Optimizing the Computation of N-Point Correlations on Large-Scale Astronomical Data

William B. March*, Kenneth Czechowski*, Marat Dukhan*, Thomas Benson†, Dongryeol Lee*, Andrew J. Connolly‡, Richard Vuduc*, Edmond Chow*, and Alexander G. Gray*
*School of Computational Science & Engineering
Georgia Institute of Technology, Atlanta, GA, USA
†Georgia Tech Research Institute, Atlanta, GA, USA
‡School of Physics and Astronomy
University of Washington, Seattle, WA, USA

Abstract—**The n-point correlation functions (npcf) are powerful statistics that are widely used for data analyses in astronomy and other fields. These statistics have played a crucial role in fundamental physical breakthroughs, including the discovery of dark energy. Unfortunately, directly computing the npcf at a single value requires $\mathcal{O}(N^n)$ time for N points and values of n of 2, 3, 4, or even larger. Astronomical data sets can contain billions of points, and the next generation of surveys will generate terabytes of data per night. To meet these computational demands, we present a highly-tuned npcf computation code that show an order-of-magnitude speedup over current state-of-the-art. This enables a much larger 3-point correlation computation on the galaxy distribution than was previously possible. We show a detailed performance evaluation on many different architectures.**

I. Introduction

In this paper, we study a hierarchy of powerful statistics: the *n-point correlation functions* (npcf), which constitute a fundamental and widely-used tool for detailed characterizations and analyses of spatial data. The n-point correlation functions (npcf) are analogous to moments for spatial distributions. They can provide a simple yet comprehensive description of the data and play a key role in many powerful statistical analyses.

Applications in astronomy. The n-point correlation functions are an integral tool in astronomy. They are used in the study of many fundamental questions, including the large scale structure of the galaxy distribution [22], anisotropies and fluctuations in the cosmic microwave background [34], the physics underlying the formation of clusters of galaxies [39], and the galaxy-mass bias [18]. They are widely used to compare observations to theoretical models through perturbation theory [1, 6]. They are also used to understand the results of N-body simulations and to compare them to observations.

Three point correlation functions played a crucial role in a fundamental scientific breakthrough – the collection of evidence for the existence of dark energy. This study [9] was written up as the Top Scientific Breakthrough of 2003 in Science [31]. Due to the massive potential implications for fundamental physics of the outcome, the accuracy of the n-point statistics used were a central focus in the scientific scrutiny of these results – underscoring both the centrality of n-point correlations as a tool for significant modern scientific problems, as well as the importance of their accurate estimation.

Materials science and medical imaging. The n-point correlation functions are also used in materials science to form three-dimensional models of microstructure [13] and to characterize that microstructure and relate it to macroscopic properties such as the diffusion coefficient, fluid permeability, and elastic modulus[36, 37]. The n-point correlations have also been used for medical image processing, clustering, and classification [3, 20, 27].

Generality of npcf. In addition to these important scientific applications, the n-point correlations are a completely general tool for any multivariate or spatial data analysis problem. A univariate distribution is completely defined by its moments. Similarly, the hierarchy of n-point correlation functions completely characterizes any point process [22, 28]. In order to fully understand and characterize the data, higher-order correlations are often necessary [39]. For instance, it has been shown that models with the same two-point correlation can generate drastically different structures [33].

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

Large data. Spatial data sets in the sciences are very large and rapidly growing. In astronomy, the Sloan Digital Sky Survey [30] spent eight years surveying the northern sky and collected tens of terabytes of data. The Large Synoptic Survey Telescope [14], scheduled to come online later this decade, will collect as much as 20 terabytes *per night* for ten years. The Square Kilometer Array [4] recently announced plans to construct an exascale computer for data collection and analysis. N-body simulations of large scale structure formation, such as the Virgo Consortium's Hubble Volume simulation [32], contain tens of billions of points, with future surveys growing even larger. The Dark Energy Universe Simulations consist of over 550 billion particles [26]. In order to make use of these plentiful sources of data, we must be able to efficiently and accurately compute their n-point correlations.

Computational challenge. Unfortunately, directly computing the npcf is extremely computationally expensive. For each scale, a direct computation of the npcf will require enumerating all possible n-tuples of data points. Since there are $O(N^n)$ n-tuples for N data points, this is prohibitively expensive for even modest-sized data sets and low-orders of correlation. Furthermore, the npcf is a continuous quantity. In order to understand its behavior at all the scales of interest for a given problem, we must repeat this difficult computation many times. Thus, studies of the npcf are limited in terms of the size of the data N, the order of the correlation n, and the scales being investigated. The scientific challenges discussed above use very large data sets, require at least 3-point correlations (if not higher), and must cover a variety of scales.

Parallelism. The $O(N^n)$ work in the brute-force computation can be straightforwardly parallelized. However, this approach is still not scalable due to the overwhelmingly large amount of work required. Overcoming this bottleneck requires a more efficient algorithm. Existing approaches use space-partitioning trees to reduce the total number of tuple examinations required [10, 19]. These methods can in turn be parallelized for further speedups [8].

However, even with these improvements, npcf estimation remains extremely expensive. The largest 3-point correlation estimation thus far for the distribution of galaxies used only approximately 10^5 galaxies and consumed 300,000 cpu-hours to compute [16, 17]. Higher-order correlations have been even more restricted by computational considerations. One of the few applications of the four-point correlation in the literature uses only a few thousand galaxies [7].

We address the bottleneck in npcf computation directly by using a highly-tuned implementation of the kernel computation. By addressing this restriction and leveraging previous approaches, we can show up to an order-of-magnitude speedup over the existing most efficient algorithm.

A. Contributions

We make two major contributions to our current understanding of efficient methods for calculating n-point correlations on large-scale data sets:

1) **Fastest exact npcf** (§ IV): We present a novel implementation of the kernel computation in efficient n-point correlation algorithms which delivers an order-of-magnitude speedup over current state-of-the-art methods. This approach has enabled us to perform the largest scientifically relevant 3-point correlation computation to date, with much larger data sets than the previously largest computation in a fraction of the computer time.
2) **Performance evaluation** (§ V): We implement highly tuned base-case kernels for various architectures to compare the performance. Our evaluation reveals strengths and weaknesses of each architecture and identifies microarchitectural features that could benefit npcf-like computations in future exascale systems.

