Parallel Variable-Resolution Bathymetric Estimation with Static Load Balancing

B.R. Calder[a,*]

[a]Center for Coastal and Ocean Mapping and NOAA-UNH Joint Hydrographic Center, University of New Hampshire, Durham, NH 03824, USA

Abstract

A method for partitioning a large computation task (direct, variable resolution bathymetric grid construction from raw observations) into thread-parallel code is described. Based on the data density estimated for the first pass of the CHRT algorithm, this algorithm statically partitions the estimation task into spatially distinct blocks of approximately equal total data observation count so that each can be executed in parallel and be expected to complete approximately concurrently. No communication between blocks or further load balancing is therefore required. A branch-and-bound algorithm is used to control the complexity of the partitioning task, but the computation time increases significantly as more partitions are required, leading to a degree of diminishing returns for allocating further computational resources and suggesting alternative approaches for high thread-count systems. Speed-up of the algorithm over a pair of test datasets (using real-world hydrographic survey data) shows that the performance consistently improves with the number of computational tasks assigned, initially (super-) linearly, although ultimately sub-linearly as other resource sharing limitations take over. An overall speedup of 4.1 times is demonstrated with a quad-core single-processor workstation.

Keywords: Parallel processing, CHRT, CUBE, Data-driven Estimation, Branch and Bound, Bathymetric Data Processing, Surface Estimation

1. Introduction

Bathymetry is often a base layer in marine spatial modelling, providing an important constraint on the physical environment (e.g., defining the waveguide for acoustic propagation studies) and driving other analyses. The reconstruction of a best-estimate of depth (or, more generally, any scalar field) within a given area based on remote-sensed observations (Krishnan et al., 2010; Hofierka et al., 2017) is therefore an important problem with many practical applications, including ocean mapping, geophysical modelling, coastal zone management, and nautical charting.

The problem is computationally challenging. The datasets are often large (order $10^9 - 10^{10}$ observations), and the algorithms can be complex due to

*E-mail: brc@ccom.unh.edu, Tel.: +1 603 862 0526. Declarations of Interest: None.

dataset features such as observational blunders (Calder and Mayer, 2003; Debese et al., 2012; Isenburg et al., 2006). The datasets may also have a spatial-varying data density, requiring spatial adaptation of reconstruction resolution to avoid spatial aliasing or over-smoothing. (In hydrographic practice, over-smoothing could result in missed navigationally significant objects, which are a primary concern.) In many cases, the data density is approximately a function of the water depth, so deeper areas can only be reconstructed at significantly lower resolution; these changes can happen within very short distances, for example in fjord-like environments.

Efficient computation of the reconstruction is therefore essential. In addition to minimising processing time, fast computation allows for more advanced algorithms to be built around the basic estimation task. For example, it can be difficult to compensate for the effects of slope in CHRT (the estimation algorithm considered here) *a priori* because there is no good estimate of slope until the reconstruction is computed, but that reconstruction is biased by lack of slope compensation. Iterating to solution is plausible, but if the algorithm is expensive to compute, the iterations might take sufficiently long as to render the method ineffective.

Computing a reconstruction in parallel is therefore advantageous. Many algorithms, however, are either global (i.e., require all data in an area to proceed, for example the surface fitting of Debese et al. (2012)), or non-local (i.e., need to interact with nearby estimation sites to complete the estimation, for example the continuous spline in tension method of Smith and Wessel (1990)), which can make them difficult to segment for parallel implementation. Sending packets of observations to different threads for update of a common data structure generally requires a significant level of complexity in the data structure to allow simultaneous access, while splitting spatially has difficulties in ensuring that the sub-tasks are well balanced. A solution which is well balanced, does not require memory locking, and can be scaled to many computational resources is therefore key.

Load-balancing, or the more general case of splitting a non-even workload computational domain over a number of computational resources so as to achieve some metric, is a common problem, and has therefore received much attention in the literature. The problem of finding the optimal general partition in two dimensions is known to be NP-Hard (Khanna et al., 1998), or NP-Complete in some cases such as the Generalized Block Distribution (Grigni and Manne, 1996); even achieving a bound on the performance within a factor of two is NP-Hard (Aspvall et al., 2001). Consequently, most of the research on the matter has revolved around finding better approximations to the problem, and lowering the upper bound on performance (see, e.g., (Manne and Sørevik, 1996), (Aspvall et al., 2001), (Berman et al., 2001), (Lorys and Paluch, 2003), (Saule et al., 2012), etc.).

The most general case of partitioning would allow for arbitrary segments to be assigned to a computational resource; in keeping with Tobler's Law (Tobler, 1970), however, most solutions focus on assigning rectangular areas in order to maintain advantages of spatial locality in the computations. Most work has been done on recursive partitions (Berger and Bokhari, 1987), the rectilinear partition (Nichol, 1991), also known as the Generalized Block Distribution (Grigni and Manne, 1996), and the jagged partition (Manne and Sørevik, 1996), also known as the Semi-generalized Block Distribution (see (Saule et al., 2012) for a good overview), although there are many more potential partitioning schemes.

The complexity and performance of the best known algorithms for the various approximations vary widely. Saule et al. (2012) compare a variety of algorithms, with different heuristics designed to achieve better performance, including hierarchical sub-division, rectilinear division, and jagged partitions, for which polynomial time algorithms are available. They conclude that their jagged partition variant, or a hierarchical subdivision may achieve better load balancing than other algorithms while still being runtime efficient, while a combination of algorithms (a "hybrid") can improve on achievable balance even further. (Intriguingly, the best case load imbalances reported are of similar magnitude to those reported here.)

These approaches, while efficient, are more constrained than is required in the case presented here, primarily due to a basic assumption that communications costs between the segments of the partition are important. Here, however, given the observations, each segment of the partition can compute independently its part of the solution to the overall estimation problem, so there is no limitation to the arrangement of the segments, and a more general solution can be attempted. Furthermore, the focus here is on splitting the overall task over a relatively small number of computational resources, since the primary goal is a multithreaded, single CPU solution, since it remains the most commonly available computational resource for most users in the field (Qin and Zhan, 2012). Consequently, the more sophisticated heuristic-based algorithms are not required, and an optimal solution of the (constrained) partitioning problem can be used, simplifying the implementation. (The problem of scaling to higher numbers of computational resources is considered in Section 4.)

In this paper, therefore, a spatial partitioning algorithm is proposed for the CHRT (CUBE with Hierarchical Resolution Techniques) algorithm (Calder and Rice, 2017) which takes advantage of the structure of CHRT to ensure that each computational resource can operate independently of the others without communication or interlocks so long as they have global access to all of the observations. (Section 2.1 has an outline of the CHRT algorithm.) The algorithm uses the data density estimates computed during the first pass of the CHRT algorithm to drive the partition, which allows for the partition segments to contain approximately the same number of observations, and consequently to have nearly uniform computation time. Our goal here is not to minimise the maximum cost for any segment (c.f. (Saule et al., 2012), or (Muthukrishnan and Suel, 2005)) but to have even load across all of the computational resources, hence keeping them all busy for the minimal time with highest efficiency. Operating system file caching assists in delaying IO-limited performance (i.e., where recently used files remain in memory, and therefore are not subject to spinning-disc latency), and a branch-and-bound evaluator allows the partition to be computed efficiently. Use of the partitioning algorithm allows ready extension of the CHRT algorithm to a multi-threaded implementation, with consequent performance improvement.

The remainder of the paper outlines the relevant features of the CHRT algorithm that support the partitioning algorithm, its implementation, and the performance improvements achieved using commodity single-processor workstation hardware. Finally, some perspectives on the ability of the algorithm to be scaled, and generalised to a distributed (i.e., network-connected) implementation, are offered.

