juliacon

Adaptive Hierarchical Regular Binning of Data Point Features

Dimitris Floros[1], Antonios Skourtis[2], Nikos P. Pitsianis[2, 3], and Xiaobai Sun[3]

[1]Nicholas School of the Environment, Duke University, Durham, NC, USA
[2]Department of Electical and Computer Engineering, Aristotle University of Thessaloniki, Greece
[3]Department of Computer Science, Duke University, Durham, NC, USA

ABSTRACT

The package `AdaptiveHierarchicalRegularBinning.jl` [1], or AHRB for short, is introduced in this paper. In the base case, given a set of n point particles, or feature vectors, in a d-dimensional metric space, AHRB constructs a tree hierarchy that sorts the particles into nested bin nodes up to a cut-off level ℓ_c. The bin nodes at each level correspond to non-overlapping d-dimensional cubes of the same size, each bin containing at least one particle, each non-leaf bin containing more than p_c particles. The choice of the geometric shape and partition parameters ℓ_c and p_c serves the purpose of facilitating downstream tasks that involve accurate multi-resolution analysis of particle-particle relationships. AHRB offers additional functionalities, especially for near-neighbor extraction or far-neighbor filtering at various spatial scales. When the feature dimension is low or modest, AHRB is competitive in time and space complexities with other `Julia` packages for recursive binning of particles into nested cubes. Distinctively, AHRB is capable of accommodating higher-dimensional data sets without suffering from high-order or exponential growth in memory usage with the increase in dimension. We demonstrate the basic functionalities of AHRB and some extended ones, provide guaranteed time and space complexities, and present execution times on benchmarking datasets.

Keywords

high-dimensional feature vectors, multi-resolution data analysis, adaptive binning of nonuniform depth, variable code length, multi-scale near-neighbor search

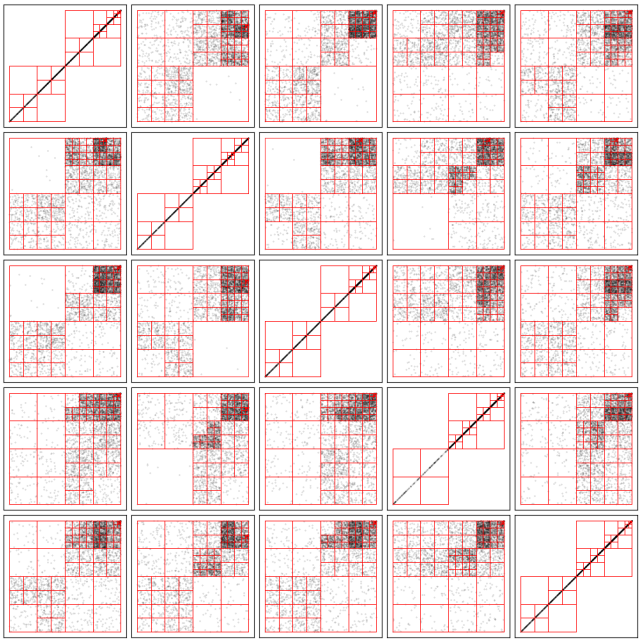

Fig. 1: AHRB is able to accommodate high-dimensional feature point datasets. The plots illustrate the AHRB structure on a dataset X of $3,672$ particle points in a 5 dimensional space. The maximal division level $\ell_{\max}$ is set to 8. Each non-diagonal plot (i, j), $i \neq j$, is the planar projection of the 5-dimensional particle cube-binning onto the plane with axes i and j, each plot (i, i) is the linear projection onto axis i, $1 \leq i, j \leq 5$. Any bin with more than p_c particles, $p_c = 100$, is further divided up to level 8. Any bin above level 8 with p_c or fewer particles is a leaf node of AHRB. Any bin at level 8 is a leaf node. All leaf bins make a partition of the dataset X. The distributions of leaf bins and non-leaf bins are shown in fig. 6.

1. Particle binning for multi-resolution data analysis

1.1 Particle binning

Particle binning is a computation task fundamental to many downstream data processing and analysis applications across various domains. Here, a *particle* refers to a spatial point or a feature vector in a multi-dimensional metric space. The feature dimension, i.e., the feature vector length, denoted by d, is not necessarily limited to low dimensions. We assume a dataset X of n particles in the same d-dimensional feature space $\mathbb{R}^d$. A particle binning process involves organizing, grouping, and indexing the particles into spatial regions, compartments, or bins by certain criteria. Typically, a particle binning process partitions the particles by recursively subdividing the finite spatial domain that tightly contains the n particles, resulting in a tree hierarchy of particle bins or bin nodes as the data structure. Each bin node on the tree has a definitive geometric shape and holds a specific number of particles. Different particle binning processes give rise to various kinds of trees [2, 4, 11, 3, 32] for particle partition and indexing, suitable for different types of subsequent computation tasks. A particle indexing tree of one kind has different characteristics from another kind, such as by the geometric shape of the bins, e.g., hyper-spheres or hyper-rectangular

[1]https://github.com/pitsianis/AdaptiveHierarchicalRegularBinning.jl

boxes, the leaf node distribution across the tree hierarchy, the population distribution across the subtrees, or other attributes.