In Sec. II, we review related methods for the npcf problem. We then define the npcf in Sec. III and describe the underlying computational task. We describe the tree-based algorithm in Sec. IV and our novel base-case kernels for the computational bottleneck in Sec. V. We discuss experimental results in Sec. VI. We conclude in Sec. VII.

II. Related Work

Due to the computational difficulty associated with estimating the npcf, many alternative statistics have been developed. Counts-in-cells [35] and Fourier space methods [23, 24] are commonly used for astronomical data. However, these methods are generally less powerful than the full npcf. For instance, the counts-in-cells method cannot be corrected for errors due to the edges of the sample window. Fourier transform-based methods suffer from ringing effects and suboptimal variance [33].

Approximate methods. Methods to approximate the npcf have also been developed. Some methods use a Fourier transform-based approach to approximate the counts required for npcf estimators [41]. While this approach can be fast, it introduces additional errors into the statistic due to both the

Fourier transform and the inexact nature of the counts computed. Other methods compute the approximate npcf using space-partitioning trees [40]. However, given the scientific importance of the results being investigated with the npcf and the sensitivity of the results, it is crucial to reduce all possible sources of error. Therefore, we confine our attention to exact methods for computing the npcf.

Exact methods. Since we deal exclusively with computing the exact npcf, we only compare against other methods for this task. The existing state-of-the-art methods for exact npcf estimation use multiple space-partitioning trees to overcome the $O(N^n)$ scaling [10, 19]. An efficient parallel version of this algorithm has been implemented using the Ntropy framework [8]. This is the currently fastest implementation of general n-point correlation estimation and was used to compute the largest correlations of galaxy positions to-date [16, 18].

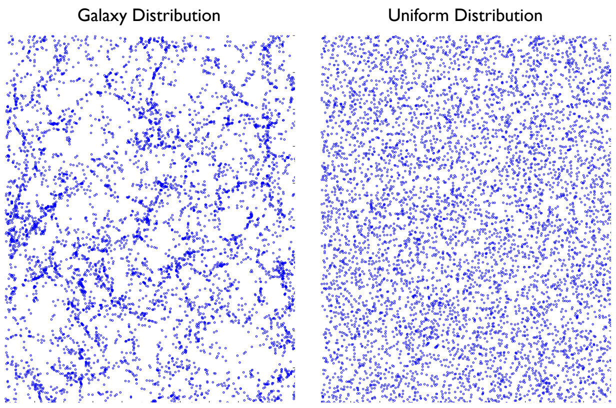


Fig. 1: A visualization of the galaxy distribution. The figure shows a 2D slice of a galaxy data set (left) compared to uniformly distributed data (right). Both images contain approximately 6,500 points.

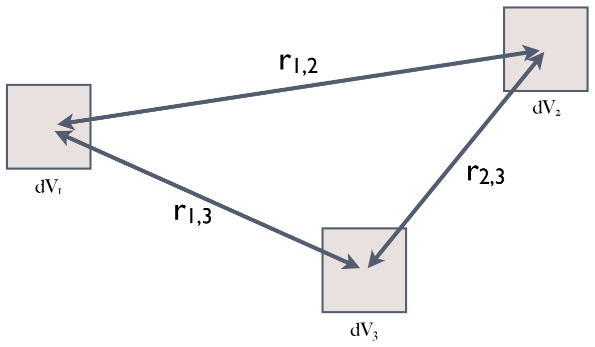


Fig. 2: A visualization of the counts required for the three-point correlation function.

III. N-Point Correlation Functions

We now define the n-point correlation functions. We provide a high-level description; a more thorough definition is available in the referenced work [22, 33]. Once we have given a simple description of the npcf, we turn to the main problem of this paper: the computational task of estimating the npcf from real data. We will show that this task boils down to a simple counting problem: the *Raw Correlation Count* problem. We show that the npcf problem consists of many instances of the computationally expensive task.

Problem setting. Our data consist of a set of points D in a subset of $\mathbb{R}^d$, drawn from a point process [29]. As we observed previously, the n-point correlation functions are analogous to the moments of a univariate distribution. The univariate mean tells us the “center” of the distribution, while the standard deviation tells us its “width”. Higher moments contain information about the symmetry of the distribution, its sharpness, and other properties. Taken together, these moments fully describe the distribution. The set of n-point correlation functions provide a similar description for point processes.

Density. Following standard practice in astronomy and many other settings, we assume the process to be homogeneous and isotropic. Therefore, the number of points in a small region of the sample window does not depend on the location of the region. We can characterize this “first-order” property with a single density parameter ρ, which is constant. The number of points in a small volume is proportional to ρ times the volume.

Two-point correlation. The assumption of homogeneity and isotropy does not require the process to lack structure. For example, the data in Fig. 1 were generated from processes with the same density, but have clearly different structure, with the galaxy data set showing much stronger clustering. The two-point correlation function captures the tendency of points in a sample to be clustered (or separated). The *two-point correlation function*, $\xi(r)$, is proportional to the increased (or decreased) probability of finding points separated by a distance r. Intuitively, $\xi(r)$ is positive if points in a data sample frequently occur at a separation of r and negative if this configuration is rare.

Three-point correlation. While the two-point correlation can roughly describe clustering in the data, higher-order correlations are needed to understand more complex structures. Higher-order correlations describe the probabilities of more than two points in a given configuration. We first consider three small volume elements, which form a triangle. (See Fig. 2). The probability of finding three points that form a triangle depends on the value of the *three-point correlation function* ζ. The function varies continuously as we change the size and shape of the triangle. While the two-point correlation ξ depends on only a single scale r, the three-point correlation ζ depends on all three pairwise distances.

Higher-order correlations. Higher-order correlation functions describe the increased (or decreased) probability of finding n-tuples of points in a specified configuration. This configuration is given by $\binom{n}{2}$ pairwise distance constraints. We refer to this set of pairwise distances as a *configuration*, or in the computational context, as a *matcher* (see Sec. III-A).

A. Computing the n-point Correlation Functions

The mean, standard deviation, and higher moments of a distribution are fundamental properties of it and do not depend on a particular sample or set of samples. If we are only given samples without knowing anything about the underlying distribution (a nearly universal setting in data analysis problems), we must estimate properties of the distribution, including the moments, from the data. Similarly, in the npcf problem, we are given a sample from a point process and must estimate the n-point correlation functions of interest from it.