2. Methods

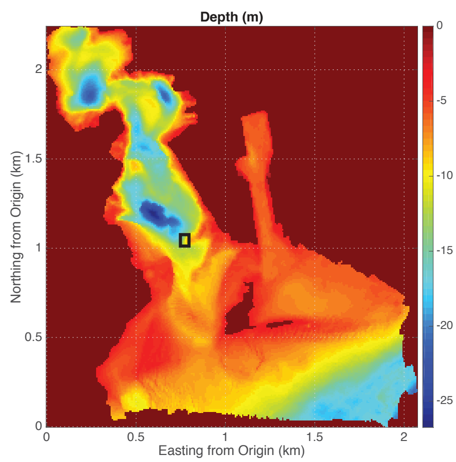
2.1. Core Estimator

The CHRT (CUBE with Hierarchical Resolution Techniques) algorithm (Calder and Rice, 2017), a development of the CUBE (Combined Uncertainty and Bathymetry Estimator) algorithm (Calder and Mayer, 2003) was used as the basis for the current work. The CHRT algorithm was developed to estimate variable resolution depths from raw observational data based on the premise that in regions where there is higher data density it should be possible to reconstruct with smaller sample spacings, giving higher resolution reconstructions of the surface. The algorithm starts with a low-resolution virtual tile (Yıldırım et al., 2015) grid across the area of interest, and at each grid node estimates the data density of the observations. A piecewise constant sample spacing (PCSS) grid is then constructed by replacing each low-resolution grid cell with a regular grid at the sample spacing determined by the data density, after which a variable resolution depth reconstruction can be computed in a second pass. Figure 1 shows an example of the first pass of the algorithm applied to a hydrographic survey in Woods Hole, MA.

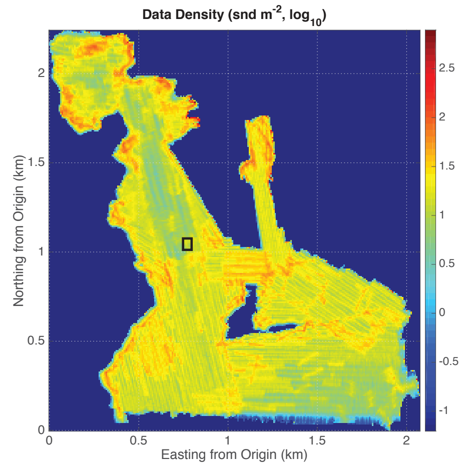
A basic problem for any parallel algorithm is how to split the task into manageable sub-tasks. The low-resolution grid used in CHRT allows for a relatively simple solution to this problem since the refinement grids established after the first pass of the algorithm are by design constrained to lie entirely within their parent cell, Figure 2. Consequently, given the observations that contribute to the cell (which may include some immediately adjacent in order to avoid edge effects), the computation of each cell is independent of the others, and therefore can be processed without any communication or interlock, and there is no requirement for "ghost cell" edge buffers (Tesfa et al., 2011). Any sub-group of cells can therefore be assigned to any available computational resource, so long as it has access to all of the observations. This decomposition of the base algorithm avoids having to design a variant for parallel implementation, with all of the associated development and maintenance costs (Hofierka et al., 2017). The CHRT Conformance Test Suite (Calder and Plumlee, 2017) ensures equivalence of serial and parallel computation.

2.2. Partitioning Scheme

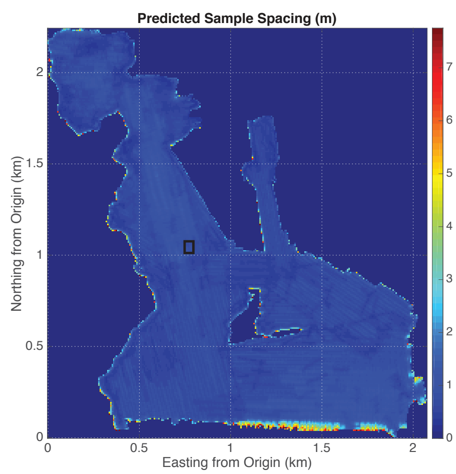
For the CHRT algorithm, the processing cost is reasonably approximated by the number of observations that have to be assimilated at a particular reconstruction location. A plausible load balancing partitioning scheme is therefore



(a) Low resolution bathymetry (m); the gross features are clear, but significant objects (e.g., mooring blocks, pilings) require the variable-resolution data to resolve.



(b) Data density (snd m^{-2}, log scale). Note the hole in the middle caused by a very shallow area that was not surveyed for safety.



(c) Predicted sample spacing (m) for refinements.

Figure 1: Example of the CHRT algorithm applied to a NOAA survey in Woods Hole, MA, showing (a) first-pass low-resolution depth estimate, (b) data density estimate (note logarithmic scale), and (c) estimated sample spacing for each low-resolution cell. Black rectangle is shown in detail in Figure 2. This figure is reproduced from Calder and Rice (2017).

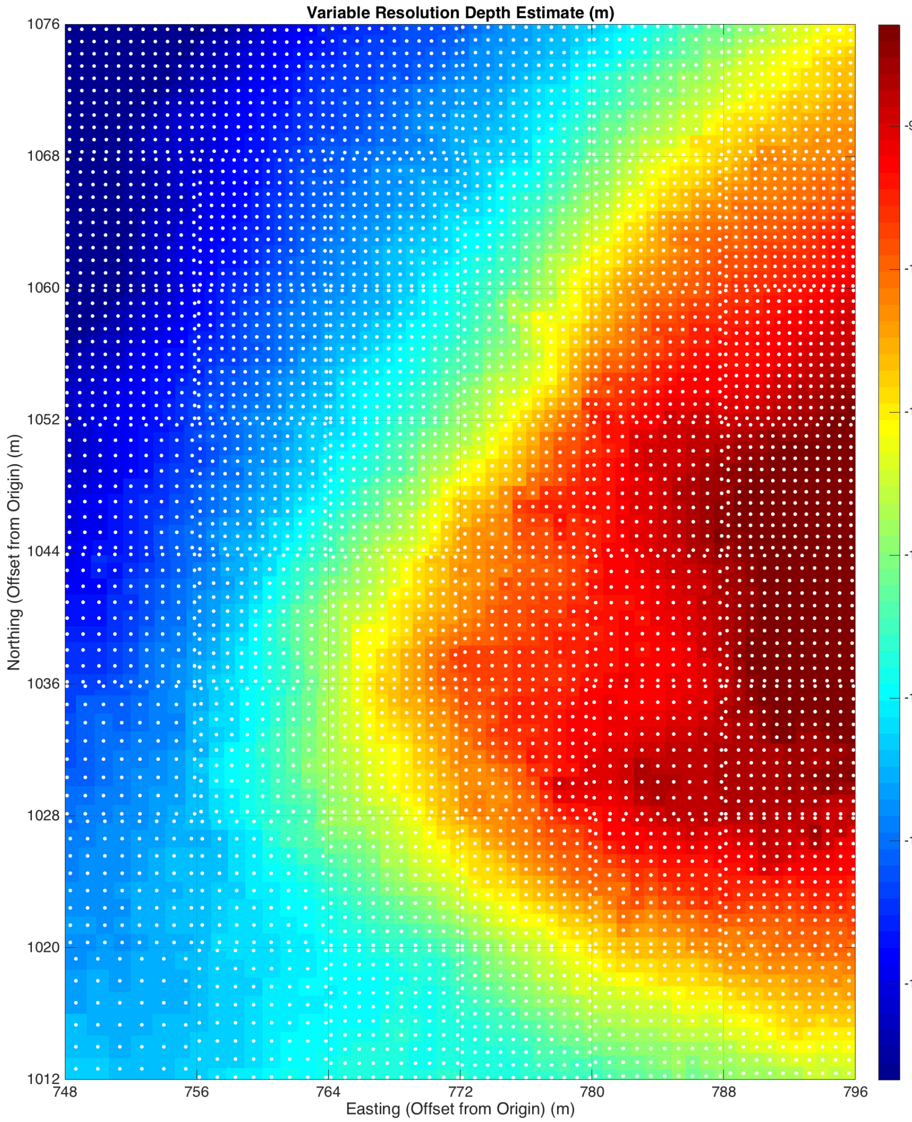


Figure 2: Example of variable resolution depth reconstruction, and the location of variable resolution reconstruction points derived from data density estimates during the first pass for the data indicated in the black rectangle in Figure 1. Each refinement grid is constrained to be entirely within the parent low-resolution grid cell (here, at 8 m intervals). Colours represent the estimated depths; white dots mark the locations of the variable resolution estimation points. Labels on the geographic axes mark the edges of the low-resolution cells containing the refined (white dot) grids. This figure is reproduced from Calder and Rice (2017).