1.2 AHRB objectives and properties

The primary goal of AHRB (pronounced as "arb", with a silent "h") is to support and facilitate multi-resolution analysis of particle-particle relations or interactions, especially for near-neighbor location or search at multiple spatial scales or for far-neighbor filtering. The bins of AHRB are thereby chosen to be d-dimensional cubes, extending the quadtree and octree to a d-dimensional hierarchical data structure.

The basic tree structure of AHRB is described as follows. At the top level, $\ell = 0$, the root cube tightly contains all n particles. For simplicity, the root cube is uniformly scaled to have a sidelength of 1. The unit cube is subdivided into subcubes of sidelength $1/2$, and the particles are binned into the subcubes. The tree nodes at level $\ell = 1$ represent the non-empty subcube bins. Any subcube bin with more than p_c particles is further divided; any bin with p_c articles or fewer becomes a leaf/terminal node at level 1. This division process continues up to a cut-off level ℓ_c or terminates sooner. All nodes at the finest level are leaf/terminal nodes. The two partition parameters p_c and ℓ_c are integers with the basic lower and upper bounds as follows,

$$1 < \ell_c = O(\log n), \quad 1 < p_c < \sqrt{n}. \tag{1}$$

The particular values of ℓ_c and p_c are determined by individual applications, and they may be refined internally by AHRB. As p_c serves as a threshold on the population of leaf bins above level ℓ_c, the recursive partition not only establishes a hierarchy for geometric resolution at multiple levels but also effectively addresses and resolves the particle sparsity or density distribution at multiple levels. In this sense, the hierarchical binning is adaptive to the spatial distribution of the particle sparsity or density. We will introduce shortly the other adaptation aspects of AHRB in response to the a priori uncertainty about the division depth, to certain dynamic changes in the particle dataset, as well as to locality improvements on specific memory hierarchy systems.

AHRB has a few distinctive properties and provides extended functionalities. Regardless of the dimensionality, the leaf bins of the AHRB data structure form a partition of the particle set X. The total number of leaf nodes is no greater than n; the total number of non-leaf nodes is no greater than n/p_c. The time complexity of AHRB construction is $O(d \cdot n)$. The space complexity is $O(d \cdot n)$. Unlike other existing Julia packages for cube-binning, AHRB does not suffer from higher-order or exponential growth in memory usage as the dimension d increases. See fig. 5. An additional development objective of AHRB is to explore and exploit the inherent potential in parallel particle binning and utilize the parallel processing architectures, in both hardware and software, that are prevalent in modern computer systems. In order to take full advantage of the AHRB structure and properties for multi-resolution data analysis, AHRB also provides efficient and valuable functions for locating neighbors in all dimensions, or selective dimensions, within various ranges.

1.3 Multi-resolution analysis in high-dimensional feature spaces

Multi-resolution analysis (MRA) is crucial to many important applications across various fields. The traditional MRA was mostly used in two-dimensional (2D) or three-dimensional (3D) ambient spaces. In modern applications, MRA has extensive and expanding use in higher dimensional feature spaces. We illustrate in fig. 2 a few areas, among numerous others, where MRA plays an important role.

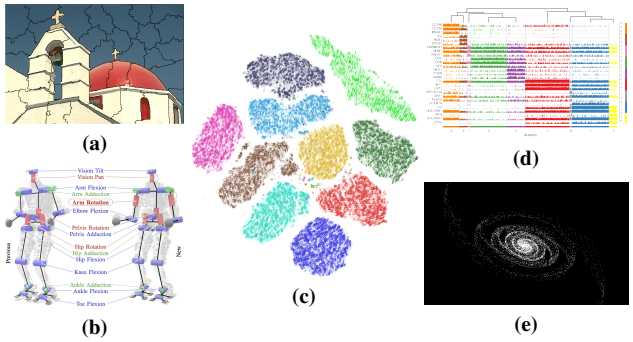

Fig. 2: A few examples of downstream tasks, in different areas of scientific study, that utilize or rely on multi-resolution analysis of multi-dimensional feature data. **(a)** image segmentation by multiple pixel features into super-pixels[9]; **(b)** robotic motion planning based on fused sensor data features [25]; **(c)** dimension reduction subject to near-neighbor preservation [24, 28, 12, 31, 30, 23, 20]; **(d)** biomarker identification or discovery from single-cell gene expressions [33][1]; **(e)** particle simulations in astrophysics [1, 18, 15].