For simplicity, we consider the 2-point function first. Recall that $\xi(r)$ captures the increased (or decreased) probability of finding a pair of points at a distance r over finding the pair in a uniformly distributed set. This suggests an intuitive way to compute $\xi(r)$. We generate a random set of points R from a uniform distribution filling the same volume as our data. We then count the frequency with which points appear at a distance close to r in our data set D and in the random set, denoted $DD(r)$ and $RR(r)$, respectively. We can use these counts to arrive at a value for $\xi(r)$.

Other methods with reduced variance, sensitivity to noise, and errors due to boundary effects also count pairs of points at a given separation [11, 12]. These improved estimators are simple functions of DD(r) and RR(r), along with counts DR(r) where one point is from the data and the other from the random set. Note that we must use a random set containing at least 20-30 times as many points as in our data.

Counting n-tuples. Higher-order correlation functions depend on the pairwise distances between n points, rather than a single distance as before. We will therefore need to specify $\binom{n}{2}$ distance constraints, and estimate the function for that configuration. We again generate points from a uniform distribution, and the estimator is also a function of quantities of the form $D^{(n)}$, or $D^{(i)}R^{(n-i)}$ [11, 35]. The latter refers to the number of unique triples of points, i of which are from the data and $n-i$ are from the random set, with the property that their pairwise distances lie close to the distances in the matcher.

B. The Computational Task

Note that all the estimators described above depend on the same fundamental quantities: the number of tuples of points from the data/random set that satisfy some set of distance constraints, denoted $D^{(i)}R^{(n-i)}(\mathbf{r})$. Thus, our task is to compute this number given the data D and a suitably large uniform set R. We refer to this task as the *Raw Correlation Count* problem.

Matchers. We mentioned above that we count tuples of points whose pairwise distances are "close" to the given distance constraints. Each of the $\binom{n}{2}$ pairwise distance constraints consists of a lower and upper bound: $r_{ij}^{(l)}$ and $r_{ij}^{(u)}$. In the context of our algorithms, we refer to this collection of distance constraints $\mathbf{r}$ as a *matcher*.

Given an n-tuple of points and a matcher $\mathbf{r}$, we say that the tuple *satisfies* the matcher if there exists a permutation of the points such that each pairwise distance does not violate the corresponding distance constraint in the matcher. More formally:

Definition 1. *Given an n-tuple of points $(p_1, \ldots, p_n)$ and a matcher $\mathbf{r}$, we say that the tuple* ***satisfies*** *the matcher if there exists at least one permutation σ of $\{1, \ldots, n\}$ such that*

$$r_{\sigma(i)\sigma(j)}^{(l)} < \|p_i - p_j\| < r_{\sigma(i)\sigma(j)}^{(u)} \tag{1}$$

for all indices $i, j \in \{1, \ldots, n\}$ such that $i < j$.

We can now identify the central computational task in this paper: *Raw Correlation Count*.

Definition 2. *Given a matcher $\mathbf{r}$, data set D, random set R, and integer $0 \leq i \leq n$, the task of computing $D^{(i)}R^{(n-i)}(\mathbf{r})$: the number of unique n-tuples of points, i from D, $n-i$ from R, such that the tuple satisfies the matcher is the* ***Raw Correlation Count*** *task.*

The *Raw Correlation Count* problem is the central computational bottleneck in npcf estimation.

Complete npcf task. One instance of *Raw Correlation Count* gives us a single count: $D^{(i)}R^{(n-i)}(\mathbf{r})$. The estimators referenced above consist of sums and products of these counts for values of $i = 0 \ldots, n$. Thus, in order to estimate the npcf for a single configuration, we must compute multiple instances of *Raw Correlation Count*. However, recall that the npcf is a continuous quantity. In order to understand the behavior of the npcf at a range of scales and configurations, we must repeat the *Raw Correlation Count* computation for many different matchers representing different scales. Furthermore, in many scientific analyses, jackknife resampling is used to bound variance [15] and identify variations across the sample [17]. This approach requires the calculation of the npcf on many subsets of the data. The full computational task thus requires many instances of *Raw Correlation Count*, often on the order of 1,000 times. Therefore, any increase in the efficiency of an algorithm for the *Raw Correlation Count* problem will result in significant time savings.

IV. Algorithm

We now turn to the central problem of this paper: the task of efficiently computing the *Raw Correlation Counts* from Def. 2. Clearly, these counts can be found by iterating through all unique n-tuples of points and checking if their pairwise distances satisfy the matcher. Although this algorithm can be trivially parallelized, its $O(N^n)$ scaling is prohibitively slow.

There are two clear ways to improve on this naïve algorithm: we can avoid checking some of the n-tuples of points, and we can more efficiently evaluate those tuples that we do. Much of the previous work on exactly solving the *Raw Correlation Count* problem has focused on the first approach [8, 10, 19, 40] by using space-partitioning trees to prove that many of the possible n-tuples cannot satisfy the matcher. However, to our knowledge, no one has focused on optimizing the evaluation of individual tuples. We employ a similar, tree-based approach in this work combined with optimizations for the efficient processing of those tuples that we must check. In this section, we briefly describe tree-based algorithms for the *Raw Correlation Count* problem. Then, in the next section, we discuss our improved methods for checking point tuples.

Trees. In this paper, we use both octrees and kd-trees [5, 25]. (See Fig. 3) The kd-tree [5, 25] is a binary space partitioning tree which maintains a bounding box for all the points in each node. The root consists of the entire set. Children are formed recursively by splitting the parent's bounding box along the midpoint of its largest dimension and partitioning the points on either side. In both trees, we can quickly compute the minimum and maximum distances between a pair of nodes, by using the bounding boxes in the kd-tree, or the z-order index and level for the octree. This bounding information is used in the algorithm to eliminate point tuples from consideration.