to split the overall area to be processed into sub-areas that contain approximately the same number of observations. (Alternatives, such as partitioning by input files and then recombining partial grids, or dynamic partitioning of subgroups (Yıldırım et al., 2015) would lead to serialized code or higher communications costs, respectively.) In order to facilitate this, the partition algorithm assumes that a spatial observation density estimate is available from the first pass of the CHRT algorithm (this is a core component of the base algorithm).

Let the data density estimates be arranged in a grid, $\rho(u,v), 0 \leq u < U, 0 \leq v < V, (u,v) \in \mathbb{Z}^2$, with N total observations in the dataset spread through the area. The goal is to find an optimal partition of the overall domain $\mathsf{S}_0 = \{(u,v) : (u,v) \in [0,U) \times [0,V)\}$ into C segments, one for each computational resource, each containing an equal number of observations. The grid could be partitioned into arbitrarily-shaped groups of cells with equal numbers of observations, but to contain the complexity consider admitting only north-south or east-west segment boundaries (Berger and Bokhari, 1987).

The algorithm solves this problem recursively over the general segment of the grid, $\mathsf{S} = \{(u,v) : (u,v) \in [u_0, u_1-1] \times [v_0, v_1-1], 0 \leq u_0 < u_1 < U, 0 \leq v_0 < v_1 < V\}$, which is to be split into $C_\mathsf{S} \leq C$ segments each of $N_\mathsf{S}/C_\mathsf{S}$ observations, where $N_\mathsf{S} \leq N$ is the observation count for S. The recursion root is $\mathsf{S} = \mathsf{S}_0$, $C_\mathsf{S} = C$, $N_\mathsf{S} = N$.

Consider first an algorithm that enumerates all possible partitions. Each partition line is placed to split S into sub-segments of some multiple $cN_\mathsf{S}/C_\mathsf{S},\ 1 \leq c < C_\mathsf{S}$ of the observations in the segment, Figure 3. Given an area A for each cell in the domain, the partition line is placed at $\{u_c,\ 1 \leq c < C_\mathsf{S}\}$ where

$$u_c = \max_{u_0 \leq x < u_1} \left\{ x : \sum_{u=u_0}^{x} \sum_{v=v_0}^{v_1-1} \rho(u,v)A < cN_\mathsf{S}/C_\mathsf{S} \right\} \tag{1}$$

or at $\{v_c,\ 1 \leq c < C_\mathsf{S}\}$ where

$$v_c = \max_{v_0 \leq y < v_1} \left\{ y : \sum_{v=v_0}^{y} \sum_{u=u_0}^{u_1-1} \rho(u,v)A < cN_\mathsf{S}/C_\mathsf{S} \right\} \tag{2}$$

for north-south and east-west partition lines, respectively, giving $2(C_\mathsf{S} - 1)$ potential partial segmentations.

For north-south partitions, these potential positions split S into

$$\mathsf{S}_L = \{(u,v) : (u,v) \in [u_0, u_c] \times [v_0, v_1 - 1]\} \tag{3}$$

and

$$\mathsf{S}_R = \{(u,v) : (u,v) \in [u_c + 1, u_1 - 1] \times [v_0, v_1 - 1]\} \tag{4}$$

so that $\mathsf{S} = \mathsf{S}_L \cup \mathsf{S}_R$ and $\mathsf{S}_L \cap \mathsf{S}_R = \emptyset$, and equivalently for east-west partitions. The algorithm can then be applied recursively to S_L and S_R, each now of N_{S_L} and N_{S_R} observations, respectively, with a target of $C_\mathsf{S} - 1$ computational resources assigned to them. The recursion terminates when $C_\mathsf{S} = 1$. Since each

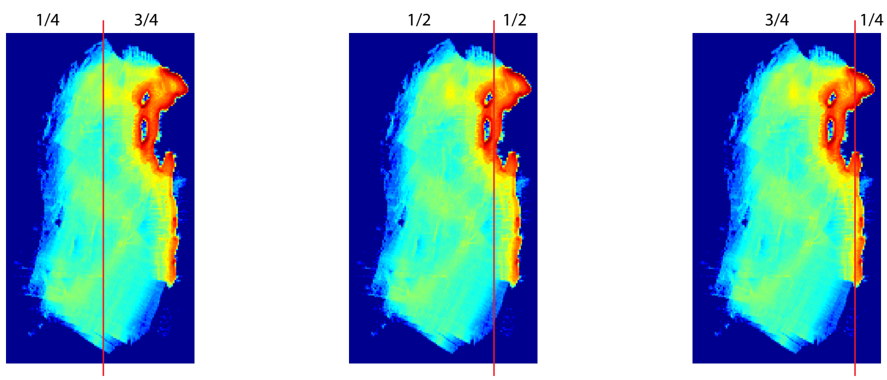


Figure 3: Example of potential position points for the first stage of the partitioning algorithm with four computational resources, where the goal is to split off one, two or three quarters of the observations (with regions of two or three quarters being split in later stages of algorithm); the background images are the data density estimated from the observations in the Ernest Sound, AK test dataset (see Section 3.1). Note the significantly larger area associated with the 1/4 position (left) image compared to the 3/4 position (right) image, due to the significantly lower data density to the west of the partition line in the former case. Solving for a different number of computational resources would have different partition points as the algorithm attempts to split off different sized portions of the data.

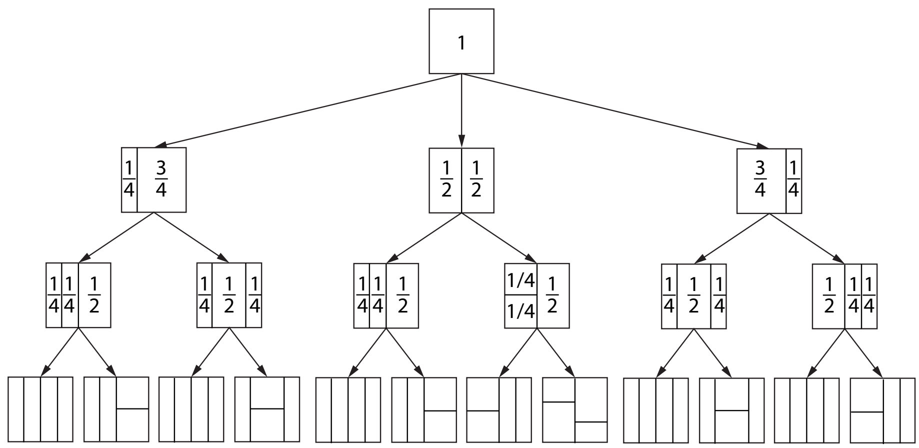


Figure 4: Example of a partial hierarchical tree for four computational resources, indicating some of the potential combinations of north-south and east-west partitions applied in sequence to split the problem into four segments, each of one quarter of the problem. Note the practical redundancy in many of the logically separate partitions, which can reduce the efficiency of the search if the whole tree is enumerated.

partition can be placed either north-south or east-west on each occasion, this algorithm naturally leads to a tree of potential partition schemes, Figure 4. Many of the partitions generated are logically separate (i.e., the order in which the partition lines were generated are different) but practically the same (i.e., the resulting segments are identical). This can lead to implementation efficiencies, a topic pursued in the following section.