- In image and signal processing, MRA is used in feature space for image segmentation, object recognition, noise reduction or removal, classification, or compression. See [26, 6, 19, 27, 23, 9]. For example, a 2D image of 28×28 pixels for a handwritten digit can be represented by a histogram of oriented gradients (HOG) feature of dimension 324 in fig. 2.
- In robotics planning and autonomous vehicle navigation, MRA of fused sensor data (sensor features) is used for dynamic environment mapping, path or trajectory tracking and planning, obstacle or anomaly detection in short and long ranges, collision prediction and avoidance, and decision-making for safe navigation [7, 16, 13, 25].
- MRA is valuable to biomedical data analysis [31, 33, 29]. It is commonly used in MRI, EEG, CT, fMRI, and DTI for medical image segmentation, neuronal connectivity, abnormality detection, and diagnosis. It is applied to analyze gene expression data obtained from microarray or RNA sequencing experiments to help identify patterns and biomarkers and obtain insights into disease mechanisms. It is also used in proteomics and metabolomics to explore the interactions between proteins and metabolites

These MRA applications are common in higher feature dimensions, requiring accuracy control and assurance. AHRB aims at facilitating MRA tasks in high-dimensional spaces.

1.4 Relevant Julia packages

There are several existing Julia packages for particle binning. Those based on first-order statistical approximations tend to use non-cubical bins.[2] The notable structures using cube bins for multi-resolution analysis and approximation are the quad trees and oct

[1] Image: https://scanpy-tutorials.readthedocs.io/en/latest/plotting/core.html

[2] https://github.com/JuliaGeometry/OctTrees.jl, https://github.com/rdeits/RegionTrees.jl, https://github.com/alexhad6/ParallelBarnesHut.jl,

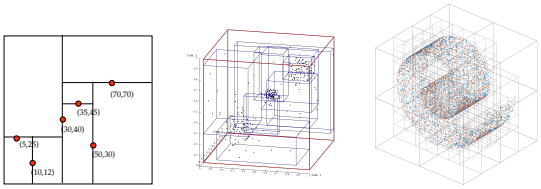

Fig. 4: Three types of tree structures with rectangle bins, besides other trees with different bin shapes, for feature point binning to facilitate various downstream applications. **Left:** kd-tree where every non-leaf node is associated with one of the d-dimensions and a pivot point for splitting, i.e., a splitting hyperplane [4]. **Middle:** R-tree with rectangle bins of variable side lengths, all leaf bins are at the same level [11, 3], similar to B-trees [2]. **Right:** Octree with cube bins, each non-leaf node has upto 8 child nodes, all cube bins at the same level are of the same size, the leaf bins are not necessarily at the same level [10].

trees, they are explicitly for 2D or 3D particles. Existing extensions of cube binning to higher-dimensional feature spaces tend to have full d-dimensional trees, perhaps, for the ease of indexing to the bins, including the empty bins. The indexing cost in memory usage grows sharply, or even exponentially, with the increase in dimension, thereby limiting the feature dimension implicitly.

2. AHRB method

In this section, we introduce the particle binning method that equips AHRB with the distinctive properties briefly described in section 1.2. There are three conceptually integral components: encoding, counting, and permuting (ECP). Each particle is mapped to a binary code. The code length can be made uniformly fixed or variable across the particles, which are the two extreme cases of the AHRB encoding scheme with variable block length. For sequential algorithm implementations, we introduce two major ways to organize the E-operations, C-operations, and P-operations. For parallel algorithm implementations, there are more options to coordinate and schedule the ECP operations. In the current version, AHRB implements specific parallel patterns with multi-thread processing using Julia's dynamic scheduler on shared-memory computers. AHRB uses the same memory allocation and management scheme for both sequential and parallel executions.

https://github.com/krcools/CollisionDetection.jl,
https://github.com/alyst/SpatialIndexing.jl

2.1 Block ECP

We describe the ECP components in detail and provide a pictorial description in fig. 3.