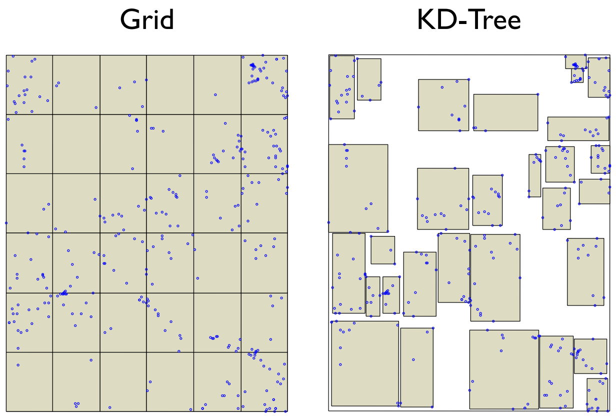


Fig. 3: A visualization of boxes in a uniform grid compared to a kd-tree.

Dual-tree algorithm. For simplicity, we begin by considering the *Raw Correlation Counts* for two-point correlation estimation. Recall that the task is to count the number of unique pairs of points that satisfy a given matcher with distance constraints $r_{\min}$ and $r_{\max}$. We consider two tree nodes at a time. We compute the maximum and minimum distances between them. If the minimum distance is larger than $r_{\max}$ or the maximum is smaller than $r_{\min}$, then we know that no pair of points from the nodes can possibly satisfy the matcher. We *prune* this pair of nodes and do no further work on them. If we cannot prune, then we split one (or both) nodes, and recursively consider the resulting pairs of nodes. When the recursion reaches leaf nodes, a base case kernel is called to compare all pairs of points exhaustively.

Multi-tree algorithm. We can extend this algorithm to the general n case. Instead of considering pairs of tree nodes, we compare an n-tuple of nodes in each step of the algorithm. This *multi-tree* algorithm uses the same basic idea – use bounding information between pairs of tree nodes to identify sets of nodes whose points cannot satisfy the matcher [10]. We require two subroutines. **TestNodeTuple** returns false if the distance bounds show that no tuples from the nodes can possibly satisfy the matcher – *i.e.* if we can prune. **BaseCase** exhaustively considers all tuples and counts the number which satisfy the matcher.

Alternative traversal patterns. We can also consider more efficient ways of traversing the tree. We give one such method in Alg. 2. Consider a matcher $\mathbf{r}$. Let $\hat{r}^{(l)} = \min r_{ij}^{(l)}$ and $\hat{r}^{(u)} = \max r_{ij}^{(u)}$. Then,

Algorithm 1 MultiTreeNpt (Tree node $T_1, \ldots,$ Tree node T_n, matcher $\mathbf{r}$)

```
    if all nodes $T_i$ are leaves then
      BaseCase($(T_1, \ldots, T_n, \mathbf{r})$)
    else if not TestNodeTuple($T_1, \ldots, T_n, \mathbf{r}$) then
      Prune
5:  else
      Let $T_i$ be the largest node
      for all children $T'$ of $T_i$ do
        MultiTreeNpt($T_1, \ldots, T', \ldots, T_n, \mathbf{r}$)
      end for
10: end if
```

Algorithm 2 InteractionListNpt (Tree root T, matcher $\mathbf{r}$)

```
    for all leaves $T_l$ of $T$ do
      Construct interaction list $L(T_l)$ for $T_l$
    end for
    for all leaves $T_l$ of $T$ do
5:    for all unique $(n-1)$-tuples of nodes $T'_1, \ldots, T'_{n-1}$ in $L(T_l)$ do
        if TestNodeTuple($T_l, T'_1 \ldots, T'_{n-1}, \mathbf{r}$) then
          BaseCase($(T_l, T'_1 \ldots, T'_{n-1}, \mathbf{r})$)
        end if
      end for
10: end for
```

we know that if a pair of nodes are separated by more than $r_{ij}^{(u)}$ (or less than $r_{ij}^{(l)}$), any tuple of nodes that contains this pair can be pruned. Using this observation, we can avoid the full tree traversal in Alg. 1. Instead, we proceed in two phases. First, for each tree leaf, we construct an interaction list containing all leaves at a distance in the range $[\hat{r}^{(l)}, \hat{r}^{(u)}]$, which can be done efficiently using existing methods. Then, for each leaf T_l, we iterate through all unique n-tuples of nodes consisting of T_l and $n-1$ nodes from its interaction list. We can then compute the base case on this node tuple.

Leaf size. These algorithms have as a tuning parameter the size of the tree leaves. Ideally, we would use leaves containing a single point. This provides the maximum opportunity to avoid unnecessary work, since we prune all possible tuples that do not satisfy the matcher. However, in practice the overhead of deeper trees and more traversal time becomes greater than the time required for an exhaustive computation at some level of the tree. The optimal leaf size then depends on the efficiency of the base case computation.

Parallelism. Note that all these approaches can be parallelized. Different parts of the tree traversal (different recursive calls) in Alg. 1 can be carried out independently. The loop over leaves in Alg. 2 can also be parallelized. As we noted previously, Alg. 1 has been parallelized in the Ntropy framework [8]. Any such parallel approach must still exhaustively compute a base case for leaves. We now turn to our optimizations for this bottleneck computation.

V. Optimizations

We focus here on the base case computation of the *Raw Correlation Counts* (defined in Sec. III-B). We restrict our attention to counts for the 3-point correlation for simplicity. We have three sets of points A, B, C. Our task is to identify all triples of points (a, b, c) where $a \in A$, $b \in B$, $c \in C$ and the triple satisfies a given matcher. A brute force algorithm for the base case calculation would simply iterate through all triples using a triply-nested loop and test each triple against the matcher. Assuming the sets of points have cardinality N, such an approach involves $O(N^3)$ distance calculations and $O(N^3)$ conditional evaluations to determine if the matcher is satisfied.

Two-step algorithm. The brute-force approach computes many redundant distances. We use a two-phase base case calculation that first computes all of the unique pairwise distances and stores the result of whether or not the distance satisfied each of the matcher constraints. Note that the distances themselves need not be stored. We then iterate through all triples and check the stored set of matcher results to determine if the matcher has been satisfied. Thus, this algorithm involves $O(N^2)$ distance calculations, $O(N^2)$ conditional checks, and $O(N^3)$ matcher satisfiability checks. We demonstrate in the following section that the $O(N^3)$ matcher satisfiability checks can be performed without the use of conditionals.

