Multidimensional Included and Excluded Sums

Helen Xu* Sean Fraser* Charles E. Leiserson*

Abstract

This paper presents algorithms for the ***included-sums*** and ***excluded-sums*** problems used by scientific computing applications such as the fast multipole method. These problems are defined in terms of a d-dimensional array of N elements and a binary associative operator $\oplus$ on the elements. The included-sum problem requires that the elements within overlapping boxes cornered at each element within the array be reduced using $\oplus$. The excluded-sum problem reduces the elements outside each box. The ***weak*** versions of these problems assume that the operator $\oplus$ has an inverse $\ominus$, whereas the ***strong*** versions do not require this assumption. In addition to studying existing algorithms to solve these problems, we introduce three new algorithms.

The ***bidirectional box-sum (BDBS)*** algorithm solves the strong included-sums problem in $\Theta(dN)$ time, asymptotically beating the classical ***summed-area table (SAT)*** algorithm, which runs in $\Theta(2^d N)$ and which only solves the weak version of the problem. Empirically, the BDBS algorithm outperforms the SAT algorithm in higher dimensions by up to $17.1\times$.

The ***box-complement*** algorithm solves the strong excluded-sums problem in $\Theta(dN)$ time, asymptotically beating the state-of-the-art ***corners*** algorithm by Demaine *et al.*, which runs in $\Omega(2^d N)$ time. The box-complement algorithm empirically outperforms the corners algorithm by about $1.4\times$ given similar amounts of space in three dimensions.

If the assumptions for the weak excluded-sums problem can be satisfied, the ***bidirectional box-sum complement (BDBSC)*** algorithm, which is a trivial extension of the BDBS algorithm, can beat box-complement by up to a factor of 4.

1 Introduction

Many scientific computing applications require reducing many (potentially overlapping) regions of a tensor, or multidimensional array, to a single value for each region quickly and accurately. For example, the integral-image problem (or summed-area table) [3, 6] preprocesses an

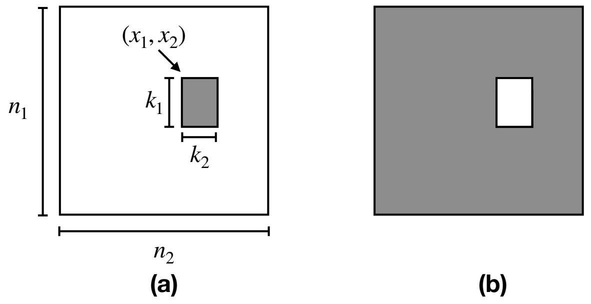

Figure 1: An illustration of included and excluded sums in 2 dimensions on an $n_1 \times n_2$ matrix using a (k_1, k_2)-box. **(a)** For a coordinate (x_1, x_2) of the matrix, the included-sums problem requires all the points in the $k_1 \times k_2$ box cornered at (x_1, x_2), shown as a grey rectangle, to be reduced using a binary associative operator $\oplus$. The included-sums problem requires that this reduction be performed at *every* coordinate of the matrix, not just at a single coordinate as is shown in the figure. **(b)** A similar illustration for excluded sums, which reduces the points outside the box.

image to answer queries for the sum of elements in arbitrary rectangular subregions of a matrix in constant time. The integral image has applications in real-time image processing and filtering [11]. The fast multipole method (FMM) is a widely used numerical approximation for the calculation of long-ranged forces in various N-particle simulations [1, 9]. The essence of the FMM is a reduction of a neighboring subregion's elements, excluding elements too close, using a multipole expansion to allow for fewer pairwise calculations [5, 7]. Specifically, the multipole-to-local expansion in the FMM adds relevant expansions outside some close neighborhood but inside some larger bounding region for each element [1, 14]. High-dimensional applications include the FMM for particle simulations in 3D space [4, 10] and direct summation problems in higher dimensions [13].

These problems give rise to the excluded-sums problem [8], which underlies applications that require reducing regions of a tensor to a single value using a binary associative operator. For example, the excluded-sums problem corresponds to the translation of the local expansion coefficients within each box in the FMM [9]. The problems are called "sums" for ease of presentation, but the general problem statements (and therefore algorithms to solve the problems) apply to any context involving a ***monoid*** $(S, \oplus, e)$, where S is a set of values,

*Computer Science and Artificial Intelligence Laboratory. Massachusetts Institute of Technology. Email: {hjxu, sfraser, cel}@mit.edu.

$\oplus$ is a binary associative operator defined on S, and $e \in S$ is the identity for $\oplus$.