Encoding. By the spatial domain division, each cube node at level 1 is associated with a d-bit geospatial index, specifying the cube location with respect to the center of the unit box containing all n particles. Similarly, each cube node at level ℓ has a geospatial index with $d\ell$-bits. The dyadic division of the particle domain leads directly to the binary quantization of the particle coordinates in the feature space $\mathbb{R}^d$. Assume temporarily that the division of the spatial domain terminates at a cut-off level ℓ_c and that each coordinate of a particle is quantized to ℓ_c bits. Then, each particle $x = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$ in the data set $X \subset \mathbb{R}^d$ is mapped to a $d \times \ell_c$ *dimension-and-depth* bit array. The leading column of the dimension-and-depth bit array for particle x is comprised of the leading bits in all dimensions, and column-ℓ composed of the ℓ-th bits in all dimensions, $\ell = 1, \cdots, \ell_c$. We refer to column ℓ as the ℓ-th codeword for particle x. We refer to the sequence of ℓ_c codewords as the Morton code of particle x [21]. See (a) in 3. The prefix with the first word of the Morton code for particle x is the geospatial index to the cube-bin node at level 1 the particle x resides. If the cube bin does not have more than p_c particles, the bin node is a leaf node. Otherwise, the prefix with the first two words of the Morton code for particle x is the geospatial index to the cube-bin node x resides at level $\ell = 2$, and so on and so forth. This binning methodology is essentially the radix sort by the most significant bits in all dimensions [5, 22].

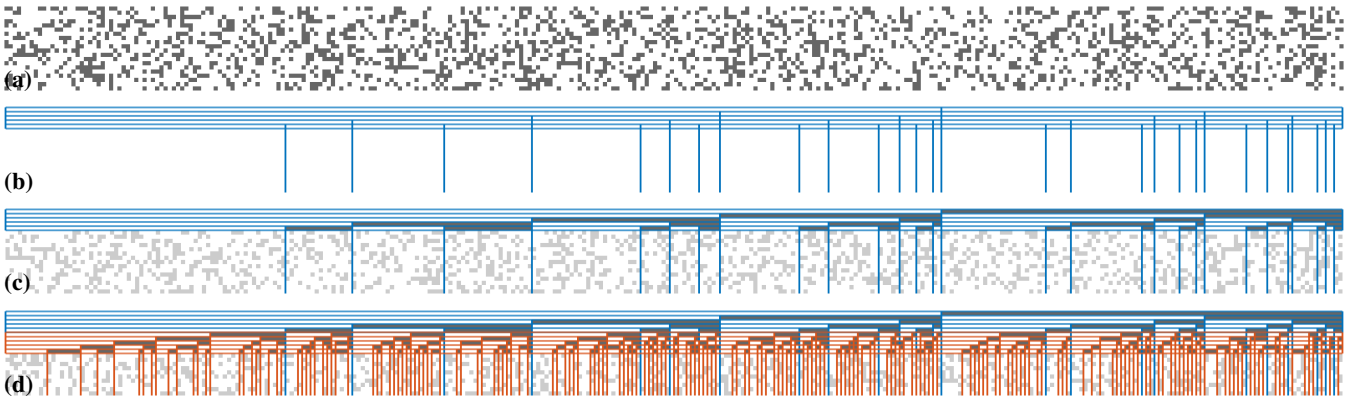

Fig. 3: Pictorial description of the ECP operations in AHRB construction. **(a)** The Morton code matrix with $n = |X|$ columns, column j is the Morten code of ℓ_c words for particle j, each word has d-bits, one bit per dimension. **(b)** The number of bins (non-empty cubes) and the bin boundaries at level 1 are determined by the particle counting process. The particles in the same bin have the same word prefix in their Morton codes. In the depiction, $d = 5$. **(c)** Particle permutation. Particles are relocated in memory to the bins. **(d)** The ECP process is applied at level 2 to each and every bin at level 1, and recursively applied to the bins at the subsequent finer levels. REMARKS. The subplots also depict the dependency of the bin division at level $\ell > 1$ on the previous level and the (non-uniform) concurrency in the ECP options among the non-empty bins at each level.

To eliminate the problem with rapid growth in memory usage with the increase in dimension, *the* AHRB *structure does not keep or register any empty cubes.* It provides instead economical mappings between the geospatial indices and the (memory) address indices for the tree nodes. The change guarantees that the memory space requirement is well bounded by a small multiple of n memory units, regardless of the dimension. This fundamentally enables AHRB to accommodate high-dimensional datasets and sets AHRB apart from other existing hierarchical structures for cube-binning.

Counting. At each division level ℓ, the number of particles with the same geospatial prefix is counted. By the counting, a cube at level ℓ is either empty or non-empty. Only non-empty cubes are registered as the AHRB tree nodes. Any non-empty cube is categorized as either sparse if it has at most p_c particles or dense if it has more than p_c particles. Any dense cube undergoes further division until reaching level ℓ_c. The total amount of memory space, denoted as $\mathcal{M}$, for all tree nodes is known by the counting. We allocate $(1+\beta)\mathcal{M}$ memory space at once, where β is a non-negative prescribed number. When $\beta = 0$, the memory space is tight for the current data set. When $\beta > 0$, we allocate $\beta\mathcal{M}$ additional buffer space in order to accommodate new particles into the same bins. The rate of such changes is anticipated by the parameter β. Once the memory is allocated, the boundaries for the leaf nodes, and non-leaf nodes, can be easily determined by the bin counts and the buffering factor $(1+\beta)$. By this memory layout scheme, all nodes at any subtree are located in a consecutive block of the allocated memory space. The mapping from the geospatial indices of the tree nodes and to the memory address indices is thus established, and so is the reverse mapping. Each mapping takes only a vector of length equal to the total number of tree nodes, no greater than $(1+1/p_c)n$. We depict in (b) of fig. 3 the counting at level 1 and show the bin boundaries at the bottom line with $d = 5$ and $\beta = 0$.