A. Bitwise interpretation of Raw Correlation Counts

We now turn to an approach to count the number of triples satisfying a matcher using only bitwise logical operations and population counts. Let r_{12}^{AB} represent a K-width set of bits corresponding to pairs of points from sets A and B and indicating whether or not the matcher distance r_{12} was satisfied by each pair of points. In other words, the i^{th} bit of r_{12}^{AB} is one if the distance between the i^{th} pair is between $r_{12}^{(l)}$ and $r_{12}^{(u)}$. For three sets of points A, B, and C, there are nine such K-width values to represent the three different matcher thresholds and three different pairings of sets of points. We can then increment the total matcher count for a set of K triples by

$$\begin{aligned}
&\mathrm{POPCNT}(\\
&\quad (r_{12}^{AB} \wedge r_{23}^{AC} \wedge r_{13}^{BC}) \vee (r_{12}^{AB} \wedge r_{13}^{AC} \wedge r_{23}^{BC}) \vee \\
&\quad (r_{23}^{AB} \wedge r_{12}^{AC} \wedge r_{13}^{BC}) \vee (r_{23}^{AB} \wedge r_{13}^{AC} \wedge r_{12}^{BC}) \vee \\
&\quad (r_{13}^{AB} \wedge r_{12}^{AC} \wedge r_{23}^{BC}) \vee (r_{13}^{AB} \wedge r_{23}^{AC} \wedge r_{12}^{BC}))
\end{aligned} \tag{2}$$

where POPCNT indicates the population count (i.e., the number of bits set to one).

Each of the six parenthesized bit-field conjunctions above represents one of the possible assignments of pairwise distances to edges of the triangle given by the matcher. The conjunction of terms will be one if the matcher is fully satisfied by that arrangement of edges and distances and the disjunction of all six arrangements will thus be one if any such arrangement satisfies the matcher. Therefore, counting the set bits in the resulting K-width bit vector will count the number of triples that satisfy the matcher.

For many K-width vectors, the above computation involves only logical operations, population counts, and integer additions to accumulate the total count. No distances are calculated and no conditionals are required in the $O(N^3)$ portion of the two-phase algorithm. This significantly increases the performance of the part of the kernel that dominates the asymptotic computational complexity.

B. Base Case Merging

The interaction list tree traversal algorithm (Alg. 2) provides opportunities for further optimizations. We consider the interaction list for a leaf node A and a node in the list B. We then merge together all remaining nodes in the interaction list for A into a single "meta-node" C. Thus, we generate a set of base cases with node C as large as possible to exploit the efficient bit-field-based kernels.

As we show in Sec. VI, optimal runtimes for the entire algorithm come from leaf nodes with 10-20 points. The resulting base cases will not use all the bits of even a 32-bit register. By combining nodes into one large node, we can make use of an entire word and test many potential tuples in parallel.

C. Architecture-specific optimizations

While the bitwise interpretation of *Raw Correlation Counts* described in Section V-A is already an efficient representation for most platforms, there are several architectural features that will significantly impact ultimate performance. In particular, the optimal width K of the triples to be considered simultaneously will depend upon the register widths and supported instruction sets. In addition, population count instructions (i.e., a single instruction that returns the number of set bits in some bit-field) are particularly useful. Fortunately, such instructions are available on most modern CPUs and GPUs.

D. Acceleration on CPUs

In this section, we consider *Raw Correlation Counts* on several modern CPUs, including Intel Nehalem and Sandy Bridge processors and AMD K10 and Bulldozer processors. We also discuss optimizations for low-power systems including Intel Atom and ARM Cortex-A9 processors.

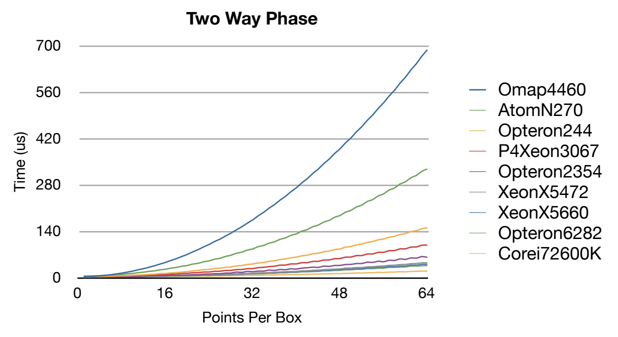
Two phases. As noted above, we split the base case computation into two phases: the calculation of all pairwise distances between points and the population counts based on the logical expression in Eqn. 2. These two phases, one consisting of floating-point instructions and the other logical operations and population counts, are largely independent and can be optimized separately. To achieve optimal performance we have implemented several kernels for each phase and selected the fastest by running on actual hardware.

Floating point phase. For phase one of the base case calculation, we have implemented kernels using scalar SSE2 instructions, 128-bit vector SSE2, SSE3, and FMA4 instructions, and 256-bit AVX instructions. With two exceptions, using the highest available instruction set delivered optimal performance. The exceptions are Intel Atom and AMD Bulldozer processors. Although Atom supports the SSE3 instruction set and Bulldozer supports the AVX instruction set, both handle the widest vector cases by either issuing multiple narrower microoperations or fusing narrower vector resources.

The ARM architecture does not support double-precision SIMD instructions, so double-precision computations are performed on a standard VFPv3 floating-point unit. However, the ARMv7-A ISA uses the floating point status register for double precision comparison results rather than writing a bitmask to a floating point register. In order to avoid transfers of comparison results to general purpose registers, we exploit the fact that the NEON SIMD unit uses the same registers as the floating-point unit and construct a comparison bitmask using available NEON instructions. NEON is a widespread SIMD extension to the ARMv7 ISA. This approach is faster than the naïve approach of transfers from the status registers to general-purpose registers. Figure 4 illustrates the performance of the floating-point intensive phase of the base case calculation on the CPUs listed in Table I.

Processor	Clock	ISA	Microarchitecture
OMAP 4460	1.2 GHz	ARMv7	Cortex-A9
Atom N270	1.6 GHz	x86	Bonnell
Opteron 244	1.8 GHz	x86-64	K8
P4 Xeon 3067	3.1 GHz	x86	NetBurst
Opteron 2354	2.2 GHz	x86-64	K10
Xeon X5472	3.0 GHz	x86-64	Harpertown
Opteron 6282	2.6 GHz	x86-64	Bulldozer
Xeon X5660	2.8 GHz	x86-64	Nehalem
Core i7 2600K	3.4 GHz	x86-64	Sandy Bridge

TABLE I: Processors used in the experiments shown in Fig. 4.



