The motivation for this program stems from multiple ongoing studies that had previously calculated habitat heterogeneity using a serial MATLAB program. The previous program was encumbered by the extremely high resolution of the Atlas, which created a massive computational challenge that limited these calculations to areas of $O(10\text{ km})$. Because the habitat heterogeneity algorithm is highly parallelizable, Chapel offered an alternative solution for exploiting the potential parallelism and speeding up these calculations drastically.

The layout of this paper is as follows. Section 2 provides an overview of the mathematics underpinning the habitat heterogeneity calculation, which is a variant of the well-known Rao's Quadratic Entropy technique [2, 24, 28, hereafter Rao's Q]. Section 3 will discuss the Chapel program and solution strategy, including the approaches for preprocessing the input images and limiting the memory burden through distributed I/O. Section 4

*Also with Khaled bin Sultan Living Oceans Foundation, Annapolis, Maryland, USA.

will show results and performance metrics. Concluding remarks appear in Section 5.

2 OVERVIEW OF THE RAO'S Q FORMULA

The program discussed in this paper, hereafter referred to as "RapidQ", is a highly parallelized solver for calculating Rao's Q for each pixel of an input image. Its primary application thus far has been for calculating habitat diversity for shallow aquatic environments, though its use could readily be extended to other ecological settings as well.

Rao's Q is defined as the expected dissimilarity between two samples selected randomly with replacement, and is expressed mathematically as

$$Q = \sum_i \sum_j D_{ij} P_i P_j. \tag{1}$$

The indices i and j represent the different sample types in the image, such that the summations are taken over all possible types. D_{ij} is the dissimilarity between samples i and j, and P_i is the probability that sample i is drawn. The dissimilarity coefficients can be based on a variety of measures depending on whether Rao's Q is being applied to species, habitats, or otherwise [e.g. 16, 30]. Rao's Q is often favored because of its dependence on both the degree of dissimilarity between samples (D_{ij}) and their relative abundances (P), making it more well-aligned with a modern notion of biological diversity than other measures [26].

3 CHAPEL IMPLEMENTATION

3.1 Preprocessing the inputs

The recent release of global coral reef maps with sub-decameter spatial resolution [1, 22, 29] allows the habitat heterogeneity hypothesis to be tested at scale. One such map is the ACA, which contains both benthic and geomorphic map layers at 5 m pixel resolution using PlanetScope imagery [1]. The ACA largely automates its map productions by first applying a random forest classifier to identify the benthos and geomorphology, and then refines those classifications with object-based analysis. The ACA benthic habitat composition maps are mapped to approximately 10 m depth and use six classes, while the geomorphic composition maps are mapped to 15 m depth and use 12 classes [12]. All ACA map products are available to download on their data portal.

Some preprocessing must be done to extract the ACA maps in a usable format for most geographic information system (GIS) software. On the ACA portal, the user can specify an area of interest and request the underlying benthic and geomorphic maps to be sent to their email in a GeoJSON format. These files can be easily read into QGIS, or converted to a more widely used shapefile format. Once the desired area is loaded as a vector feature, it is then converted from a geographic to a projected coordinate system. A raster image is then created with pixels at 5 m resolution (mirroring the resolution of the original data), with specific values assigned to each habitat classification. Land polygons as used in ACA processing are downloaded from OpenStreetMap, converted to a raster at the same resolution as the habitat maps, and then merged. Two resulting rasters for benthic and geomorphic maps are created, with specified values for land pixels, valid mapped ACA habitat pixels, and unmapped ocean pixels (hereafter referred to as deep water pixels).

The habitats in the map are expressed as integer values in the set $\{0, 1, \ldots n, s\}$, where 0 represents land, $\{1, \ldots, n\}$ are valid aquatic habitats with nonzero pairwise dissimilarities, and s represents deep water. A $(n+1) \times (n+1)$ symmetric matrix stores the dissimilarity coefficients between respective habitats, which is saved to a text file that is read in during the initialization phase of RapidQ. The first line of this text file contains the integer $(n+1)$, which Chapel uses to declare the array size for storing the dissimilarity matrix (Listing 1).

The rasterized habitat map is then passed through a final pre-processing phase before it is ready for use in RapidQ. A Python-based Jupyter Notebook is used to load the map as a NumPy array using the Geospatial Data Abstraction Library (GDAL) package, whereupon the latitudinal index is reversed and all s values are replaced by $(n+1)$ to make the dissimilarity table smaller. Finally, the array elements are converted to have **uint**(8) type, and the array is saved to file in binary format. A companion text file is saved to store the array size for RapidQ to read in.

```
proc ReadArray(filename: string) throws {

    var infile = open(filename, iomode.r);
    var reader = infile.reader();

    // Read the number of rows and columns
    var m = reader.read(int);

    // Declare an array of the specified dimensions.
    var X: [1..m, 1..m] real(32);

    // Read the array
    reader.read(X);
}
```

Listing 1: Reading of dissimilarity matrix

3.2 The RapidQ algorithm

At its core, RapidQ is a highly optimized and parallelized algorithm for calculating and summing over a histogram *at every point in an image*. At any given point $\mathbf{x}$, the histogram records the number of times that each habitat type appears within a circular "window" of diameter L centered at $\mathbf{x}$. Dividing the histogram by the number of valid habitat pixels in the window yields a vector of probabilities, $\mathbf{P}$. Taking the outer product of $\mathbf{P}$ with itself yields the probability matrix P_iP_j. Rao's Q is calculated by double contracting P_iP_j with the dissimilarity matrix D_{ij}, which is read from file and has a copy stored on each Chapel locale.