Permuting. AHRB relocates the particles from their ordering given at input to the bin locations in the memory space. In terms of functionality, AHRB not only provides the particle index mapping but also renders the particles in the bin order, such that the particles in any subtree are in a consecutive memory block. We depict in (c) of fig. 3 the particle permutation at level 1. The process of counting and permuting is equivalent to the counting sort [17].

Algorithm 1: AHRB_ECP
(the basic ECP module for AHRB construction)

Input: X, d, n // X is a set of n particles in $\mathbb{R}^d$
  ℓ_c, p_c // level cutoff $\ell_c \geq 1$; population cutoff $p_c > 1$
Output: $T = \text{AHRB\_ECP}(X, \ell_c, p_c)$
  // AHRB tree with upto ℓ_c levels of bin-nodes below the root
$Q \leftarrow \text{ENCODING}(X, \ell_c)$
T.bin_boundaries $\leftarrow \text{COUNTING}(Q)$
T.leafbin_particles $\leftarrow \text{PERMUTING}(X, T.\text{bin\_boundaries})$

We are in a position to describe *Block ECP.* The ECP operations described above can proceed at different operational granularity settings. At the one extreme, $\ell_c = \ell_{\max}$, where $\ell_{\max}$ is prescibed by the user. At the other extreme case, one can set $\ell_c = 1$, one level at a time, and apply the same process to the particle set in each of the dense bins recursively. The particle set in a (sparse) leaf bin node at level 1 is no further partitioned. The effective Morton code for a leaf bin has one word in d bits, one bit per dimension. Similarly, the particle set in a leaf bin node at level 2 has the effective 2-words Morton codes, and so on. This results in variable code length in the unit of words, ranging from 1 word to $\ell_{\max}$ words. Such variable-code-length encoding scheme adapts to and reflects the spatial variation of the particle density of dataset X in a high-dimensional feature space.

There are multiple advantages to organizing the ECP operations in block-batched options. Each block ECP batch involves ℓ_c levels using the corresponding ℓ_c words in the Morton codes, ℓ_c is an internal block parameter of AHRB, $1 \leq \ell_c \leq \ell_{\max}$. When $\ell_c > 1$, AHRB can leverage fast vector operations supported on modern computer architectures. Given the a priori uncertainty about the spatial variation in the data density/sparsity, the prescribed hyperparameter $\ell_{\max}$ may be large. It is therefore beneficial to set ℓ_c as a small number so that the encoding cost is determined primarily by the data size, data variation, and the parameter ℓ_c, eliminating the potential problem of overestimation by $\ell_{\max}$. An additional benefit of block ECP is in data locality improvement. Modern memory systems are organized in multiple cache levels, which favor algorithms with spatial and temporal localities at multiple levels in data access and movements. In [8], it is shown that for data permutation alone, permutation at multiple block stages allows one to achieve better data locality and time performance on any specific memory system, without being confined to permuting all data at once or permuting the data at an unnecessarily finer granularity. The concept and technique of block data permutation are seamlessly incoorporated in AHRB.

In summary, the block ECP method encompasses all cases with $1 \leq \ell_c \leq \ell_{\max}$, enabling adaptation to the sparse patterns in datasets, as well as to vector operations and case hierarchies on modern computer systems. We outline in algorithm 1 and algorithm 2 the basic ECP module and the block ECP algorithm, respectively. For simplicity and clarity, we treat ℓ_c as a constant in algorithm 2. The tree depth increases by up to ℓ_c levels at each block ECP step, and the code lengths for the new bin nodes increase proportionally. The memory requirement increases as well for the new nodes. We use a pair of ping-pong buffers for memory management in adaption to the tree growth. We have omitted such nuance details in the algorithm outlines.