Although the excluded-sums problem is particularly challenging and meaningful for multidimensional tensors, let us start by considering the problem in only 2 dimensions. And, to understand the excluded-sums problem, it helps to understand the included-sums problem as well. Figure 1 illustrates included and excluded sums in 2 dimensions, and Figure 2 provides examples using addition as the $\oplus$ operator. We have an $n_1 \times n_2$ matrix $\mathcal{A}$ of elements over a monoid $(S, \oplus, e)$. We also are given a "box size" $\mathbf{k} = (k_1, k_2)$ such that $k_1 \leq n_1$ and $k_2 \leq n_2$. The ***included sum*** at a coordinate (x_1, x_2), as shown in Figure 1(a), involves ***reducing*** — accumulating using $\oplus$ — all the elements of $\mathcal{A}$ inside the $\mathbf{k}$-box ***cornered*** at (x_1, x_2), that is,

$$\bigoplus_{y_1=x_1}^{x_1+k_1-1} \bigoplus_{y_2=x_2}^{x_2+k_2-1} \mathcal{A}[y_1, y_2] \ ,$$

where if a coordinate goes out of range, we assume that its value is the identity e. The ***included-sums problem*** computes the included sum for all coordinates of $\mathcal{A}$, which can be straightforwardly accomplished with four nested loops in $\Theta(n_1 n_2 k_1 k_2)$ time. Similarly, the ***excluded sum*** at a coordinate, as shown in Figure 1(b), reduces all the elements of $\mathcal{A}$ outside the $\mathbf{k}$-box cornered at (x_1, x_2). The ***excluded-sums problem*** computes the excluded sum for all coordinates of $\mathcal{A}$, which can be straightforwardly accomplished in $\Theta(n_1 n_2 (n_1 - k_1)(n_2 - k_2))$ time. We shall see much better algorithms for both problems.

$$\begin{pmatrix} 1 & 3 & 6 & 2 & 5 \\ 3 & 9 & 1 & 1 & 2 \\ 5 & 1 & 5 & 3 & 2 \\ 4 & 3 & 2 & 0 & 9 \\ 6 & 2 & 1 & 7 & 8 \end{pmatrix} \quad \begin{pmatrix} 16 & 19 & 10 & 10 & 7 \\ 18 & 16 & 10 & 8 & 4 \\ 13 & 11 & 10 & 14 & 11 \\ 15 & 8 & 10 & 24 & 17 \\ 8 & 3 & 8 & 15 & 8 \end{pmatrix} \quad \begin{pmatrix} 75 & 72 & 81 & 81 & 84 \\ 73 & 75 & 81 & 83 & 87 \\ 78 & 80 & 81 & 77 & 80 \\ 76 & 83 & 81 & 67 & 74 \\ 83 & 78 & 83 & 76 & 83 \end{pmatrix}$$

(a) (b) (c)

Figure 2: Examples of the included- and excluded-sums problems on an input matrix in 2 dimensions with box size $(3, 3)$ using the max operator. **(a)** The input matrix. The square shows the box cornered at $(3, 3)$. **(b)** The solution for the included-sums problem with the $+$ operator. The highlighted square contains the included sum for the box in (a). The included-sums problem requires computing the included sum for every element in the input matrix. **(c)** A similar example for excluded sums. The highlighted square contains the excluded sum for the box in (a).

Excluded Sums and Operator Inverse One way to solve the excluded-sums problem is to solve the included-sums problem and then use the inverse $\ominus$ of the $\oplus$ operator to "subtract" out the results from the reduction of the entire tensor. This approach fails for operators without an inverse, however, such as the maximum operator max. As another example, the FMM involves solving the excluded-sums problem over a domain of functions which cannot be "subtracted," because the functions exhibit singularities [8]. Even for simpler domains, using the inverse (if it exists) may have unintended consequences. For example, subtracting finite-precision floating-point values can suffer from catastrophic cancellation [8, 17] and high round-off error [12]. Some contexts may permit the use of an inverse, but others may not.

Consequently, we refine the included- and excluded-sums problems into ***weak*** and ***strong*** versions. The weak version requires an operator inverse, while the strong version does not. Any algorithm for the included-sums problem trivially solves the weak excluded-sums problem, and any algorithm for the strong excluded-sums problem trivially solves the weak excluded-sums problem. This paper presents efficient algorithms for both the weak and strong excluded-sums problems.

Summed-area Table for Weak Excluded Sums The ***summed-area table (SAT)*** algorithm uses the classical summed-area table method [3, 6, 15] to solve the weak included-sums problem on a d-dimensional tensor $\mathcal{A}$ having N elements in $O(2^d N)$ time. The SAT algorithm cannot be used to solve the strong included-sums problem, however, because it requires an operator inverse. The summed-area table algorithm can easily be extended to an algorithm for weak excluded-sums by totaling the entire tensor and subtracting the solution to weak included sums. We will call this algorithm the ***SAT complement (SATC) algorithm***.