When executed from the command line, RapidQ takes as input the name of the image file to be processed, the size of the window (in hectares), and the resolution of the image (in meters). The algorithm is schematized in Figure 1.

3.3 Masking with sparse domains

One of the key optimizations in RapidQ is that the window is convolved over the image in an order that minimizes the number of total computations. For example, consider a computational sequence in which habitat heterogeneity is calculated at location $\mathbf{x} = (i, j)$ using a window with $L = 100$ (see Figure 1). Once that sequence is complete, the window is moved one pixel to the right to $\mathbf{x} = (i+1, j)$. The location of the new window largely overlaps that of the previous window, except for two thin slivers of points on the rightmost edge of the new window and the leftmost edge of the previous window, respectively (Figure 2a). Therefore, rather than calculating $\mathbf{P}$ for all 7,854 points in the new window, it is considerably more efficient to save $\mathbf{P}$ from the overlapping region, adding the contribution from the 100 points

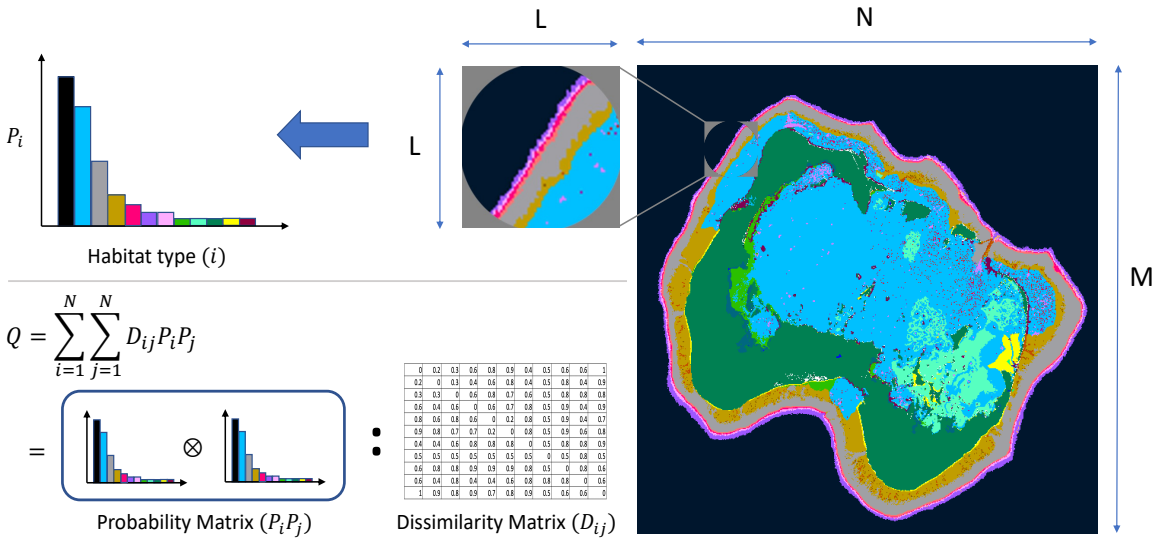

Figure 1: Schematic showing the procedure for calculating Q at a single point. Habitat map courtesy of the Living Oceans Foundation [22].

on the right edge of the new window, and discarding the contribution from the left edge of the previous window. Applying this technique over the full image, the computational complexity of RapidQ is reduced from $O(MNL^2)$ to $O(MNL)$, which is a substantial improvement when using larger windows.

```
proc create_masks(radius: real, dx: real, L: int) {

  const r = L/2;
  const Dom : domain(2, int) = {-r..r, -r..r};
  var Dom_center : sparse subdomain(Dom);
  var center_mask : [Dom_center] bool;
  // Set Implicitly Replicated Value (IRV) to "true"
  center_mask.IRV = true;

  var dist : real;

  // Define center mask.
  for (i,j) in Dom do {
    dist = dx * sqrt(i**2 + j**2);
    if (dist < radius) {
      Dom_center += (i,j);
    }
  }
```