In principle, this algorithm can be applied from S_0 to enumerate all leaves of the partition tree. Due to the granularity of the low-resolution cells, it is unlikely that any given partition will exactly split the problem in C segments of N/C observations. The viability of the different leaves of the partition tree can therefore be assessed according to how closely they achieve this goal, with the closest match being the preferred solution. An example of an 8-partition applied to the data in Figure 3 is shown in Figure 5.

2.3. Partitioning Algorithm Implementation

In theory, the partition that best matches the ideal, even, distribution of observations could be determined by simply enumerating the tree of potential partitions. The first stage of splitting has $2(C-1)$ potential splits; the second has $2(C-2)$, and so on, for a total of $2^{C-1}(C-1)!$ potential solutions. Some reduction in effort could be obtained by exploiting similarities in the potential partitions, but for any reasonable target number of segments the size of the tree rapidly makes a full enumeration intractable: for $C = 8$, for example, a reasonable choice for quad-core hyper-threaded processors, a total of 645,120 potential

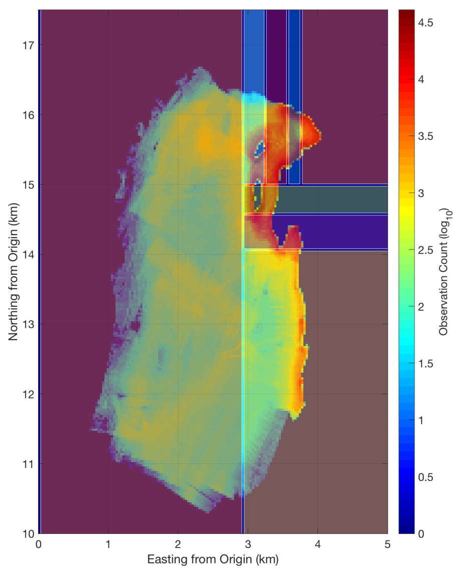


Figure 5: Example of a partition for eight computational resources applied to the Ernest Sound data (Figure 3). The segments selected by the algorithm are shown as white outlines (with semi-transparent colours) over the sounding count (in log-scale). Note that the segments are shown with boundaries slightly separated for clarity; in reality, they completely tile the computational area.

solutions would have to be enumerated; at $C = 16$, the total is 4.285×10^{16}. The goal is still to evaluate the whole tree, however, so the algorithm applies the "branch and bound" technique (Land and Doig, 1960) to avoid evaluating inefficient branches of the tree as often as possible, and applies heuristics to attempt to accelerate the process.

Consider the situation at any node in the tree of Figure 4. Assume that for any individual segment S there is a cost function $P(\mathsf{S}, N, C)$ that represents the penalty for not matching the nominal ideal observation count of N/C observations per segment. If $C_{\mathsf{S}} = 1$ (i.e., the segment represents the best approximation to a single quantum of observations), then the cost can be evaluated directly; otherwise, the potential refinements of the segment are $\mathsf{R} = \{v_1, \ldots, v_{C_{\mathsf{S}}-1}, h_1, \ldots, h_{C_{\mathsf{S}}-1}\}$ where the v_i represent north-south, and h_i east-west partitions, respectively, computed according to (1)–(2). Each refinement induces a pair of segments $(\mathsf{S}_P(r), \mathsf{S}_S(r)),\ r \in \mathsf{R}$ according to (3)–(4), corresponding to the segment prior to, and subsequent to, the partition location, respectively. The overall penalty for each potential refinement of the segment can therefore be computed as

$$
\begin{aligned}
P(r) &= P(\mathsf{S}_P(r) \cup \mathsf{S}_S(r), N, C) \\
&= P(\mathsf{S}_P(r), N, C) + P(\mathsf{S}_S(r), N, C),
\end{aligned}
\tag{5}
$$

with the individual penalties being evaluated recursively. Clearly, the optimal refinement is

$$
r^* = \arg\min_r P(r), \tag{6}
$$

and the parent node therefore has a "best known partition" penalty of $P(r^*)$.

At each node in the tree, the partitioning decisions made further up the tree lead to a penalty which the algorithm has already assumed in order to get to the decision point represented by the node. An allowable penalty can therefore be passed to each node by its parent, indicating the maximum penalty remaining to the branch for any refinement to be viable in comparison with the best available refinement elsewhere; testing against this limit can therefore reduce the number of evaluations that need to be attempted.

Let α_{S} be the available penalty provided to the node for segment S; to seed the recursion, let $\alpha_{\mathsf{S}_0} \to \infty$, or in practice the maximum value available. Clearly, if $P(\mathsf{S}_P(r), N, C) > \alpha_{\mathsf{S}}$, the proposed refinement is not viable, irrespective of $P(\mathsf{S}_S(r), N, C)$, and the evaluation of potential refinements of S_S need not be computed (and vice versa). (Observe that if $P(\mathsf{S}_P(r), N, C) < \alpha_{\mathsf{S}}$, then the bound for evaluating $\mathsf{S}_S(r)$ should be $P(\mathsf{S}_P(r), N, C) - \alpha_{\mathsf{S}}$, which can help to reject more potential refinements, further reducing the computation cost.) This bound can be incrementally tightened by observing that each refinement evaluated can provide a better target if $P(r) < \alpha_{\mathsf{S}}$. Therefore, define

$$
\alpha(0) = \alpha_{\mathsf{S}} \tag{7}
$$

$$
\alpha(r) = \min(\alpha(r-1), P(r)) \tag{8}
$$

so that the target that candidate refinements have to better tightens as good refinements are determined. The ordering in which refinements are attempted is essentially arbitrary, but the choice of which to evaluate first for most efficient evaluation is not. Given that one side or other of the partition might be eliminated from consideration after the other is evaluated, it is advantageous to evaluate first the side that is shallowest (i.e., with smallest N_{S}).

The efficiency of the pruning algorithm is maximised if the algorithm can establish a plausible solution (i.e., one close to the optimal) early in the sequence of evaluations, since it will lead to many more branches being pruned more quickly. There is no way to predict where a "good" solution would lie *a priori*, but a useful heuristic is to observe that splitting off a segment of N/C observations early in the sequence is unlikely to provide a good solution, since the split position can only be adjusted by a whole row or column of low-resolution cells, which can contain many observations. If, on the other hand, the first split breaks the area approximately in half (c.f. (Berger and Bokhari, 1987)), then any error in the observation count can be amortised over all of the remaining splits. The algorithm therefore starts at the $c = \lfloor C/2 \rfloor$ position, and moves outward towards either extreme, swapping sides after each step (i.e., evaluating at $\lfloor C/2 \rfloor, \lfloor C/2 \rfloor - 1, \lfloor C/2 \rfloor + 1, \lfloor C/2 \rfloor - 2, \ldots$).

Evaluating the number of observations within a proposed segment is the most expensive part of the partition computation. The count of observations within each low-resolution cell is, however, fixed. Therefore it is possible to utilise a variant of the summed-area table technique (Crow, 1984), also known as a prefix sum table (Ladner and Fischer, 1980), to cache cumulative sums and hence significantly improve the computation time.