Corners Algorithm for Strong Excluded Sums The naive algorithm for strong excluded sums that just sums up the area of interest for each element runs in $O(N^2)$ time in the worst case, because it wastes work by recomputing reductions for overlapping regions. To avoid recomputing sums, Demaine *et al.* [8] introduced an algorithm that solve the strong excluded-sums problem in arbitrary dimensions, which we will call the ***corners algorithm***.

Given a d-dimensional tensor of N elements, the corners algorithm takes $\Omega(2^d N)$ time to compute the excluded sum in the best case because there are 2^d corners and each one requires $\Omega(N)$ time to add its contribution to each excluded box [16]. The bound is tight: given $\Theta(dN)$ space, the corners algorithm takes $\Theta(2^d N)$ time. With $\Theta(N)$ space, the corners algorithm takes $\Theta(d 2^d N)$ time [16].

Algorithm	*Source*	*Time*	*Space*	*Included or Excluded?*	*Strong or Weak?*
Naive included sum	[This work]	$\Theta(KN)$	$\Theta(N)$	Included	Strong
Naive included sum complement	[This work]	$\Theta(KN)$	$\Theta(N)$	Excluded	Weak
Naive excluded sums	[This work]	$\Theta(N^2)$	$\Theta(N)$	Excluded	Strong
Summed-area table (SAT)	[6, 15]	$\Theta(2^d N)$	$\Theta(N)$	Included	Weak
Summed-area table complement (SATC)	[6, 15]	$\Theta(2^d N)$	$\Theta(N)$	Excluded	Weak
Corners(c)	[8]	$\Theta((d+1/c)2^d N)$	$\Theta(cN)$	Excluded	Strong
Corners spine	[8]	$\Theta(2^d N)$	$\Theta(dN)$	Excluded	Strong
Bidirectional box sum (BDBS)	[This work]	$\Theta(dN)$	$\Theta(N)$	Included	Strong
Bidirectional box sum complement (BDBSC)	[This work]	$\Theta(dN)$	$\Theta(N)$	Excluded	Weak
Box-complement	[This work]	$\Theta(dN)$	$\Theta(N)$	Excluded	Strong

Table 1: A summary of all algorithms for excluded sums in this paper. All algorithms take as input a d-dimensional tensor of N elements. We include the runtime, space usage, whether an algorithm solves the included- or excluded-sums problem, and whether it solves the strong or weak version of the problem. We use K to denote the volume of the box (in the runtime of the naive algorithm).

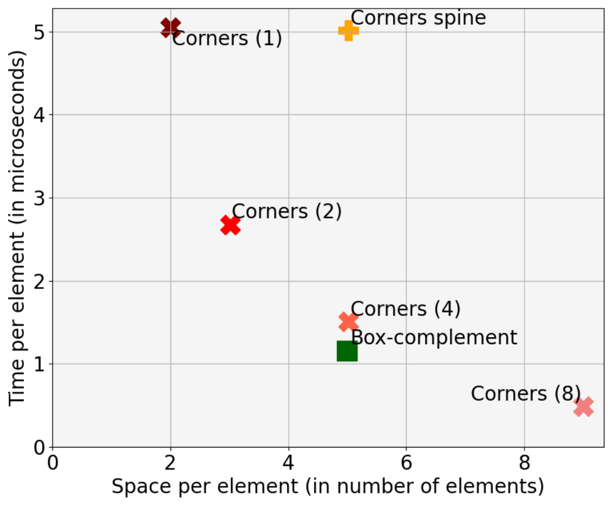

Figure 3: Space and time per element of the corners and box-complement algorithms in 3 dimensions. We use `Corners(c)` and `Corners Spine` to denote variants of the corners algorithm with extra space. We set the number of elements $N = 681472 = 88^3$ and the box lengths $k_1 = k_2 = k_3 = 4$ (for $K = 64$).

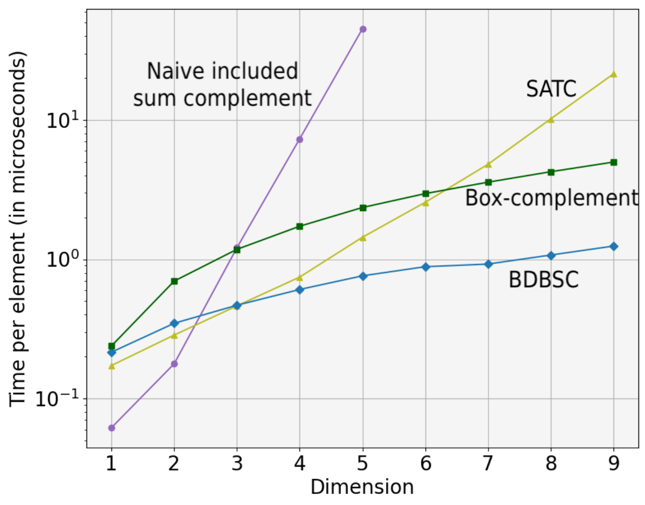

Figure 4: Time per element of algorithms for excluded sums in arbitrary dimensions. The number of elements N of the tensor in each dimension was in the range $[2097152, 134217728]$ (selected to be a exact power of the number of dimensions). For each number of dimensions d, we set the box volume $K = 8^d$.