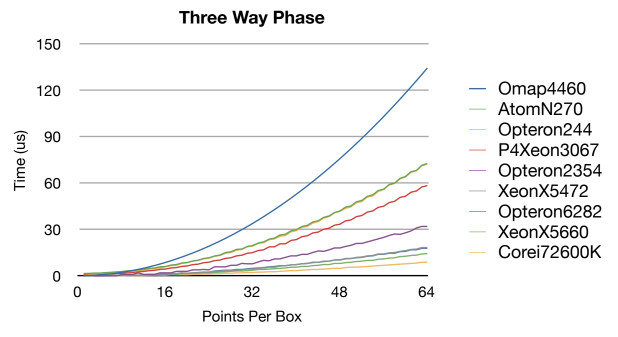


Fig. 4: Execution times for the first phase (two way) and second phase (three way) of the base case calculations. Reported times correspond to the minimum time over 100,000 kernel executions. All kernels were compiled with gcc 4.6.3 using architecture-specific optimization flags.

Logical operations phase. For the second phase of the base case computation, we leverage 64-bit instructions available on modern CPUs and vector representations of the bit-fields to achieve optimal performance. We restrict the sizes of sets A, B, and C to be at most 64. This restriction is acceptable for our tree-based implementation because very few leaf nodes have more than 64 points in an optimal tree (see Sec. VI). We handle base cases with nodes containing more than 64 points by splitting the nodes and making multiple smaller kernel calls. Thus, we can reduce the phase two inner loop to a small number of logical operations on 64-bit binary vectors and a final population count computation on a 64-bit binary vector. It is worth noting that 64-bit instructions are available on all modern x86 CPUs.

POPCNT instructions. The performance of this kernel is primarily determined by the speed of the population count computation. Intel Nehalem and Sandy Bridge and AMD K10 and Bulldozer CPUs all have hardware support for this operation via a POPCNT instruction. We found that on all of these CPUs using POPCNT instructions results in optimal performance or near optimal performance. For older CPUs we tried several methods of computing population counts. First, we considered using 8-bit, 11-bit and 16-bit lookup tables. We also tried a simple divide-and-conquer implementation, its more sophisticated variant in Alg. 5.1 from [38], the algorithm from HAKMEM [2], and its variant with 4-bit fields from [38]. For the last algorithm, we also designed SIMD versions using MMX and SSE2 instructions. Additionally, we implemented SSSE3 and XOP versions of the SIMD-oriented population count algorithm suggested by W. Mula[21].

The basic ARMv7 instruction set does not contain a population count instruction, but NEON provides a SIMD instruction VCNT that performs population counts on SIMD vectors. Cortex-A9 CPUs have only 64-bit execution units, so using 128-bit instructions does not yield performance benefits. Moreover, Cortex-A9 can issue only one NEON SIMD instruction per clock cycle compared to two scalar operations. Despite these limitations, in our tests the NEON version is about 50% faster than our fastest scalar version. Figure 4 illustrates the performance of phase two of the base case computation on various CPUs.

Additionally, we can see further performance benefits by short-circuiting the loop over the third element of the tuple. When considering a tuple (a, b, c), if we find that the distance between a and b will not satisfy any constraint in the matcher, then we can stop testing possible values of c.

E. Performance factors on CPUs

We now turn to the question of what makes the compute kernels perform well on some microarchitectures and poorly on others. To determine the impact of parameters of CPU microarchitectures we tested the best kernels on 16 different x86 CPUs. Where possible, we measured both 64-bit and 32-bit versions on the same CPU. For each CPU, we call the kernels using the actual base case sizes from a computation on a kd-tree of 1.1×10^6 galaxy positions, and measure the total time spent in the floating-point and logical parts of the base case computation. Then we use linear regression to explain the logarithms of these times by a linear combination of CPU features. The CPU features we used are logarithm of frequency, logarithm of issue width (the maximum number of instructions per clock cycle), logarithm of width of floating-point execution units (64 for P4, K8, Atom; 256 for Sandy Bridge; 128 for other CPUs), logarithm of width of integer execution units (64 for P4, K8; 128 for other CPUs), an indicator for a 64-bit kernel, and an indicator for a hardware population count instruction.

The results are presented in Table II. The two columns show analyses for the floating point and logical operations parts of the computation. Each row represents one of the independent variables listed above. The values in the intersections of rows and column show how much the factor in the row header affects the variable in column header. For example, a 1% increase in CPU frequency corresponds to, on average, 0.81% reduction in time spent on floating-point part of the basecase, and 1.3% time reduction in logical operations part.

	log(FPPartTime)	log(IntPartTime)
(Intercept)	Yes	Yes
log(Frequency)	-0.8074***	-1.3000***
	(0.2269)	(0.4529)
x86-64	-0.1669***	-0.4581****
	(0.0562)	(0.1058)
Hw Popcnt		-0.2710*
		(0.1315)
log(Issue Width)	-0.4361**	-0.4280
	(0.1880)	(0.4054)
log(FP EU Width)	-0.8553****	
	(0.1063)	
log(Int EU Width)		-1.4674****
		(0.3030)
N	29	29
R^2	94.02%	88.75%
Adj R^2	93.03%	86.30%

TABLE II: Regression results for floating-point and logical parts of n-point correlation. Standard errors are in parentheses. {*, **, ***, ****} denotes significance on the {10, 5, 1, 0.1}% level.

F. *Acceleration on GPUs*

Although we do not utilize GPUs for our application, we have investigated their use and briefly include some of our analysis. The bit-field approach developed previously can also be employed on modern NVIDIA GPUs, which include population count instructions and 32-bit registers. As shown later in Sec. VI, typical leaf sizes for tree configurations that provide optimal run-times correspond to approximately 10 to 20 points per leaf node. Given that we also employ base case merging, the C leaf size can increase to approximately 100. Thus, typical base case dimensions will include roughly 10,000 to 40,000 triples to test. Taking the upper end of this range and dividing by 32 for the bit-field representation, we observe that approximately 1,250 bit-field checks will need to be made for the larger base cases and correspondingly fewer checks for the smaller base cases.