There are a number of plausible definitions for the cost function. Here, a simple comparison against the nominal observation count per computational resource is used,

$$P(\mathsf{S}, N, C) = |N_{\mathsf{S}} - N/C| \,. \tag{9}$$

In the simplest case, $N_{\mathsf{S}} = \sum_{(u,v)\in\mathsf{S}} \rho(u,v)A$. Evaluation of cost functions in bounded arithmetic (i.e., where there is a maximum representable cost, $P_{\max}$) requires some care. In particular, saturation addition is required, so that if

$$P'(\mathsf{S}, N, C) = \min\left(P_{\max}, P(\mathsf{S}, N, C)\right) \tag{10}$$

is the bounded arithmetic representation of the cost function, then

$$\begin{aligned} P'(r) = P'(&\mathsf{S}_P, N, C) + \\ &\min\left(P'(\mathsf{S}_S, N, C), P_{\max} - P'(\mathsf{S}_P, N, C)\right). \end{aligned} \tag{11}$$

The CHRT algorithm utilises some observations from just outside an assigned computation domain so as to avoid any edge effects in the estimation. If these are ignored, then the computational cost of processing a segment will be underestimated, potentially significantly in shallow areas with dense observations. Let S_E be the annulus, one low-resolution cell wide, around segment S, with N_E observations. The CHRT algorithm, by default, uses observations only out to $\sqrt{2}W$ from the segment boundary for cells of width (and height) W m. The annulus therefore provides approximately $\sqrt{2}N_E$ observations (simplifying for the corner cells), and the total effective number of observations used in (9) is

$$N_{\mathsf{S}} = \sqrt{2} \sum_{(u,v)\in\mathsf{S}_E} \rho(u,v)A + \sum_{(u,v)\in\mathsf{S}} \rho(u,v)A. \tag{12}$$

2.4. Parallel Estimator Implementation

Although the core estimation algorithm is the same for serial and parallel computation, some care in staging is required. The CHRT algorithm supports virtual tiles, with memory-mapped files that are demand paged with least-recently-used cache replacement. The memory-mapped structures may cross segment boundaries, however, and to avoid interlocks it is therefore necessary for each computational resource to have a separate copy of the results of the first pass of the algorithm for the tiles associated with the segment assigned. Knowledge of the segment bounds allows this computation to be done *a priori*, and the parallel wrapper code can pre-copy the required files along with the base metadata for the data structure. After the initial configuration, the computational resources are independently scheduled as separate threads within the main process.

A producer-consumer pattern is used with one consumer implementing the estimator for each segment of the partition. The producer implements the Command pattern (Gamma et al., 1994) by constructing work packages for each stage of the computation, derived from an abstract interface, that are queued for all of the worker threads to execute. A modified barrier synchronisation pattern (Wilkinson and Allen, 2005) allows for the threads to be marshalled, indicating to the producer that all required computations have been completed (e.g., so that the client interface can determine when it is safe to request a reconstruction take place).

2.5. Partial Result Reassembly

As with the partitioning problem, reassembly of the partial results from each computational resource is made simpler because the segments are aligned with low-resolution cell boundaries. Each computational resource can therefore, on demand, generate its partial result and write them into the shared output data structure for the overall result without interlocks.

In theory, the partial reconstruction computations for each segment could be overlapped with any remaining primary computation in order to avoid any serial-code delays. The operational paradigm for reconstruction is user-driven, however, so it is not necessarily the case that reconstruction immediately follows primary computation. The test implementation therefore treats these as separate events.

3. Results

3.1. Test Datasets

Two datasets were used to test the performance of the partitioning algorithm, and the parallel version of CHRT; both are hydrographic datasets collected by the U.S. National Oceanographic and Atmospheric Administration as part of the U.S. national charting programme.

The first, a primary hydrographic survey in the vicinity of Woods Hole, MA, was conducted by the NOAA Ship *Whiting* in 2001 (Barnum, 2001), and consists of a total of 37.7×10^6 observations in depth ranges from 2–30 m. The second dataset is a portion of the survey conducted in Ernest Sound, AK in the vicinity of Union Point by the NOAA Ship *Fairweather* in 2009 (Baird, 2009), and consists of a total of 9.3×10^6 observations in depth ranges from 4–220 m.

Both datasets were used previously to demonstrate the development and behaviour of the CHRT algorithm, and are more fully described in Calder and Rice (2017).

3.2. Reconstruction Partitioning

To assess the performance of the partitioning algorithm itself, the data density estimates from both datasets were used to compute a partition for differing computational resource counts. The run-time efficiency of the algorithm is essential to its use: if the algorithm takes longer to compute the partition than having the problem partitioned improves the run-time of the estimation algorithm, then the effort is wasted. Figure 6 illustrates the actual and relative computational time observed on a particular computer system, and in particular the rapid increase in run-time engendered by increasing computational resource allocation (c.f. Saule et al. (2012)), as might be expected given the known complexity class of the general case. For the Woods Hole data, slightly higher run-times are observed, corresponding to the larger area being surveyed and the 8 m low-resolution cell size compared to the 32 m size used for Ernest Sound. For moderate (e.g., single workstation) resource counts, however, the actual run-time is significantly smaller than the estimator run-time, making the algorithm a pragmatic solution. (Note that the actual computational time is only illustrative, since it will vary with hardware and compiler selections.) Trade-offs between the estimator and partitioning algorithm runtime, and their implications for how to partition the problem, are considered further in Section 4.

The potential performance of the algorithm depends on the evenness with which the observations can be distributed among the segments of the partition (i.e., the degree of deviation from the nominal allocation of N/C observations). Figure 7 shows the mean deviation per segment as a function of the number of computational resources assigned, clearly showing that the percentage mismatch between actual and ideal workload per computation resource is on the order of a few percent of nominal workload. The mismatch rises with computational resources since each segment becomes smaller, making each row or column added

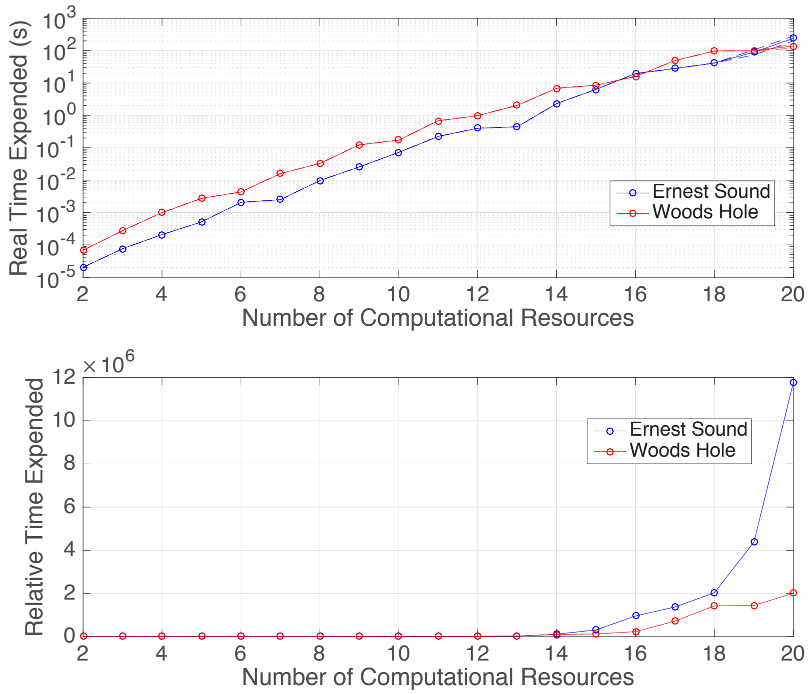


Figure 6: Estimate of absolute and relative run-time to compute a partition as a function of computational resource count. Note logarithmic scale on absolute run-time plot; dashed lines (only visible to the right of the absolute run-time plot due to scale) are 95% CI limits for $N = 100$ runs of the partitioning algorithm. Relative run-times are computed with respect to that for a computational resource count of two units.