Contributions This paper presents algorithms for included and strong excluded sums in arbitrary dimensions that improve the runtime from exponential to linear in the number of dimensions. For strong included sums, we introduce the ***bidirectional box-sum*** (BDBS) algorithm that uses prefix and suffix sums to compute the included sum efficiently. The BDBS algorithm can be easily extended into an algorithm for weak excluded sums, which we will call the ***bidirectional box-sum complement*** (BDBSC) algorithm. For strong excluded sums, the main insight in this paper is the formulation of the excluded sums in terms of the "box complement" on which the ***box-complement*** algorithm is based. Table 1 summarizes all algorithms considered in this paper.

Figure 3 illustrates the performance and space usage of the box-complement algorithm and variants of the 3D corners algorithm. Since the paper that introduced the corners algorithm stopped short of a general construction in higher dimensions, the 3D case is the highest dimensionality for which we have implementations of the box-complement and corners algorithm. The 3D case is of interest because applications such as the FMM often present in three dimensions [4, 10]. We find that the box-complement algorithm outperforms the corners algorithm by about 1.4× when given similar amounts of space, though the corners algorithm with twice the space as box-complement is 2× faster. The box-complement algorithm uses a fixed (constant) factor of extra space, while the corners algorithm can use a variable amount of space. We found that the performance of the corners algorithm depends heavily on its space usage. We use `Corners(c)` to denote the implementation of the corners algorithm that uses a factor of c in space to store leaves in the computation tree and gather the results into the output. Furthermore, we also explored a variant of the corners algorithm [16] called `Corners spine`, which uses extra space to store the spine of the computation tree and asymptotically reduce the runtime.

Figure 4 demonstrates how algorithms for weak excluded sums scale with dimension. We omit the corners algorithm because the original paper stopped short of a construction of how to find the corners in higher dimensions. We also omit an evaluation of included-sums algorithms because the relative results of all algorithms would be the same. The naive and summed-area table perform well in lower dimensions but exhibit crossover points (at 3 and 6 dimensions, respectively) because their runtimes grow exponentially with dimension. In contrast, the BDBS and box-complement algorithms scale linearly in the number of dimensions and outperform the summed-area table method by at least $1.3\times$ after 6 dimensions. The BDBS algorithm demonstrates the advantage of solving the weak problem, if you can, because it is always faster than the box-complement algorithm, which doesn't exploit an operator inverse. Both algorithms introduced in this paper outperform existing methods in higher dimensions, however.

To be specific, our contributions are as follows:

- the bidirectional box-sum (BDBS) algorithm for strong included sums;
- the bidirectional box-sum complement (BDBSC) algorithm for weak excluded sums;
- the box-complement algorithm for strong excluded sums;
- theorems showing that, for a d-dimensional tensor of size N, these algorithms all run in $\Theta(dN)$ time and $\Theta(N)$ space;
- implementations of these algorithms in `C++`; and
- empirical evaluations showing that the box-complement algorithm outperforms the corners algorithm in 3D given similar space and that both the BDBSC algorithm and box-complement algorithm outperform the SATC algorithm in higher dimensions.

Outline The rest of the paper is organized as follows. Section 2 provides necessary preliminaries and notation to understand the algorithms and proofs. Section 3 presents an efficient algorithm to solve the included-sums problem, which will be used as a key subroutine in the box-complement algorithm. Section 4 formulates the excluded sum as the "box-complement," and Section 5 describes and analyzes the resulting box-complement algorithm. Section 6 presents an empirical evaluation of algorithms for excluded sums. Finally, we provide concluding remarks in Section 7.

2 Preliminaries

This section reviews tensor preliminaries used to describe algorithms in later sections. It also formalizes the included- and excluded-sums problems in terms of tensor notation. Finally, it describes the prefix- and suffix-sums primitive underlying the main algorithms in this paper.