Listing 2: Creation of center mask

RapidQ follows this strategy by employing **sparse** domains to assign the locations of points in the left, center, and right subregions ("masks") of the window. Listing 2 shows how the center mask, *Dom_center*, is defined inside the function *create_masks* using a Euclidean distance calculation. Here the argument *radius* represents the radius of the window in physical dimensions (equal to $r = L/2$ multiplied by the pixel resolution in meters, *dx*). Any point whose physical distance from the center pixel is less than *radius* is added to *Dom_center*, whose corresponding **sparse** boolean array has a default value of *true*. The **sparse** domain for the left edge, *Dom_left*, is then populated by considering all points in *Dom_center* and selecting those points (i, j) whose neighbor to the left, $(i - 1, j)$, is not in *Dom_center* (Listing 3). The selected point is then removed from *Dom_center* to ensure there is no double-counting of points. An analogous procedure is used to populate *Dom_right* and remove its points from the center mask, so the final result is three non-overlapping masks that together form the circular window. A local copy of each mask is created on each Chapel locale to eliminate any accesses to remote memory later on.

```
var Dom_left : sparse subdomain(Dom);
var left_mask : [Dom_left] bool;
left_mask.IRV = true;

// Define left mask
for (i,j) in Dom do {
  if ( Dom_center.contains(i,j) &&
     (i == -r || !Dom_center.contains(i-1,j)) ) {
    Dom_left += (i,j);
  }
}

// Remove overlapping points from center mask
for (i,j) in Dom_left do {
  if Dom_center.contains((i,j)) {
    Dom_center -= (i,j);
  }
}
```

Listing 3: Creation of left mask

3.4 Distributed computations and I/O

The RapidQ algorithm is embarrassingly parallel since each row of the image can be processed independently from the others. The program exploits this parallelism by using Chapel's **Block** distributed array type to partition the input image into smaller pieces, each of which is assigned to its own Chapel locale (Figure 2b). To ensure that the window is contained entirely within the domain of the image, the **expand** method is used to create a halo of width r around the edge of the image where Q is not calculated. The interior of the image, *Inner*, is where the window is convolved and is contained within the thick dashed red line in Figure 2b. The distribution of the **Block** domain, B, is defined by partitioning *Inner* among the locales (thin dashed red lines).

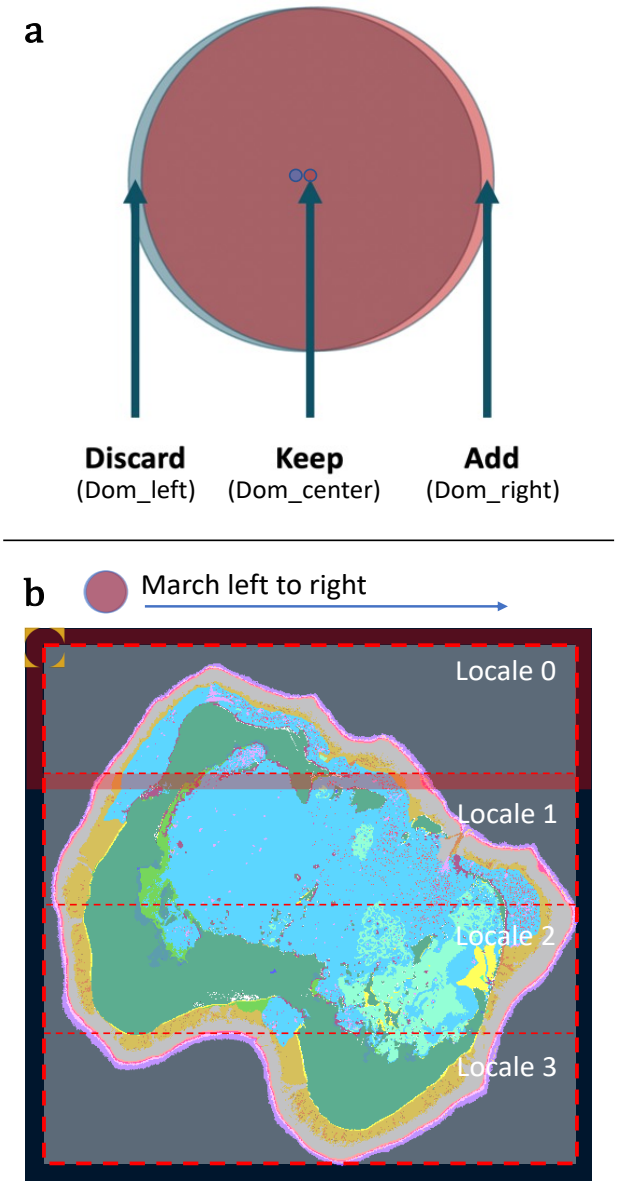

Figure 2: a) Convolution strategy using sparse domains. b) Block distribution of input image among locales, including halo region (red shading, shown for Locale 0 only).

The array *OutputArray* is distributed according to *B*, and stores the *Q* calculations on each locale.