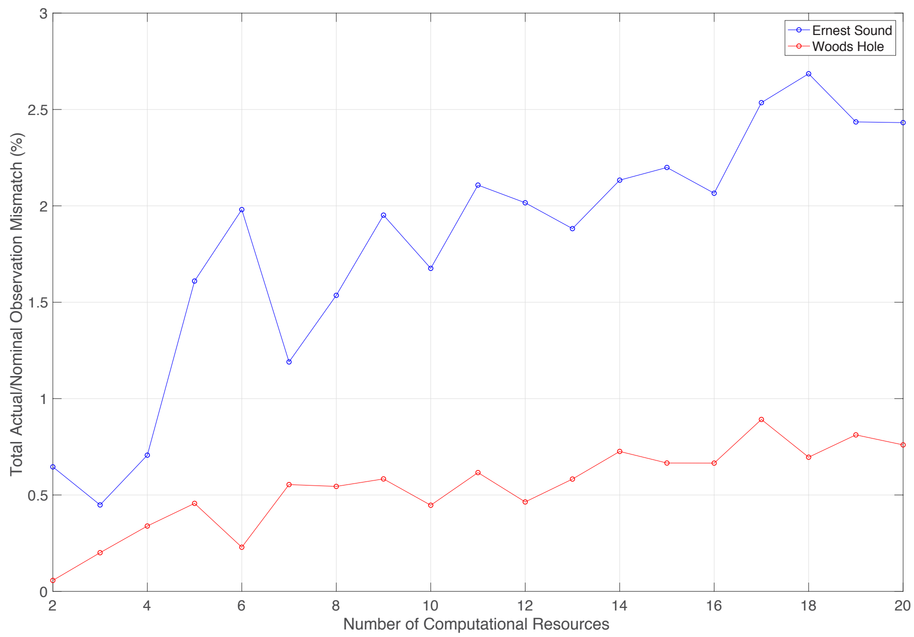


Figure 7: Percentage average deviation from ideal observation distribution per segment as a function of computational resource count.

or removed a larger (potential) percentage of the nominal workload. The difference between the two datasets is due to dataset size and geographical extent.

3.3. Speedup and Processing Rate

The two test datasets were processed using first the serial version of the algorithm, and then the parallel version, repeating the process 100 times in each case in order to gather statistics on variability. The test hardware having a quad-core, hyper-threaded processor, a range of 2–8 computational elements were considered.

The speedup achieved for the multi-threaded version of the algorithm is shown in Figure 8, and the efficiency (also known as strong scaling (Barnes, 2016)) is shown in Figure 9. The algorithm demonstrates almost perfect (and very slightly super-linear) speedup for 2–3 computational elements, but then starts to diverge from ideal speedup as the effects of cache, memory bandwidth, and I/O contention start to take effect; the corresponding efficiency shows the equivalent relatively gentle decline with additional computational resource. Note, however, that performance does not decrease as further resources are added. An overall speedup of about 4.1 times is achieved, which is perhaps not surprising on a single quad-core (albeit hyper-threaded) processor.

The overall computation rate per thread is shown in Figure 10. Clearly, the theoretical observation processing rate for each thread is constant; the ap-

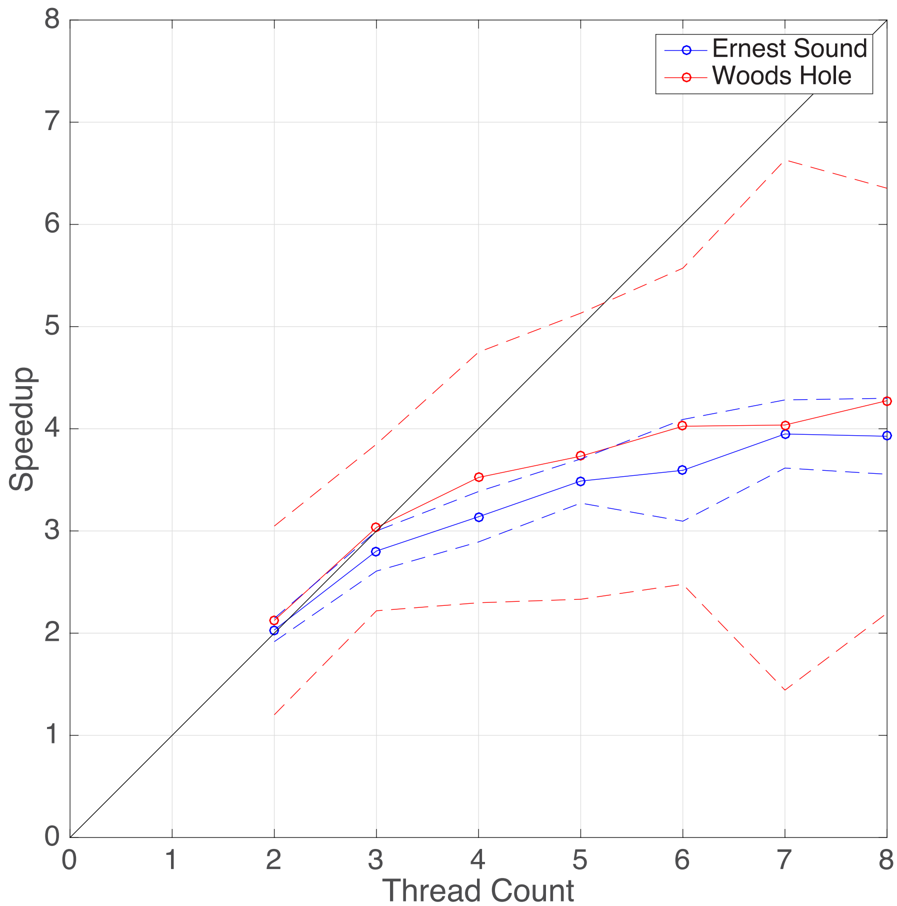


Figure 8: Speedup achieved by the algorithm with 2–8 threads on a single, quad-core, hyper-threaded CPU. Dashed lines indicate 95% CI limits.

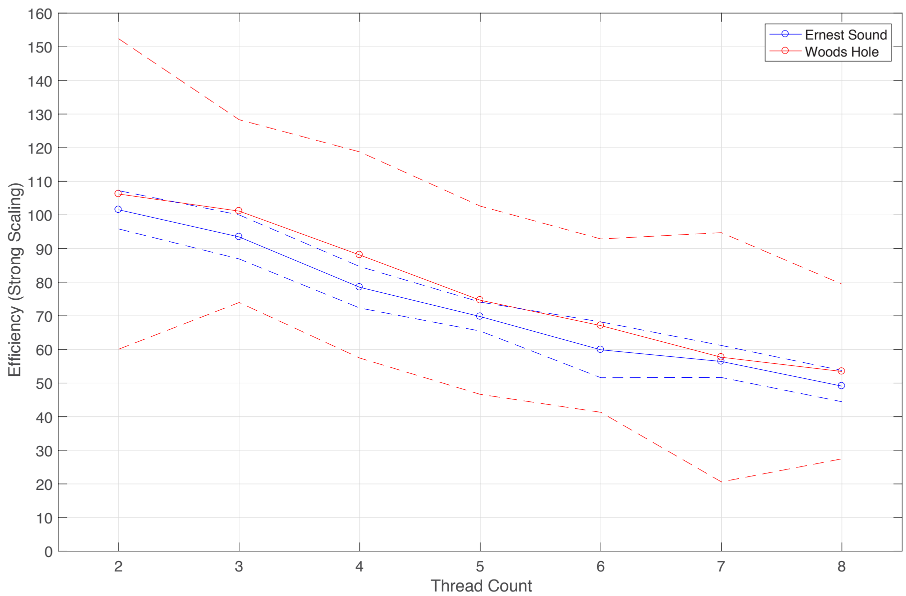


Figure 9: Efficiency of computation (i.e., speedup per computational resource committed to the task) corresponding to Figure 8. Dashed lines indicate 95% CI limits.

parent processing rate, however, drops as the number of threads increases and contention for resources takes effect. For small numbers of threads, the additional threads lead to sufficient improvement to compensate for the reduction in apparent processing rate; for larger number of threads, the resource contention overwhelms the benefit of extra threads, leading to reduced speed improvements.