In addition to testing the bit-fields, each thread block will need to accumulate the number of triples satisfying the matcher. Such accumulations can be performed, for example, using atomic add operators or reductions within the thread block. The performance of the former option will vary with the proportion of satisfying triples, which may be large due to the small leaf sizes. Thus, we focus on the reduction approach, which offers more predictable performance. Using reductions, the requirement of accumulating contributions to the total satisfied matcher count encourages increased work per thread to amortize reduction costs while the need for more parallelism than is typically available in our base cases encourages less work per thread.

We note that the CUDA programming environment supports concurrent kernel execution via streams and thus several base cases can be executed concurrently. Furthermore, it may be possible to aggregate multiple base cases in such a way that we achieve high parallelism per kernel launch while amortizing reduction costs for the accumulation counter. While such approaches are promising, we herein focus on CPU optimization and defer a more thorough GPU evaluation for future studies.

VI. Results

We now turn to some performance results for our optimized solution for the *Raw Correlation Count* problem. We compare the runtimes against an efficient implementation of a tree-based npcf estimation algorithm in the Ntropy framework [8]. This implementation was used for the previously largest 3pcf computation for studying large scale structure in the galaxy distribution, using 106,824 galaxies at 45 scales [17].

We were able to compute the 3pcf of 1.1 million galaxy positions at the same scales on a single 8-core node in 14 hours. Note that this calculation requires computing the *Raw Correlation Counts* on a random set of 20 million points. To our knowledge, this represents the largest 3pcf calculation of large scale structure at scientifically relevant scales. Since we focus on accelerating the base case computations, our approach can build on existing parallelization techniques in order to achieve scalability to large numbers of nodes. When combined with the speedups due to our base case optimizations, it will then be possible to handle truly massive astronomical data sets.

In addition to the proof-of-concept computation on the galaxy positions, we plot runtimes for a galaxy set of 10^6 points, which is contained in a cube of side length 1,000 megaparsecs (Mpc). We specify matchers in terms of three lengths and a tolerance parameter f; the upper / lower bounds of the matcher are then $r_{ij} \pm \frac{f}{2} r_{ij}$. We show computations for three matchers with $r = 0.25$: a small matcher with side lengths 3, 6, and 3.09 Mpc, a mid matcher consisting of an equilateral triangle with side length 10 Mpc, and a large matcher with side lengths 9, 18, and 27 Mpc. These parameters represent the smallest and largest matchers used in the 3pcf computation mentioned above [17].

In Fig. 6, we compare the runtime for our optimized implementation with the Ntropy framework. For both methods, we used the overall optimal tuning parameter settings. For Ntropy, this meant a bucket size of 2 for all cases. In our algorithms, we used kd-trees with small leaf sizes for the smaller matchers and octrees for the large one. Notice that our implementation outperforms Ntropy regardless of the size of the matchers but yields the greatest speedups when the matchers are large.

In Fig. 5, we show the runtime of our kd-tree implementation as a function of the average number of points per leaf on the three matchers mentioned above. Note that the optimal leaf size depends on the size of the matcher. As the scale of the correlation being computed increases, a slightly larger leaf size becomes optimal.

Experiments were run on a dual-socket node with two quad-core Intel X5550 2.67 GHz Nehalem processors, 24GB of DDR3 host memory, running Red Hat Enterprise Linux 5.

VII. Conclusion

We have investigated an optimized implementation of the fundamental bottleneck computation in computing the npcf. Although efficient, space-partitioning-tree-based algorithms have been developed for this problem and scaled using distributed memory parallelism, the expensive kernel computations between n-tuples of points have limited the use of the npcf for very large data. The previously largest 3-point correlation computation to study large scale structure used only approximately 10^5 galaxies. By optimizing this expensive computation, we are able to show an order-of-magnitude speedup over the existing best implementation. We also demonstrate the applicability of our method with a full-scale computation on 10^6 galaxies performed on a single node. In the future, we will therefore be able to combine our approach with distributed memory parallelism in order to scale to truly massive datasets.

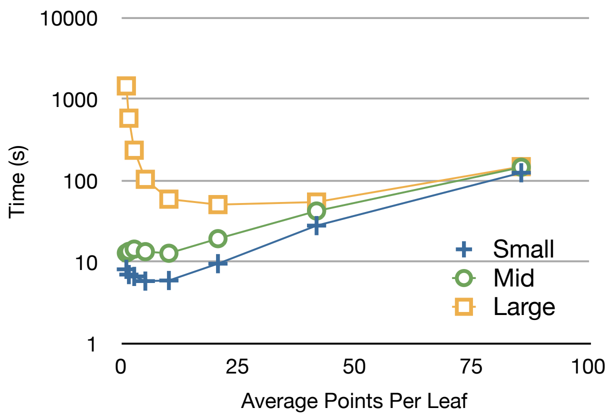


Fig. 5: Runtime vs. average number of points per leaf node in our optimized kd-tree algorithm. The series show results for the small, mid, and large-sized matchers.

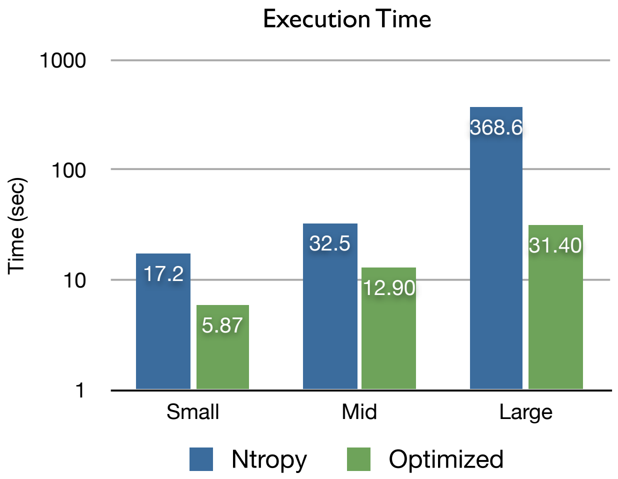


Fig. 6: A plot of the execution time of our implementation compared with the Ntropy code. Results include timing values for three different sizes of matchers.