The computation strategy shown in Figure 2a dictates that the window is marched from left to right across each row of the image. The only locations where all three masks in the window need to be calculated (and therefore are the most expensive to compute) are along the left edge of each **Block** subdomain. RapidQ employs a subtle optimization strategy to minimize the number of left edge points by reshaping the **Block** distribution so that the subdomains are rectangular strips across the full width of the image, rather than the default "as square as possible" configuration (Listing 4).

```
const Inner = ImageSpace.expand(-r);
// Split the indices up only along the first dimension
const myTargetLocales = reshape(Locales,
                        {1..Locales.size, 1..1});
const B = Inner dmapped Block(Inner,
          targetLocales=myTargetLocales);
var OutputArray : [B] real(32);
```

Listing 4: Distributed output array, neglecting halo region

RapidQ minimizes the memory burden on each locale by loading only the part of the image that is needed to calculate its own portion of *B*. Each locale also needs to load a halo of width *r* around its portion in order calculate *Q* up to the edges of *Inner*, which is depicted for Locale 0 by the red shaded region in Figure 2b. This is achieved by using the **expand** method to define a domain on each locale, *locB_plus*, that contains all the necessary points. The **first** and **last** methods define opposing corners of this domain, and are used to calculate the offset and bounds for reading the input image from binary (Listing 5).

```
coforall loc in Locales do on loc {

  var locB = B.localSubdomain();
  var locB_plus = locB.expand(r);
  var locImage : [locB_plus] int(8);

  var f = open(filename, iomode.r);
  var start = locB_plus.first[0]*locB_plus.shape[1]
              + locB_plus.first[1];
  var g = f.reader(kind=ionative, region=start..);

  // Read local portion of input image
  for (i,j) in locB_plus {
    var tmp : int(8);
    g.readBinary(tmp);
    locImage[i,j] = tmp;
  }

  r.close();
  ...
```

Listing 5: Distributed read

Each row of the image can be calculated independently from the others, but each pixel within that row depends on the preceding computation at the pixel to its left. Therefore, after the image is partitioned onto each locale, Chapel will attempt to process the rows in parallel using as many tasks as it thinks is appropriate, which is generally the number of available cores. Each task will begin the calculation of *Q* at the left edge of the row and march rightward.

For the habitat diversity problem being considered here, there are a relatively small number of habitat types ($d\_size = O(1)$) relative to the number of pixels in the window (generally thousands to hundreds of thousands). Therefore, an efficient solution strategy is to simply count the instances of each habitat type within the window and save them in a histogram (Figure 1). To align with the strategy in Figure 2a, a separate histogram must be created for each of the three sparse domains comprising the window. These are labeled *H_left*, *H_center*, and *H_right*, respectively, and the algorithm specifies the order in which they are added, updated, or discarded to achieve the desired result (Listing 6). Once this sequence is complete at each point, the final histogram is stored as *H*.

```
// Create a unique thread for each row
forall center in B.localSubdomain()[..,first_point] {

  // Calculate histograms and habitat heterogeneity at
  // leftmost point in this row
  var H_left: [0..<d_size] real(32) = 0;
  var H_center: [0..<d_size] real(32) = 0;
  var H_right: [0..<d_size] real(32) = 0;

  for m in Dom_left do {
    var tmp = Image[(center,first_point) + m];
    H_left[tmp] += 1;
  }
  for m in Dom_center do {
    var tmp = Image[(center,first_point) + m];
    H_center[tmp] += 1;
  }
  for m in Dom_right do {
    var tmp = Image[(center,first_point) + m];
    H_right[tmp] += 1;
  }

  var H = H_left + H_center + H_right;
  var H_old = H;

  // Perform habitat heterogeneity calculation using H
    ...

  // Calculate histograms and habitat heterogeneity
  //for all other points in this row
  for point in (first_point+1)..last_point do {

    // Reset and calculate H_right
    H_right = 0;
    for m in Dom_right do {
      var tmp = Image[(center,point) + m];
      H_right[tmp] += 1;
    }

    // Keep the old center, discard the old left,
    // and add the new right
    H = H_old + H_right - H_left;
    H_old = H;

    // Reset and calculate H_left
    H_left = 0;
    for m in Dom_left do {
      var tmp = Image[(center,point) + m];
      H_left[tmp] += 1;
    }

  // Perform habitat heterogeneity calculation using H
     ...
  }
```

Listing 6: Convolution across each row

The calculation of Q proceeds by dividing each bin of H by the total number of valid habitat pixels in the window, creating the discrete probability distribution **P** (Listing 7). Taking the outer product of **P** with itself creates a probability matrix, which is multiplied elementwise by the dissimilarity matrix D_{ij}. The final value of Q is calculated by performing a sum reduction of the result. A slight modification to the original Rao's Q formula that we use is to multiply Q by the proportion of valid habitat pixels in the window, which is saved as *habitat_frac*. This naturally scales down the calculated diversity in regions where there are a small number of good pixels, such as along coastlines or adjacent to deep water. This is a completely optional step and does not add any significant complexity to the algorithm, but we employ it as an alternative to simply neglecting points if they fail to have a certain proportion of good points in the window.

```
// Count number of valid habitat pixels
var num_habitat_pixels = (+ reduce H[1..(d_size-2)]);
var habitat_frac = num_habitat_pixels / Mask_Size;

var P = H / num_habitat_pixels;

var Q = + reduce (dissimilarity * outer(P,P));
// Scale Q by habitat_frac
Output[center,point] = (habitat_frac * Q);
```