Tensor Preliminaries We first introduce the coordinate and tensor notation we use to explain our algorithms and why they work. At a high level, tensors are d-dimensional arrays of elements over some monoid $(S, \oplus, e)$. In this paper, tensors are represented by capital script letters (e.g., $\mathcal{A}$) and vectors are represented by lowercase boldface letters (e.g., $\mathbf{a}$).

We shall use the following terminology. A d-dimensional ***coordinate domain*** U is the cross product $U = U_1 \times U_2 \times \ldots \times U_d$, where $U_i = \{1, 2, \ldots, n_i\}$ for $n_i \geq 1$. The ***size*** of U is $n_1 n_2 \cdots n_d$. Given a coordinate domain U and a monoid $(S, \oplus, e)$ as defined in Section 1, a ***tensor*** $\mathcal{A}$ can be viewed for our purposes as a mapping $\mathcal{A} : U \to S$. That is, a tensor maps a coordinate $\mathbf{x} \in U$ to an ***element*** $\mathcal{A}[\mathbf{x}] \in S$. The ***size*** of a tensor is the size of its coordinate domain. We omit the coordinate domain U and monoid $(S, \oplus, e)$ when they are clear from context.

We use Python-like ***colon notation*** $x : x'$, where $x \leq x'$, to denote the half-open interval $[x, x')$ of coordinates along a particular dimension. If $x : x'$ would extend outside of $[1, n]$, where n is the maximum coordinate, it denotes only the coordinates actually in the interval, that is, the interval $\max\{1, x\} : \min\{n + 1, x'\}$. If the lower bound is missing, as in $: x'$, we interpret the interval as $1 : x'$, and similarly, if the upper bound is missing, as in $x :$, it denotes the interval $[x, n]$. If both bounds are missing, as in $:$, we interpret the interval as the whole coordinate range $[1, n]$.

We can use colon notation when indexing a tensor $\mathcal{A}$ to define ***subtensors***, or ***boxes***. For example, $\mathcal{A}[3 : 5, 4 : 6]$ denotes the elements of $\mathcal{A}$ at coordinates $(3, 4), (3, 5), (4, 4), (4, 5)$. For full generality, a box B ***cornered*** at coordinates $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_d)$, where $x_i < x'_i$ for all $i = 1, 2, \ldots, d$, is the box $(x_1 : x'_1, x_2 : x'_2, \ldots, x_d : x'_d)$. Given a ***box size*** $\mathbf{k} = (k_1, \ldots, k_d)$, a **$\mathbf{k}$*-box cornered*** at coordinate $\mathbf{x}$ is the box cornered at $\mathbf{x}$ and $\mathbf{x}' = (x_1 + k_1, x_2 + k_2, \ldots, x_d + k_d)$. A ***(tensor) row*** is a box with a single value in each coordinate position in the colon notation, except for one position, which includes that entire dimension. For example, if $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is a coordinate of a tensor $\mathcal{A}$, then $\mathcal{A}[x_1, x_2, \ldots, x_{i-1}, :, x_{i+1}, x_{i+2}, \ldots, x_d]$ denotes a row along dimension i.

The colon notation can be combined with the reduction operator $\oplus$ to indicate the reduction of all elements in a subtensor:

$$\bigoplus \mathcal{A}[x_1 : x'_1, x_2 : x'_2, \ldots, x_d : x'_d] = \bigoplus_{y_1 \in [x_1, x'_1)} \bigoplus_{y_2 \in [x_2, x'_2)} \cdots \bigoplus_{y_d \in [x_d, x'_d)} \mathcal{A}[y_1, y_2, \ldots, y_d] \ .$$

Problem Definitions We can now formalize the included- and excluded-sums problems from Section 1.

Definition 1. (Included and Excluded Sums) *An algorithm for the* ***included-sums problem*** *takes as input a d-dimensional tensor $\mathcal{A} : U \to S$ with size N and a box size $\mathbf{k} = (k_1, k_2, \ldots, k_d)$. It produces a new tensor $\mathcal{A}' : U \to S$ such that every output element $\mathcal{A}'[\mathbf{x}]$ holds the reduction under $\oplus$ of elements within the $\mathbf{k}$-box of $\mathcal{A}$ cornered at $\mathbf{x}$. An algorithm for the* ***excluded-sums problem*** *is defined similarly, except that the reduction is of elements outside the $\mathbf{k}$-box cornered at $\mathbf{x}$.*

In other words, an included-sums algorithm computes, for all $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in U$, the value $\mathcal{A}'[\mathbf{x}] = \bigoplus \mathcal{A}[x_1 : x_1 + k_1, x_2 : x_2 + k_2, \ldots, x_d : x_d + k_d]$. It's messier to write the output of an excluded-sums problem using colon notation, but fortunately, our proofs do not rely on it.

As we have noted in Section 1, there are weak and strong versions of both problems which allow and do not allow an operator inverse, respectively.