Algorithm 2: AHRB_BECP for AHRB construction

Input: X, d, n // X is a set of n particles in $\mathbb{R}^d$
  $\ell_{\max}$, p_c // maximum level $\ell_{\max} > 1$; population cutoff $p_c > 1$
Output: $T = \text{AHRB\_BEPC}(X, \ell_{\max}, p_c)$
  // AHRB tree with up tp $\ell_{\max}$ levels of bin-nodes below the root
Initialization: $T \leftarrow \text{root\_bin}(X)$ // the root bin at level 0
Internal parameter: ℓ_c, $1 \leq \ell_c \leq \ell_{\max}$
for $\ell = 0 : \ell_c : \ell_{\max} - 1$ **do**
  for each dense bin_j at level ℓ of T **do**
    $T.\text{bin}(\ell, j) \leftarrow \text{AHRB\_EPS}(X(\text{bin}_j), \ell_c, p_c)$
  end
end

2.2 Empirical observation & evaluation

We present empirical observation and evaluation of the AHRB method in several figures. (a) In fig. 1 we illustrate how AHRB can accommodate datasets in higher dimensional feature spaces. (b) In fig. 5, we demonstrate how AHRB effortlessly surpasses dimension

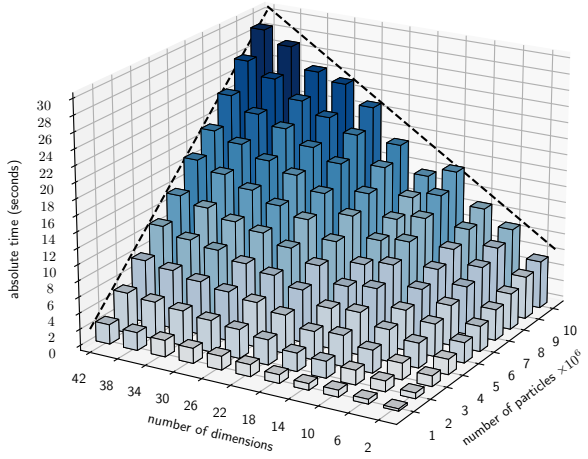

Fig. 5: The sequential execution times of a single block of ECP operations with block size $\ell_c = 3$. The time plot reflects the time complexity of algorithm 1 over the range of data size n (the number of particles) from 1 million to 10 millions and the range of d (the dimensionality) from 2 to 42. The execution times are measured by seconds on the wall clock, displayed along the z-axis. The experiment is carried out on an `Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz`. **Observation.** The execution time of the ECP operations scales linearly with dataset size n and linearly with dimension d.

40, where other cube-binning hierarchies face challenges or fail to proceed. We observed that the execution time of algorithm 1 scales linearly with the dimension d and linearly with the dataset size n, consistent with our complexity analysis. (c) In fig. 6, AHRB reveals the spatial variation in particle density/sparsity in terms of leaf-bin distribution and non-leaf-bin distribution across all levels of the AHRB tree. These distributions are unknown a priori. (d) It is illustrated in fig. 7 that the block ECP with variable code length outperforms the ECP with the fixed uniform code length on a dataset with non-uniformly distributed particles. Even on datasets with uniformly distributed particles, the former tends to terminate earlier and faster than what is expected based on a conservative estimation.

2.3 Parallel scheduling

In the aspect of ECP operations in parallel, the current version of AHRB utilizes multi-threads on shared-memory computer systems. There are different ways to explore and exploit the concurrency in the ECP operations, subject to the dependency from the top and coarse level to lower and finer levels. One must also take into account parallel execution overhead.

For ECP operations in each block batch, there is concurrency among the dense bins, as specified in line 8 of algorithm 2 and depicted in fig. 3. The E-C-P modules in algorithm 1 may be invoked sequentially, exploring the concurrency within each module. Alternatively, one may interleave the E-C-P operations, exploiting their common concurrency patterns. An earlier version of AHRB used the latter, while the current version deploys the former strategy. The modular structure makes performance tuning simple and provides better readability for program maintenance and translation.

The parallel scheduling of AHRB is inherently dynamic due to the non-uniform variation of density-sparsity patterns. We have established internal parameters to control thread spawning. The dynamic scheduler of `Julia v1.8+` plays an instrumental role in the parallel version of AHRB.