References

[1] J. Bardeen et al. The statistics of peaks of gaussian random fields. The Astrophysical

Journal, 304:15–61, 1986. 1

[2] M. Beeler, R. Gosper, and R. Schroeppel. Hakmem. 1972. 8

[3] L. Cooper et al. Two-point correlation as a feature for histology images. In Computer Vision and Pattern Recognition Workshops, pages 79–86. IEEE, 2010. 1

[4] P. Dewdney, P. Hall, R. Schilizzi, and T. Lazio. The square kilometre array. Proceedings of the IEEE, 97(8):1482–1496, 2009. 2

[5] J. H. Friedman et al. An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Trans. Math. Softw., 3(3):209–226, 1977. 5

[6] J. Fry. The galaxy correlation hierarchy in perturbation theory. The Astrophysical Journal, 279:499–510, 1984. 1

[7] J. Fry and P. Peebles. Statistical analysis of catalogs of extragalactic objects. IX-The four-point galaxy correlation function. The Astrophysical Journal, 221:19–33, 1978. 2

[8] J. Gardner et al. A framework for analyzing massive astrophysical datasets on a distributed grid. In Astronomical Society of the Pacific Conference Series, volume 376, page 69, 2007. 2, 3, 5, 6, 9

[9] T. Giannantonio et al. A High Redshift Detection of the Integrated Sachs-Wolfe Effect. Physical Review D, 74, 2006. 1

[10] A. G. Gray and A. W. Moore. 'N-Body' Problems in Statistical Learning. In NIPS, volume 4, pages 521–527, 2000. 2, 3, 5

[11] A. Hamilton. Toward better ways to measure the galaxy correlation function. The Astrophysical Journal, 417:19, 1993. 4

[12] S. Landy and A. Szalay. Bias and variance of angular correlation functions. The Astrophysical Journal, 412:64–71, 1993. 4

[13] D. Li et al. 3d reconstruction of carbon nanotube composite microstructure using correlation functions. Journal of Computational and Theoretical Nanoscience, 7(8):1462–1468, 2010. 1

[14] LSST. The Large Synoptic Survey Telescope. `www.lsst.org`. 2

[15] R. Lupton et al. The sdss imaging pipelines. In Astronomical data analysis software and systems X, volume 238, page 269. Astronomical Society of the pacific, 2001. 5

[16] C. McBride. Our non-gaussian universe: Higher order correlation functions in galaxy surveys. In Bulletin of the American Astronomical Society, volume 38, page 957, 2007. 2, 3

[17] C. McBride. Our non-Gaussian universe: Higher order correlation functions in the Sloan Digitial Sky Survey. PhD thesis, University of Pittsburgh, 2010. 2, 5, 9, 10

[18] C. McBride et al. Three-point correlation functions of sdss galaxies: Constraining galaxy-mass bias. The Astrophysical Journal, 739:85, 2011. 1, 3

[19] A. Moore, A. Connolly, C. Genovese, A. Gray, L. Grone, N. Kanidoris II, R. Nichol, J. Schneider, A. Szalay, I. Szapudi, et al. Fast algorithms and efficient statistics: N-point correlation functions. Mining the Sky, pages 71–82, 2001. 2, 3, 5

[20] K. Mosaliganti et al. Tensor classification of n-point correlation function features for histology tissue segmentation. Medical image analysis, 13(1):156–166, 2009. 1

[21] W. Mula. Ssse3: fast popcount. http://wm.ite.pl/articles/sse-popcount.html, 2008. Accessed: 18 April 2012. 8

[22] P. Peebles. The large-scale structure of the universe. Princeton Univ Pr, 1980. 1, 3

[23] U. Pen. Fast power spectrum estimation. Monthly Notices of the Royal Astronomical Society, 346(2):619–626, 2003. 2

[24] U. Pen, T. Zhang, L. Van Waerbeke, Y. Mellier, P. Zhang, and J. Dubinski. Detection of dark matter skewness in the virmos-descart survey: implications for $\omega0$. The Astrophysical Journal, 592:664, 2003. 2

[25] F. P. Preparata and M. I. Shamos. Computational Geometry: An Introduction. Springer, 1985. 5

[26] Y. Rasera, J. Alimi, J. Courtin, F. Roy, P. Corasaniti, A. Fuzfa, and V. Boucher. Introducing the dark energy universe simulation series (deuss). Arxiv preprint arXiv:1002.4950, 2010. 2

[27] R. Ridgway et al. Image segmentation with tensor-based classification of n-point correlation functions. In MICCAI Workshop on Medical Image Analysis with Applications in Biology, 2006. 1

[28] B. Ripley. Locally finite random sets: foundations for point process theory. The Annals of Probability, pages 983–994, 1976. 1

[29] B. Ripley. Spatial statistics. Wiley-Blackwell, 2004. 3

[30] SDSS. The Sloan Digital Sky Survey. `www.sdss.org`. 2

[31] C. Seife. Breakthrough of the Year: Illuminating the Dark Universe. Science, 302(5653):2017–2172, December 19 2003. 1

[32] V. Springel, S. White, A. Jenkins, C. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, et al. Simulations of the formation, evolution and clustering of galaxies and quasars. Nature, 435(7042):629–636, 2005. 2

[33] I. Szapudi. Introduction to higher order spatial statistics in cosmology. Data Analysis in Cosmology, pages 457–492, 2009. 1, 2, 3

[34] I. Szapudi et al. Fast cosmic microwave background analyses via correlation functions. The Astrophysical Journal Letters, 548:L115, 2001. 1

[35] I. Szapudi and A. Szalay. A new class of estimators for the N-point correlations. The Astrophysical Journal Letters, 494:L41, 1998. 2, 4

[36] S. Torquato. Random Heterogeneous Materials. Springer, 2002. 1

[37] S. Torquato and G. Stell. Microstructure of two-phase random media. i. the n-point probability functions. The Journal of Chemical Physics, 77:2071, 1982. 1

[38] H. Warren. Hacker's delight. Addison-Wesley Professional, 2003. 8

[39] S. White. The hierarchy of correlation functions and its relation to other measures of galaxy clustering. Monthly Notices of the Royal Astronomical Society, 186:145–154, 1979. 1

[40] L. Zhang and U. Pen. Fast n-point correlation functions and three-point lensing application. New Astronomy, 10(7):569–590, 2005. 3, 5

[41] X. Zhang and C. Yu. Fast n-point correlation function approximation with recursive convolution for scalar fields. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 634–639. IEEE, 2011. 2