Prefix and Suffix Sums The ***prefix-sums operation*** [2] takes an array $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ of n elements and returns the "running sum" $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, where

$$b_k = \begin{cases} a_1 & \text{if } k = 1, \\ a_k \oplus b_{k-1} & \text{if } k > 1 \ . \end{cases} \tag{2.1}$$

Let Prefix denote the algorithm that directly implements the recursion in Equation 2.1. Given an array $\mathbf{a}$ and indices $start \le end$, the function Prefix($\mathbf{a}$, $start$, end) computes the prefix sum in the range $[start, end]$ of $\mathbf{a}$ in $O(end - start)$ time. Similarly, the ***suffix-sums operation*** is the reverse of the prefix sum and computes the sum right-to-left rather than left-to-right. Let Suffix($\mathbf{a}$, $start$, end) be the corresponding algorithm for suffix sums.

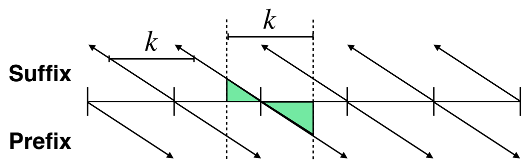

Figure 5: An illustration of the computation in the bidirectional box-sum algorithm. The arrows represent prefix and suffix sums in runs of size k, and the shaded region represents the prefix and suffix components of the region of size k outlined by the dotted lines.

3 Included Sums

This section presents the ***bidirectional box-sum algorithm (BDBS) algorithm*** to compute the included sum along an arbitrary dimension, which is used as a main subroutine in the box-complement algorithm for excluded sums. As a warm-up, we will first describe how to solve the included-sums problem in one dimension and then extend the technique to higher dimensions. We include the one-dimensional case for clarity, but the main focus of this paper is the multidimensional case. The full version of the paper includes all the pseudocode and omitted proofs for BDBS in 1D [16].

Included Sums in 1D Before investigating the included sums in higher dimensions, let us first turn our attention to the 1D case for ease of understanding. We will sketch an algorithm BDBS-1D which takes as input a list A of length N and a (scalar) box size[1] k and outputs a list A' of corresponding included sums. At a high level, the BDBS-1D algorithm generates two intermediate lists A_p and A_s, each of length N, and performs N/k prefix and suffix sums of length k on each intermediate list. By construction, for $x = 1, 2, \ldots, N$, we have $A_p[x] = A[k\lfloor x/k \rfloor : x + 1]$, and $A_s[x] = A[x : k\lceil (x+1)/k \rceil]$.

Finally, BDBS-1D uses A_p and A_s to compute the included sum of size k for each coordinate in one pass. Figure 5 illustrates the ranged prefix and suffix sums in BDBS-1D, and Figure 6 presents a concrete example of the computation.

BDBS-1D solves the included-sums problem on an array of size N in $\Theta(N)$ time and $\Theta(N)$ space. First, it uses two temporary arrays to compute the prefix and suffix as illustrated in Figure 5 in $\Theta(N)$ time. It then makes one more pass through the data to compute the included sum, requiring $\Theta(N)$ time.

[1]For simplicity in the algorithm descriptions and pseudocode, we assume that $n_i \bmod k_i = 0$ for all dimensions $i = 1, 2, \ldots, d$. In implementations, the input can either be padded with the identity to make this assumption hold, or it can add in extra code to deal with unaligned boxes.

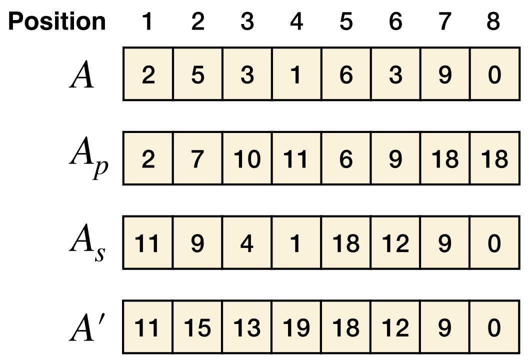

Figure 6: An example of computing the 1D included sum using the bidirectional box-sum algorithm, where $N = 8$ and $k = 4$. The input array is A, the k-wise prefix and suffix sums are stored in A_p and A_s, respectively, and the output is in A'.

Generalizing to Arbitrary Dimensions The main focus of this work is multidimensional included and excluded sums. Computing the included sum along an arbitrary dimension is almost exactly the same as computing it along 1 dimension in terms of the underlying ranged prefix and suffix sums. We sketch an algorithm BDBS that generalizes BDBS-1D to arbitrary dimensions.