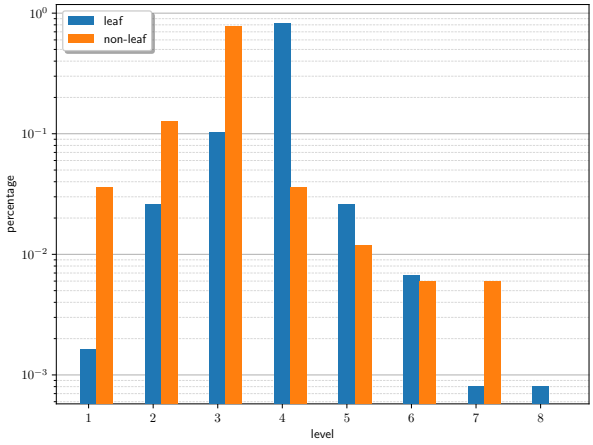

Fig. 6: The distribution of leaf nodes (in blue) and the distribution of non-leaf nodes (in orange) across multiple levels of the AHRB structure are typically non-uniform. The distribution is shown on a logarithmic scale. For the illustrated case, particles are in a 5-dimensional space, they are divided up to $\ell_{\max}$ levels, $\ell_{\max} = 8$, the same as in fig. 1. The number of leaf cube-bins is 4,914, and the number of non-leaf cube-bins is 167. The leaf-node distribution in blue shows the percentage of the leaf nodes at each level ℓ among all leaf nodes, $1 \le \ell \le 8$. The non-leaf-node distribution in red shows the percentages similarly. The majority of Morton codes for the particles have 4 or fewer effective words instead of 8 words.

3. Use case demos

3.1 User-defined node annotation

AHRB supports user-defined node annotation in order to facilitate and streamline the integration of the AHRB structure and

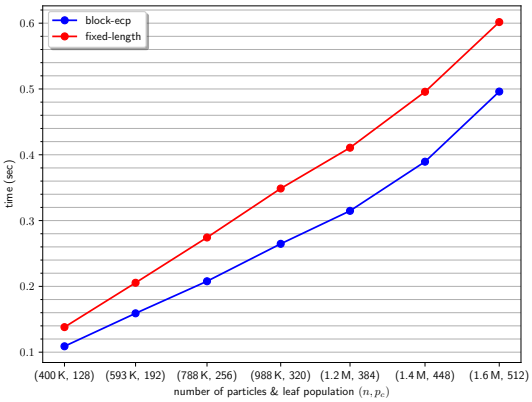

Fig. 7: Among other advantages, the block ECP of variable code length (in blue) is faster than the ECP of uniform code length (in red). In the set of experiments for this demonstration, the datasets are in a 5-dimensional space, and the number of data points ranges from 400 K(K $= 2^{10}$) to 1.6 M (M=2^{20}). The leaf and non-leaf box distributions follow those in figure fig. 6. The population cutoff parameter p_c is a small portion of n and varies from 128 for small datasets to 512 for large datasets. The maximal cutoff level is $\ell_{\max} = 8$, and the internal block parameter ℓ_c for the block version is $\ell_c = 4$. The experiment is carried out on an `Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz`. **Remark.** In two block steps, the block-ECP version of adaptive nonuniform code depth already outperforms the version with fixed uniform code length by about 1.3 times.

application-specific downstream information or structures. This feature enables efficient and stable type inference and data alignment in memory at precompilation. We demonstrate this capability of AHRB in snippet 3.1. We show how to annotate the AHRB nodes recursively with the simple example of calculating and saving the mass and mass center of the particles in each tree node. In the snippet, the functions `setcontext!` and `getcontext` are accessor functions provided by AHRB. In the supplementary material,[3] we demonstrate a dynamic particle simulation via this simple integration interface.

Snippet 3.1: User-defined node annotation

```
getmass(node::SpatialTree) = getcontext(node)[:mass]
getcom(node::SpatialTree)  = getcontext(node)[:com]

function populate_tree_ctx!(tree, dim)
  foreach(PostOrderDFS(tree)) do node
    com = zeros(dim); mass = 0.0;
    if isleaf(node)
      vm = masses[ tree.info.perm[range(node)] ]
      com  = points(node) * vm
      mass = sum( vm )
    else
      for child in children(node)
        com .+= getcom(child) .* getmass(child)
        mass += getmass(child)
      end
    end
    com ./= mass
    setcontext!(node, (; com = com, mass = mass))
  end
end
```

3.2 Neighbor bins across multiple levels

AHRB provides a function for efficient locating and rendering of all leaf bins that are within or intersect with a specified range ρ from a query point. The leaf bins are not necessarily at the same spatial resolution levels. In the extreme case, the range is large enough to cover all particles, all leaf bins are rendered. In addition, the leaf bins are rendered in an interaction list ordered from closer distance to farther distance. Such range search is fundamental to multi-resolution interaction analysis. In the special case that the range search is for the search of the k-nearsest neighbors, the range search can be terminated as soon as the k-nearest neighbors are located. In the snippet 3.2, the predicate function `filterout` returns `true` if the target box is beyond the ρ-range of the query.

[3] https://github.com/pitsianis/AdaptiveHierarchicalRegularBinning.jl/examples/barneshut.jl

Snippet 3.2: Leaf-bins within a query range

```
# ahrb construction, with additional node context
tree = ahrb(X, maxL, maxP;
  ctxtype = Vector{Tuple{Int64,Float64}})

# initialize the bin-node context
foreach(PreOrderDFS(tree)) do node
  setcontext!(node, Tuple{Int64,Float64}[])
end

# annotate target leaf bins within rho-distance
processpair!(t, s) =
  push!(getcontext(t),(nindex(s), dist(t,s)))

query = first(Leaves(tree))
rho = 1 / 8        # a particular rho value

# predicate
filterout(t, s) = dist(t,s) > rho

# recursively locate target leaf bins
recursivetraversal(query, tree,
  filterout, processpair!)
```

The tree traversal by `recursivetraversal` for locating the neighbor bins within a range has minimal complexity with cube binning, according to the analysis in [14].