The distribution of observations at the threads is given in Figure 11. A significant difference is observed between the two datasets due to the differences in bathymetry in the regions, and the type of echosounder used during the surveys. The more even distribution achieved with the Woods Hole dataset is one reason for the slightly improved speed-up observed. Analysis of the observation counts recorded at the threads indicates that the over-computation (i.e., the observations that need to be redundantly included in the partial computations so that no edge effects are engendered) average over all of the segments in the partition to approximately 2% of the total number of observations.

4. Discussion

The multi-threaded implementation of the CHRT algorithm clearly improves on the overall run-time for the algorithm, being limited on a single processor by factors other than the CPU bound of the algorithm. That is, the algorithm could theoretically complete faster if higher memory and disc bandwidth,

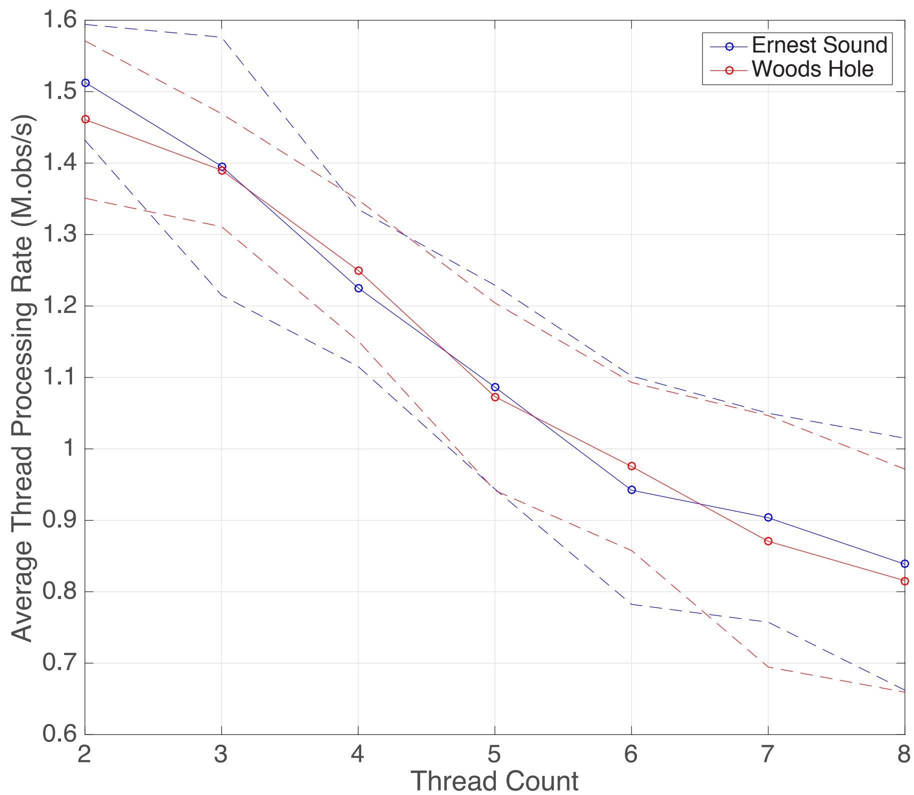


Figure 10: Processing rate per thread (in millions of observations per second processed) for 2–8 threads on a single, quad-core, hyper-threaded CPU. Dashed lines indicate 95% CI limits.

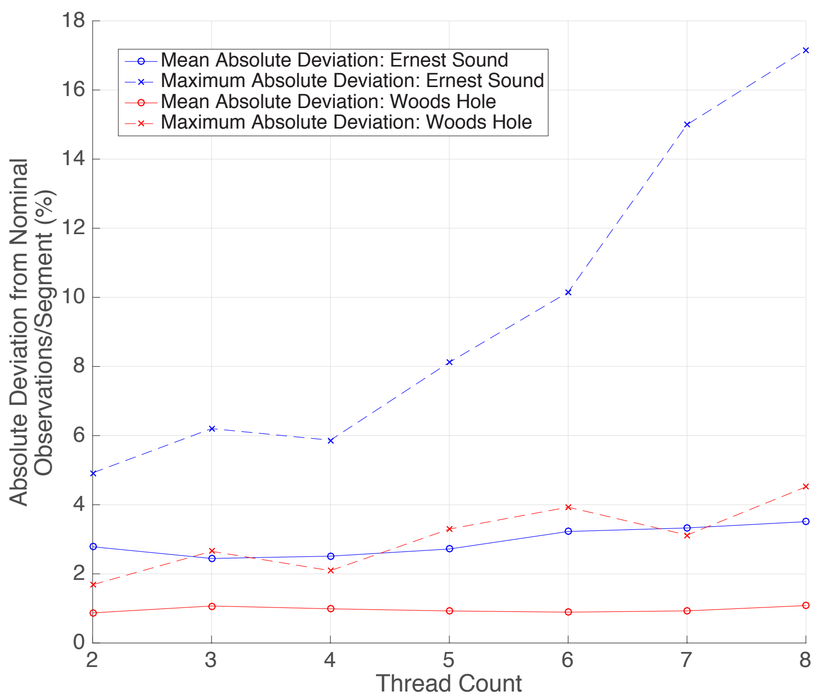


Figure 11: Maximum and mean absolute deviation of processed observations per partition segment from nominal "even" division as a percentage of nominal observation count. Non-zero deviations cause non-uniform thread run-times and hence lower overall efficiency.

and/or larger caches were available. Compressing the virtual tiles that act as intermediate results before serialization (Barnes, 2016) or the addition of solid state disc buffers (Barnes, 2017) might also improve the situation. On a single processor, however, there is a limit to achievable performance improvement, which suggests that it might be advantageous to distribute the algorithm over more nodes in order to achieve greater speedups, a topic of current research.

An increase in the number of computational resources committed to the algorithm has implications for the overall efficiency of the algorithm, and may not always be advantageous. That is, although increasing numbers of computational resources on distributed nodes will reduce the estimation algorithm's run-time, it remains an open question whether the gain will be sufficient to offset the partition run-time costs for larger computational resource counts. This in turn suggests that it might make sense to allow for a logical grouping of computational resources (i.e., making sub-clusters), partitioning over the groups at the global level, and then sub-dividing the assigned segment locally within the group either equally, or through an iteration of the partitioning algorithm applied to the assigned segment; this is similar in spirit to the "hybrid" solution of Saule et al. (2012). This would minimise the run-time for the partitioning algorithm (the more so because the local sub-division could be computed in parallel), although it would result in a globally sub-optimal partition. The difference between the performance of a locally optimal but globally sub-optimal solution and the globally optimal partition when all effects are taken into account is not obvious, and would require further investigation.

The results demonstrate that the degree of even distribution of observations between segments depends on the problem itself, although the mean performance is within 1-2% of nominal for the two (very different) datasets tested. Absolutely even distribution is likely impossible without further complexity in the partitioning algorithm to allow for non-rectangular segments. This might not be beneficial, however. Due to the CHRT algorithm's use of observations surrounding each segment to ensure that there are no edge effects, a longer perimeter, such as could be generated with non-rectangular segments, would lead to more observations being drawn into a segment. This extra computation is redundant in the sense that more than one computational resource will have to do the same base computations for the observation. Although current evidence is that this is a small effect (approximately 2% of the overall load on each thread), increased numbers of computational resources (resulting in smaller segments with higher perimeter to area ratios) and non-rectangular segments could potentially increase it to a significant degree.

Consequently, it seems likely that further improvements to the algorithm do not necessarily pertain to larger numbers of computational resources. Multi-threading of the algorithm as applied to a single segment, for example by having one thread set up the data at each low-resolution cell while one or more threads do the processing within cells, might be a productive line of investigation.