Let $\mathcal{A}$ be a d-dimensional tensor with N elements and let $\mathbf{k}$ be a box size. The BDBS algorithm computes the included sum along dimensions $i = 1, 2, \ldots, d$ in turn. After performing the included-sum computation along dimensions $1, 2, \ldots, i$, every coordinate in the output $\mathcal{A}_i$ contains the included sum in each dimension up to i:

$$\mathcal{A}_i[x_1, x_2, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{x_1 : x_1 + k_2, \ldots, x_i : x_i + k_i}_{i}, \underbrace{x_{i+1}, \ldots, x_d}_{d-i}].$$

Overall, BDBS computes the full included sum of a tensor with N elements in $\Theta(dN)$ time and $\Theta(N)$ space by performing the included sum along each dimension in turn.

Although we cannot directly use BDBS to solve the strong excluded-sums problem, the next sections demonstrate how to use the BDBS technique as a key subroutine in the box-complement algorithm for strong excluded sums.

4 Excluded Sums and the Box Complement

The main insight in this section is the formulation of the excluded sum as the recursive "box complement". We show how to partition the excluded region into $2d$ non-overlapping parts in d dimensions. This decomposition of the excluded region underlies the box-complement for strong excluded sums in the next section.

First, let's see how the formulation of the "box complement" relates to the excluded sum. At a high level, given a box B, a coordinate $\mathbf{x}$ is in the "i-complement" of B if and only if $\mathbf{x}$ is "out of range" in some dimension $j \leq i$, and "in the range" for all dimensions greater than i.

Definition 2. (Box Complement) *Given a d-dimensional coordinate domain U and a dimension $i \in \{1, 2, \ldots, d\}$, the* ***i-complement*** *of a box B cornered at coordinates $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, \ldots, x'_d)$ is the set*

$$C_i(B) = \{(y_1, \ldots, y_d) \in U : \textit{ there exists } j \in [1, i] \textit{ such that } y_j \notin [x_j, x'_j), \textit{ and for all } m \in [i+1, d], y_m \in [x_m, x'_m)\}.$$

Given a box B, the reduction of all elements at coordinates in $C_d(B)$ is exactly the excluded sum with respect to B. The box complement recursively partitions an excluded region into disjoint sets of coordinates.

Theorem 4.1. (Recursive Box-complement) *Let B be a box cornered at coordinates $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, \ldots, x'_d)$ in some coordinate domain U. The i-complement of B can be expressed recursively in terms of the $(i-1)$-complement of B as follows:*

$$C_i(B) = (\underbrace{:, \ldots, :}_{i-1}, :x_i, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \cup (\underbrace{:, \ldots, :}_{i-1}, x'_i :, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \cup C_{i-1}(B),$$

where $C_0(B) = \emptyset$.

Proof. For simplicity of notation, let $\mathrm{RHS}_i(B)$ be the right-hand side of the equation in the statement of Theorem 4.1. Let $\mathbf{y} = (y_1, \ldots, y_d)$ be a coordinate. In order to show the equality, we will show that $\mathbf{y} \in C_i(B)$ if and only if $\mathbf{y} \in \mathrm{RHS}_i(B)$.

Forward Direction: $\mathbf{y} \in C_i(B) \to \mathbf{y} \in \mathrm{RHS}_i(B)$.
We proceed by case analysis when $\mathbf{y} \in C_i(B)$. Let $j \leq i$ be the highest dimension at which $\mathbf{y}$ is "out of range," or where $y_j < x_j$ or $y_j \geq x'_j$.

Case 1: $j = i$.
Definition 2 and $j = i$ imply that either $y_i < x_i$ or $y_i \geq x'_i$, and $x_m \leq y_m \leq x'_m$ for all $m > i$. By definition, $y_i < x_i$ implies $\mathbf{y} \in (:, \ldots, :, :x_i, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d)$. Similarly, $y_i \geq x'_i$ implies $\mathbf{y} \in (:, \ldots, :, x'_i :, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d)$. These are exactly the first two terms in $\mathrm{RHS}_i(B)$.

Case 2: $j < i$.
Definition 2 and $j < i$ imply that $\mathbf{y} \in C_{i-1}(B)$.

Backwards Direction: $\mathbf{y} \in \mathrm{RHS}_i(B) \to \mathbf{y} \in C_i(B)$. We again proceed by case analysis.

Case 1: $\mathbf{y} \in \big(:, \ldots, :, :x_i, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d\big)$ or $\mathbf{y} \in \big(:, \ldots, :, x'_i :, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d\big)$.
Definition 2 implies $\mathbf{y} \in C_i(B)$ because there exists some $j \le i$ (in this case, $j = i$) such that $y_j < x_j$ and $x_m \le y_m < x'_m$ for all $m > i$.