4. References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, December 1986. doi:10.1038/324446a0.

[2] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control - SIGFIDET '70*, page 107, Houston, Texas, 1970. ACM Press. doi:10.1145/1734663.1734671.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331. ACM, 1990.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusett, fourth edition, 2022.

[6] Yining Deng and Bangalore S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810, 2001. doi:10.1109/34.946985.

[7] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Proceedings. 1984 IEEE International Conference on Robotics and Automation*, volume 1, pages 504–512, Atlanta, GA, USA, 1984. Institute of Electrical and Electronics Engineers. doi:10.1109/ROBOT.1984.1087218.

[8] D. Floros, A.-S. Iliopoulos, N. Pitsianis, and X. Sun. Multi-level data translocation for faster processing of scattered data

on shared-memory computers. In *Workshop on Data Locality (COLOC), Euro-Par 2019*, Gottingen, Germany, 2019.

[9] D. Floros, T. Liu, N. Pitsianis, and X. Sun. Fast graph algorithms for superpixel segmentation. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, Waltham, MA, USA, September 2022. IEEE. doi:10.1109/HPEC55821.2022.9926359.

[10] H. Freeman and D. J. Meagher. Octrees: A Data Structure for Solid-Object Modeling. In H. Freeman and G. G. Pieroni, editors, *Computer Architectures for Spatially Distributed Data*, pages 249–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984.

[12] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, pages 857–864, 2002.

[13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, April 2013. doi:10.1007/s10514-012-9321-0.

[14] A.-S. Iliopoulos. *All-Near-Neighbor Search Among High-Dimensional Data via Hierarchical Sparse Graph Code Filtering*. PhD thesis, Duke University, Durham, NC, USA, 2020.

[15] J. Carrier, L. F. Greengard, and V. Rokhlin. A Fast Adaptive Multipole Algorithm for Particle Simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4):669–686, 1988.

[16] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, February 1987. doi:10.1109/JRA.1987.1087068.

[17] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, Reading, Mass, 3rd ed edition, 1997.

[18] L. F. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.

[19] Zhi Liu, Xiang Zhang, Shuhua Luo, and Olivier Le Meur. Superpixel-based spatiotemporal saliency detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(9):1522–1540, September 2014. doi:10.1109/TCSVT.2014.2308642.

[20] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, September 2020. 1802.03426.

[21] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, International Buisiness Machines Co. Ltd, Ottawa 4, Ontario, Canada, 1966.

[22] O. Obeya, E. Kahssay, E. Fan, and J. Shun. Theoretically-efficient and practical parallel in-place radix sorting. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 213–224, Phoenix AZ USA, June 2019. ACM. doi:10.1145/3323165.3323198.

[23] N. Pitsianis, A.-S. Iliopoulos, D. Floros, and X. Sun. Sgtsnepi: Swift neighbor embedding of sparse stochastic graphs. In *IEEE HPEC*, September 2019.

[24] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000. doi:10.1126/science.290.5500.2323.

[25] P. Seiwald, S.-C. Wu, F. Sygulla, T. F. C. Berninger, N.-S. Staufenberg, Moritz F. Sattler, Nicolas Neuburger, Daniel Rixen, and Federico Tombari. LOLA v1.1 – An Upgrade in Hardware and Software Design for Dynamic Multi-Contact Locomotion. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 9–16, Munich, Germany, July 2021. IEEE. doi:10.1109/HUMANOIDS47582.2021.9555790.

[26] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi:10.1109/34.868688.

[27] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2107–2115, Venice, October 2017. IEEE. doi:10.1109/ICCV.2017.230.

[28] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000. doi:10.1126/science.290.5500.2319.

[29] The Tabula Sapiens Consortium* et al. The Tabula Sapiens: A multiple-organ, single-cell transcriptomic atlas of humans. *Science*, 376(6594):eabl4896, May 2022. doi:10.1126/science.abl4896.

[30] L. van der Maaten. Accelerating t-SNE using tree-based algorithms. *JMLR*, 15(1):3221–3245, 2014.

[31] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

[32] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, January 1993. doi:10.1145/313559.313772.

[33] Grace X. Y. Zheng et al. Massively parallel digital transcriptional profiling of single cells. *Nature Communications*, 8:14049, 2017. doi:10.1038/ncomms14049.