5. Conclusions

The time taken to compute a bathymetric (or other scalar field) reconstruction from raw observations is critical for practical data processing methods; acceleration of the computation can also be an enabler for more advanced algorithms built on the base computation.

The results here demonstrate that it is possible to efficiently pre-partition the computational task for a bathymetric reconstruction algorithm (in this case CHRT) into a fixed number of segments, each of which has approximately the same amount of computational effort. This allows the computation to proceed without further communication between computational units, avoiding communication or synchronisation overhead. Partition times of order 10-100 milliseconds are observed for small numbers of computational resources, along with mean absolute deviations from even distribution of effort on order 1-2%.

The resulting multi-threaded demonstration implementation of a parallel CHRT, for use on a single, quad-core CPU, is observed to achieve maximum speed-up of 4.1 on eight threads, with the sub-linear performance being driven by cache, memory, and disc contention between the threads. Nominal processing rates of up to 1.5×10^6 observations per second per thread are observed.

Examination of algorithm behaviour (partition computation rate, redundant but necessary computations, and observation count balance) with increasing numbers of computational resources indicate that it might be fruitful to examine either distribution over multiple compute nodes, or multi-threading the core algorithm to further improve the performance of the algorithm.

Acknowledgements

The support of NOAA grants NA10NOS4000073 and NA15NOS4000200 for this work is gratefully acknowledged.

Computer Code Availability

An example implementation of the algorithm, written in C++11, is available at `https://github.com/brian-r-calder/density-partition.git`, using the GNU GPL, version 2. The code was written to be portable, and therefore should require only a C++11 compiler for use; it was developed primarily on macOS, but has also been tested on both Windows and Linux platforms. Further details on compilation are provided in the source distribution. Example input data density files, and expected output, are also provided. An example implementation of a one-dimensional version of the CHRT algorithm was published to accompany Calder and Rice (2017), and can be found at `https://github.com/brian-r-calder/vr-grid-estimator.git`. The corresponding author may be contacted for further details.

References

Aspvall, B., Halldórsson, M. M., Manner, F., 2001. Approximations for the general block distribution of a matrix. Theoretical Computer Science 262, 145–160.

Baird, D. D., 2009. Descriptive report: Hydrographic survey H11825 (Ernest Sound, AK). Tech. rep., National Oceanic and Atmospheric Administation.

Barnes, R., 2016. Parallel Priority-Flood depression filling for trillion cell digital elevation models on desktops or clusters. Computers and Geosciences 96, 56–68.

Barnes, R., 2017. Parallel non-divergent flow accumulation for trillion cell digital elevation models on desktops and clusters. Environmental Modelling and Software 92, 202–212.

Barnum, S. R., 2001. Descriptive report: Hydrographic survey H11077 (Woods Hole, MA). Tech. rep., National Oceanic and Atmospheric Administation, 1315 East West Highway, Silver Spring, MD.

Berger, M. J., Bokhari, S. H., 1987. A partitioning strategy for nonuniform problems on multiprocessors. IEEE Trans. Comp. C-36 (5), 570–580.

Berman, P., DasGupta, B., Muthukrishnan, S., Ramaswami, S., 2001. Efficient approximation algorithms for tiling and packing problems with rectangles. J. Algorithms 41, 443–470.

Calder, B. R., Mayer, L. A., 2003. Automatic processing of high-rate, high-density multibeam echosounder data. Geochem., Geophys. and Geosystems (G3) DID 10.1029/2002GC000486 4 (6).

Calder, B. R., Plumlee, M. D., 2017. On testing of complex hydrographic data processing algorithms. In: Proc. U.S. Hydrographic Conference 2017. The Hydrographic Society of America, Galveston, TX, pp. 1–6.

Calder, B. R., Rice, G., 2017. Computationally efficient variable resolution depth estimation. Computers and Geosciences 106, 49–59.

Crow, F. C., 1984. Summed-area tables for texture mapping. Proc. ACM SIGGRAPH 18 (3), 207–212.

Debese, N., Moitié, R., Seube, N., 2012. Multibeam echosounder data cleaning through a hierarchic adapative and robust local surfacing. Computers and Geosciences 46, 330–339.

Gamma, E., Helm, R., Johnson, R., Vissides, J., 1994. Design Patterns. Addison-Wesley Professional.

Grigni, M., Manne, F., 1996. On the complexity of the generalized block distribution. In: Proc. International Workshop of Parallel Algorithms for Irregularly Structured Problems. Springer, Berlin Heidelberg, pp. 319–326.

Hofierka, J., Lacko, M., Zubal, S., 2017. Parallelization of interpolation, solar radiation and water flow simulation modules in GRASS GIS using OpenMP. Computers and Geosciences 107, 20–27.

Isenburg, M., Liu, Y., Shewchuck, J., Snoeyink, J., Thirion, T., 2006. Generating raster DEM from mass points via TIN streaming. In: M. Raubal et al. (Ed.), Geographic Information Science (Lecture Notes in Computer Science). Vol. 4197. Springer-Verlag, pp. 186–198.

Khanna, S., Muthukrishnan, S., Paterson, M., 1998. On approximating rectangle tiling and packing. In: Proc. ACM Symp. on Discrete Algorithms. Assoc. Comp. Machinery, pp. 384–393.

Krishnan, S., Baru, C., Crosby, C., Nov. 2010. Evaluation of MapReduce for gridding lidar data. In: Qiu, J., Zhao, G., Rong, C. (Eds.), Proc. Second IEEE International Conference on Cloud Computing Technology and Science. pp. 33–40.

Ladner, R. E., Fischer, M. J., 1980. Parallel prefix computation. J. Assoc. Comp. Mach. 27 (4), 831–838.

Land, A. H., Doig, A. G., 1960. An automatic method of solving discrete programming problems. Econometrica 28 (3), 497–520.

Lorys, K., Paluch, K. E., 2003. New approximation algorithm for RTILE problem. Theoretical Computer Science 303, 517–537.

Manne, F., Sørevik, T., 1996. Partitioning an array onto a mesh of processors. In: Waśniewski, J., Dongarra, J., Madsen, K., Olesen, D. (Eds.), Proc. International Workshop of Applied Parallel Computing. No. 1184 in Lecture Notes in Computer Science. Springer, Berlin Heidelberg, pp. 467–477.

Muthukrishnan, S., Suel, T., 2005. Approximation algorithms for array partitioning problems. J. Algorithms 54, 85–104.

Nichol, D. M., 1991. Rectilinear partitioning of irregular data parallel computations. Tech. Rep. NASA-CR-187601, NASA.

Qin, C.-Z., Zhan, L., 2012. Parallelizing flow-accumulation calulations on graphics processing units–from iterative DEM preprocessing to recursive multiple-flow-direction algorithm. Computers and Geosciences 43, 7–16.

Saule, E., Bas, E. Ö., Çatalyürek, Ü. V., 2012. Load-balancing spatially located computations using rectangular partitions. J. Parallel Distib. Comput. 72, 1201–1214.

Smith, W. H. F., Wessel, P., 1990. Gridding with continuous curvature splines in tension. Geophysics 55 (3), 293–305.

Tesfa, T. K., Tarboton, D. G., Watson, D. W., Schreuders, K. A. T., Baker, M. E., Wallace, R. M., 2011. Extraction of hydrological proximity measures using DEMs using parallel processing. Environmental Modelling and Software 26, 1696–1709.

Tobler, W., 1970. A computer movie simulating urban growth in the Detroit Region. Economic Geograpy 46, 234–240.

Wilkinson, B., Allen, M., 2005. Parallel Programming, 2nd Edition. Pearson Education, Upper Sadde River, NJ 07458.

Yıldırım, A. A., Watson, D., Tarboton, D., Wallace, R. M., 2015. A virtual tile approach to raster-based calculations of large digital elevation models in a shared-memory system. Computers and Geosciences 82, 78–88.