Listing 7: Reduction of histogram

The final phase of RapidQ involves writing *OutputArray* to file. This step uses Chapel's C-interoperability feature to create an interface to the NetCDF-C library [25], which saves data in a hierarchical data format that is favored in the geosciences. Akin to the way the input image was read, a single output NetCDF file is written to in parallel by creating a separate task on each locale using a **coforall** loop. Each locale writes its piece of *OutputArray* to the appropriate position in the file in order to create an assembled mosaic like in Figure 2b. The final result mirrors the input image and clearly indicates hotspots of habitat diversity (Figure 3).

4 RESULTS AND PERFORMANCE

A prototypical test problem for RapidQ was created using a geomorphic habitat map for Roatan Island, Honduras (Figure 3), which was generated using the procedure outlined in Section 3.1. Given the ACA resolution of 5 meters and the scale shown in the bottom right corner of the Figure, this map contains approximately $11,000 \times 4,000$ pixels. Calculating Q using a 100-hectare window around each point, a naive calculation of the habitat heterogeneity map would require $O(10^{12})$ floating-point calculations. Anecdotally, a serial execution of such a calculation using MATLAB took over one week to complete on a laptop, and nearly three days to complete using Microsoft Azure cloud services (no formal timings are available). This program *did not* employ the solution strategy in Section 3, however, so the improvement of the Chapel program is due to a combination of algorithmic efficiency as well as Chapel's inherent performance. We emphasize that a direct comparison of Chapel versus MATLAB is not the focus of this paper (nor would be useful for our research purposes), and the performance metrics shown here are only with regard to the Chapel implementation.

The RapidQ version of this problem was tested on the Cheyenne supercomputer at the National Center for Atmospheric Research. Cheyenne is a SGI ICE XA Cluster with 145,152 Intel Xeon processor cores in 4,032 dual-socket nodes (36 cores/node). Each

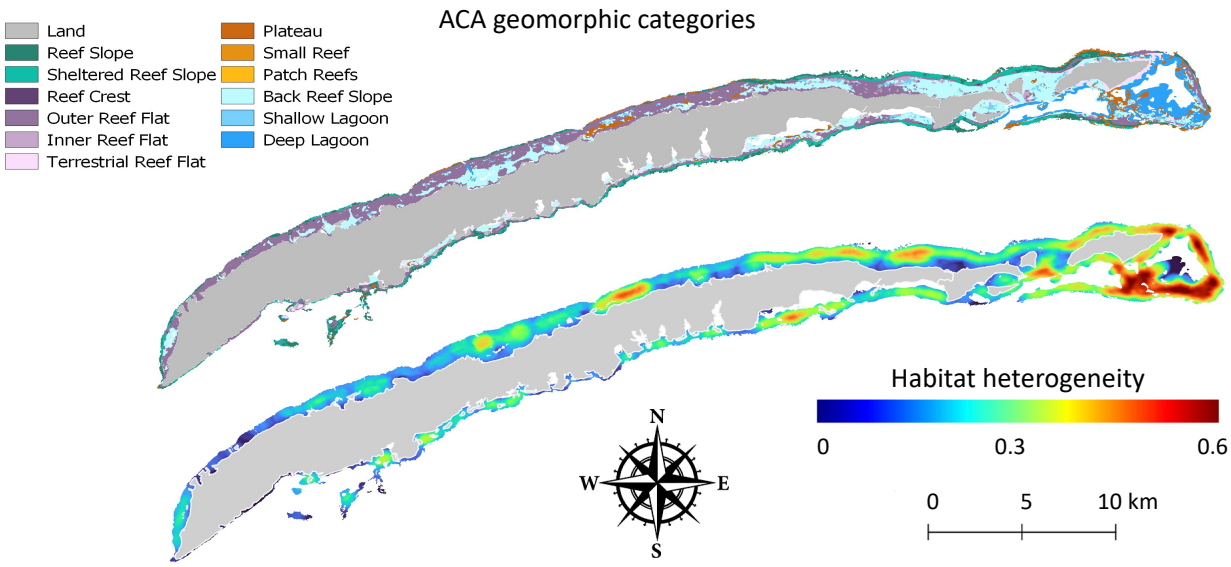

Figure 3: Top: ACA geomorphic habitat map for Roatan Island, Honduras. Bottom: Habitat heterogeneity calculated using the above habitat map.

processor is a 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) performing at 16 flops per clock cycle. The tests used Chapel version 1.28, with settings *CHPL_COMM=gasnet* and *CHPL_COMM_SUBSTRATE=ibv*. RapidQ was deployed on 10 nodes (360 cores) for 20 trials, reaching completion in an average of 6.2 seconds. This performance is achieved through a combination of RapidQ's solution strategy (reducing computational complexity by $L \approx 225$) and parallelism (x 360), resulting in an overall speedup of roughly five orders of magnitude.

Weak and strong scaling analyses of RapidQ were also performed on Cheyenne, with both types conducted on $\eta = \{1, 2, 4, 8, 16, 32, 64, 128\}$ nodes. For the weak scaling analysis, an artificial habitat map was created by filling an array with 10^8 points per node with random integers between 0 and 13, mimicking the geomorphic diversity calculation shown in Figure 3 (12 valid habitat types, plus land and deep water). The convolution window was configured to contain 10^6 pixels.

Two variants of the weak analysis were performed. The first variant initialized the image as a square, such that its shape was $10^4\sqrt{\eta} \times 10^4\sqrt{\eta}$ pixels. The second variant initialized the image as an elongated rectangle with $10^4\eta \times 10^4$ pixels. On a per-node basis, both variants have the same number of pixels, but the second variant contains $\sqrt{n}$ more points along its left edge, which is the most computationally expensive part of the image. Therefore, comparison of the two variants tests the hypothesis that the computation time is improved by minimizing the number of left edge points per locale, e.g. following the locale reshaping strategy in Listing 4.

Figure 4a shows the results of the weak scaling analysis, where the plots indicate an average over 20 independent trials. The plots are normalized by the average time to completion on a single node, where a value of 1 indicates perfectly linear scaling (the computation is performed exactly as fast as using purely local memory). Overall RapidQ demonstrates excellent weak scaling, maintaining a parallel efficiency greater than 0.8 out to 16 nodes and almost 0.6 even at 128 nodes. Interestingly, the performance between the square and rectangular domain shapes retains parity through about 16 nodes as well, but beyond that the square domain computation completes progressively more quickly. This result supports shaping the **Block** distributed domain as shown in Figure 2, though it also indicates that the performance hit would not be too severe if this approach was not taken.

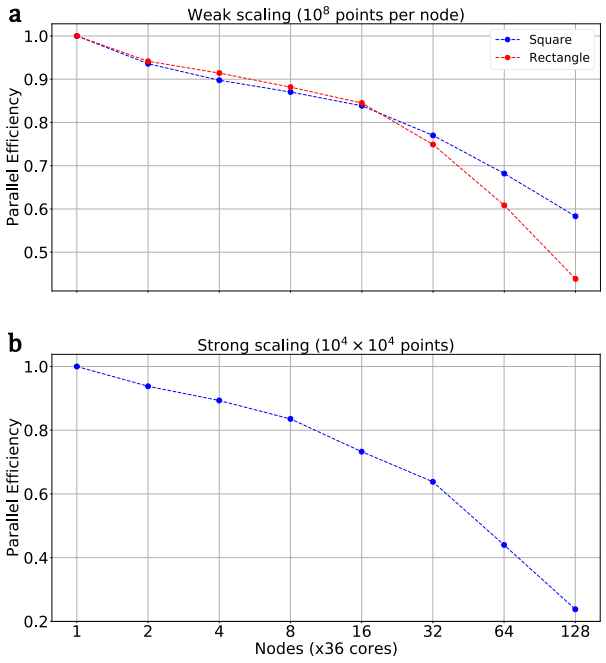

Figure 4: a) Weak scaling analysis. b) Strong scaling analysis.

Figure 4b shows the strong scaling analysis, which uses a fixed image size of $10^4 \times 10^4$ pixels for all node counts. Here the values are normalized by the average time to completion on a single node *times* the number of nodes, so that a value of 1 indicates that the time to completion on m nodes would be $1/m$ (smaller values imply slower execution). RapidQ shows fairly little dropoff in parallel efficiency even at very high levels of parallelism, though these results would suggest there is little speedup to be gained beyond 64 nodes for this problem size. The exact reason for the steeper dropoff in performance at high node counts is unclear.

In practice one would likely never need to scale beyond 32 nodes in any case. For a problem needing > 32 nodes the time to simply preprocess the ACA image would massively exceed the processing time in RapidQ, and memory would be a major limiting factor. So for all practical purposes the relevant node counts for RapidQ would be 16 or less, where both strong and weak scalings show excellent results.

5 CONCLUSION

RapidQ employs several powerful features in Chapel to both parallelize and streamline the calculation of habitat heterogeneity. Chapel's **Block**-distributed arrays allow the embarrassingly parallel nature of the computation to be exploited, and for the program to achieve near-linear performance improvements as it is scaled up. The use of **sparse** domains permits significant improvements to the computational algorithm itself, avoiding a substantial number of redundant calculations and leading to a massive performance increase independently from the parallelism. In total, Chapel allows habitat heterogeneity maps to be calculated multiple orders of magnitude faster than the incumbent MATLAB program, reducing the time to completion from days down to seconds. Furthermore, the final code is very concise and expressive, making it easy to build upon for the future.

The RapidQ application is a compelling and impactful example of how parallel programming with Chapel can enable cutting-edge science. The usage of RapidQ shown here focuses on a highly relevant and urgent problem in marine ecology and conservation, which is to inform the location and management of marine protected areas, as well as helping to shape field campaign strategies. Its performance has spurred investigation into problems that were previous thought to be intractable, such as creating a global database of habitat heterogeneity calculations at multiple scales. It has also helped to engender other, more challenging research problems; for example, the calculation of *spectral diversity* [34], which is even more computationally intense than habitat diversity and requires a multi-dimensional version of Rao's Quadratic Entropy.

There is significant opportunity for RapidQ to grow and to be applied in fields besides marine ecology, and considerations are being made about how best to disseminate it among the scientific community. A separate effort is underway to improve its performance even more, for example via deployment on graphical processing units (GPUs) or through further improvements to the algorithm. In summary, its current successes are motivating further work, and we expect RapidQ to have many opportunities to prove its value going forward.

6 ACKNOWLEDGEMENTS

We would like to acknowledge high-performance computing support from Cheyenne (doi:10.5065/D6RX99HX) provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation. We are also grateful for excellent conversations and brainstorming with Art Gleason, which have contributed significantly to this research effort.

REFERENCES

[1] Mitchell B. Lyons, Chris M. Roelfsema, Emma V. Kennedy, Eva M. Kovacs, Rodney Borrego-Acevedo, Kathryn Markey, Meredith Roe, Doddy M. Yuwono, Daniel L. Harris, Stuart R. Phinn, et al. 2020. Mapping the world's coral reefs using a global multiscale earth observation framework. *Remote Sensing in Ecology and Conservation* 6, 4 (2020), 557–568.

[2] Zoltán Botta-Dukát. 2005. Rao's quadratic entropy as a measure of functional diversity based on multiple traits. *Journal of vegetation science* 16, 5 (2005), 533–540.

[3] Alessandro Chiarucci. 2007. To sample or not to sample? That is the question... for the vegetation scientist. *Folia Geobotanica* 42 (2007), 209–216.

[4] Mayeul Dalleau, Serge Andrefouet, Colette CC Wabnitz, Claude Payri, Laurent Wantiez, Michel Pichon, KIM Friedman, Laurent Vigliola, and Francesca Benzoni. 2010. Use of habitats as surrogates of biodiversity for efficient coral reef conservation planning in Pacific Ocean islands. *Conservation Biology* 24, 2 (2010), 541–552.

[5] Shawna A Foo and Gregory P Asner. 2019. Scaling up coral reef restoration using remote sensing technology. *Frontiers in Marine Science* (2019), 79.

[6] Kevin J Gaston. 2000. Global patterns in biodiversity. *Nature* 405, 6783 (2000), 220–227.

[7] EP Green, PJ Mumby, AJ Edwards, and CD Clark. 1996. A review of remote sensing for the assessment and management of tropical coastal resources. *Coastal management* 24, 1 (1996), 1–40.

[8] Alastair R Harborne, Peter J Mumby, Kamila ZŻychaluk, John D Hedley, and Paul G Blackwell. 2006. Modeling the beta diversity of coral reefs. *Ecology* 87, 11 (2006), 2871–2881.

[9] John D Hedley, Chris M Roelfsema, Iliana Chollett, Alastair R Harborne, Scott F Heron, Scarla J. Weeks, William J Skirving, Alan E Strong, C Mark Eakin, Tyler RL Christensen, et al. 2016. Remote sensing of coral reefs for monitoring and management: a review. *Remote Sensing* 8, 2 (2016), 118.

[10] Joaquín Hortal, Francesco de Bello, José Alexandre F Diniz-Filho, Thomas M Lewinsohn, Jorge M Lobo, and Richard J Ladle. 2015. Seven shortfalls that beset large-scale knowledge of biodiversity. *Annual Review of Ecology, Evolution, and Systematics* 46 (2015), 523–549.

[11] Matthew S Kendall, Thomas J Miller, and Simon J Pittman. 2011. Patterns of scale-dependency and the influence of map resolution on the seascape ecology of reef fish. *Marine Ecology Progress Series* 427 (2011), 259–274.

[12] Emma V Kennedy, Chris M Roelfsema, Mitchell B Lyons, Eva M Kovacs, Rodney Borrego-Acevedo, Meredith Roe, Stuart R Phinn, Kirk Larsen, Nicholas J Murray, Doddy Yuwono, et al. 2021. Reef Cover, a coral reef classification for global habitat mapping from remote sensing. *Scientific Data* 8, 1 (2021), 196.

[13] Anders Knudby, Ellsworth LeDrew, and Candace Newman. 2007. Progress in the use of remote sensing for coral reef biodiversity studies. *Progress in Physical Geography* 31, 4 (2007), 421–434.

[14] Anders Knudby, Chris Roelfsema, Mitchell Lyons, Stuart Phinn, and Stacy Jupiter. 2011. Mapping fish community variables by integrating field and satellite data, object-based image analysis and modeling in a traditional Fijian fisheries management area. *Remote Sensing* 3, 3 (2011), 460–483.

[15] Robert H MacArthur. 1984. *Geographical ecology: patterns in the distribution of species*. Princeton University Press.

[16] Norman WH Mason, David Mouillot, William G Lee, and J Bastow Wilson. 2005. Functional richness, functional evenness and functional divergence: the primary components of functional diversity. *Oikos* 111, 1 (2005), 112–118.

[17] Peter J Mumby. 2001. Beta and habitat diversity in marine systems: a new approach to measurement, scaling and interpretation. *Oecologia* 128 (2001), 274–280.

[18] Harini Nagendra. 2001. Using remote sensing to assess biodiversity. *International journal of remote sensing* 22, 12 (2001), 2377–2400.

[19] Michael W Palmer, Peter G Earls, Bruce W Hoagland, Peter S White, and Thomas Wohlgemuth. 2002. Quantitative tools for perfecting species lists. *Environmetrics: The official journal of the International Environmetrics Society* 13, 2 (2002), 121–137.

[20] Nathalie Pettorelli, Jon Olav Vik, Atle Mysterud, Jean-Michel Gaillard, Compton J Tucker, and Nils Chr Stenseth. 2005. Using the satellite-derived NDVI to assess ecological responses to environmental change. *Trends in ecology & evolution* 20, 9 (2005), 503–510.

[21] Nathalie Pettorelli, Martin Wegmann, Leigh Gurney, and Gregoire Dubois. 2016. Monitoring protected areas from space. *Protected Areas: Are They Safeguarding Biodiversity?* (2016), 242–259.

[22] Sam J Purkis, Arthur CR Gleason, Charlotte R Purkis, Alexandra C Dempsey, Philip G Renaud, Mohamed Faisal, Steven Saul, and Jeremy M Kerr. 2019. High-resolution habitat and bathymetry maps for 65,000 sq. km of Earth's remotest coral reefs. *Coral Reefs* 38 (2019), 467–488.

[23] Andy Purvis and Andy Hector. 2000. Getting the measure of biodiversity. *Nature* 405, 6783 (2000), 212–219.

[24] C Radhakrishna Rao. 1982. Diversity and dissimilarity coefficients: a unified approach. *Theoretical population biology* 21, 1 (1982), 24–43.

[25] Russ Rew and Glenn Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.

[26] Carlo Ricotta and Michela Marignani. 2007. Computing β-diversity with Rao's quadratic entropy: A change of perspective. *Diversity and Distributions* 13, 2 (2007), 237–241.

[27] Duccio Rocchini, Doreen S Boyd, Jean-Baptiste Féret, Giles M Foody, Kate S He, Angela Lausch, Harini Nagendra, Martin Wegmann, and Nathalie Pettorelli. 2016. Satellite remote sensing to monitor species diversity: Potential and pitfalls. *Remote Sensing in Ecology and Conservation* 2, 1 (2016), 25–36.

[28] Duccio Rocchini, Matteo Marcantonio, and Carlo Ricotta. 2017. Measuring Rao's Q diversity index from remote sensing: An open source solution.

Ecological indicators 72 (2017), 234–238.

[29] Chris Roelfsema, Eva Kovacs, Juan Carlos Ortiz, Nicholas H Wolff, David Callaghan, Magnus Wettle, Mike Ronan, Sarah M Hamylton, Peter J Mumby, and Stuart Phinn. 2018. Coral reef habitat mapping: A combination of object-based image analysis and ecological modelling. *Remote Sensing of Environment* 208 (2018), 27–41.

[30] Kenichiro Shimatani. 2001. On the measurement of species diversity incorporating species differences. *Oikos* 93, 1 (2001), 135–147.

[31] Thomas B Smith, Salit Kark, Christopher J Schneider, Robert K Wayne, and Craig Moritz. 2001. Biodiversity hotspots and beyond: the need for preserving environmental transitions. *Trends in Ecology & Evolution* 16, 8 (2001), 431.

[32] Woody Turner, Carlo Rondinini, Nathalie Pettorelli, Brice Mora, Allison K Leidner, Zoltan Szantoi, Graeme Buchanan, Stefan Dech, John Dwyer, Martin Herold, et al. 2015. Free and open-access satellite data are key to biodiversity conservation. *Biological Conservation* 182 (2015), 173–176.

[33] Simon Van Wynsberge, Serge Andrefouet, Mélanie A Hamel, and Michel Kulbicki. 2012. Habitats as surrogates of taxonomic and functional fish assemblages in coral reef ecosystems: a critical analysis of factors driving effectiveness. *PloS one* 7, 7 (2012), e40997.

[34] Steven D Warren, Martin Alt, Keith D Olson, Severin DH Irl, Manuel J Steinbauer, and Anke Jentsch. 2014. The relationship between the spectral diversity of satellite imagery, habitat heterogeneity, and plant species richness. *Ecological Informatics* 24 (2014), 160–168.

Received 14 April 2023