Case 2: $\mathbf{y} \in C_{i-1}(B)$.
Definition 2 implies that there exists j in the range $1 \le j \le i-1$ such that $y_j < x_j$ or $y_j \ge x'_j$ and that for all $m \ge i$, we have $x_m \le y_m < x'_m$. Therefore, $\mathbf{y} \in C_{i-1}(B)$ implies $\mathbf{y} \in C_i(B)$ since there exists some $j \le i$ (in this case, $j < i$) where $y_j < x_j$ or $y_j \ge x'_j$ and $x_m \le y_m < x'_m$ for all $m > 1$.

Therefore, $C_i(B)$ can be recursively expressed as $\mathrm{RHS}_i(B)$. □

In general, unrolling the recursion in Theorem 4.1 yields $2d$ disjoint partitions that exactly comprise the excluded sum with respect to a box.

Corollary 4.1. (Excluded-sum Components) *The excluded sum can be represented as the union of $2d$ disjoint sets of coordinates as follows:*

$$C_d(B) = \bigcup_{i=1}^{d} \left(\big(\underbrace{:, \ldots, :}_{i-1}, :x_i, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}\big) \cup \big(\underbrace{:, \ldots, :}_{i-1}, x_i + k_i :, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}\big) \right) .$$

We use the box-complement formulation in the next section to efficiently compute the excluded sums on a tensor by reducing in disjoint regions of the tensor.

5 Box-Complement Algorithm

This section describes and analyzes the box-complement algorithm for strong excluded sums, which efficiently implements the dimension reduction in Section 4. The box-complement algorithm relies heavily on prefix, suffix, and included sums as described in Sections 2 and 3.

Given a d-dimensional tensor $\mathcal{A}$ of size N and a box size $\mathbf{k}$, the box-complement algorithm solves the excluded-sums problem with respect to $\mathbf{k}$ for coordinates in $\mathcal{A}$ in $\Theta(dN)$ time and $\Theta(N)$ space. The full version of the paper contains all omitted pseudocode and proofs for the box-complement algorithm [16].

Algorithm Sketch At a high level, the box-complement algorithm proceeds by dimension reduction. That is, the algorithm takes d dimension-reduction steps, where each step adds two of the components from Corollary 4.1 to each element in the output tensor. In the ith dimension-reduction step, the box-complement algorithm computes the i-complement of B (Definition 2) for all coordinates in the tensor by performing a prefix and suffix sum along the ith dimension and then using the BDBS technique along the remaining $d-i$ dimensions. After the ith dimension-reduction step, the box-complement algorithm operates on a tensor of $d-i$ dimensions because i dimensions have been reduced so far via prefix sums. Figure 7 presents an example of the dimension reduction in 2 dimensions, and Figure 8 illustrates the recursive box-complement in 3 dimensions.

Prefix and Suffix Sums In the ith dimension reduction step, the box-complement algorithm uses prefix and suffix sums along the ith dimension to reduce the elements "out of range" along the ith dimension in the i-complement. That is, given a tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdots n_d$ and a number $i < d$ of dimensions reduced so far, we define a subroutine Prefix-Along-Dim$(\mathcal{A}, i)$ that fixes the first i dimensions at $n_1, \ldots, n_i$ (respectively), and then computes the prefix sum along dimension $i+1$ for all remaining rows in dimensions $i+2, \ldots, d$ in $O\Big(\prod_{j=i+1}^{d} n_j\Big)$ time.

The subroutine Prefix-Along-Dim computes the reduction of elements "out of range" along dimension i. That is, after Prefix-Along-Dim$(\mathcal{A}, i)$, for each coordinate $x_{i+1} = 1, 2, \ldots, n_{i+1}$ along dimension $i+1$, every coordinate in the (dimension-reduced) output $\mathcal{A}'$ contains the prefix up to that coordinate in dimension $i+1$:

$$\mathcal{A}'[\underbrace{n_1, \ldots, n_i}_{i}, x_{i+1}, \underbrace{x_{i+2}, \ldots, x_d}_{d-i-1}] = \bigoplus \mathcal{A}[\underbrace{n_1, \ldots, n_i}_{i}, :x_{i+1}+1, \underbrace{x_{i+2}, \ldots, x_d}_{d-i-1}].$$

Since the similar subroutine Suffix-Along-Dim has almost exactly the same analysis and structure, we omit its discussion.

Included Sums In the ith dimension reduction step, the box-complement algorithm uses the BDBS technique along the ith dimension to reduce the elements "in range" along the ith dimension in the i-complement. That is, given a tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdots n_d$ and a number $i < d$ of dimensions reduced so far, we define a subroutine BDBS-Along-Dim.

BDBS-Along-Dim computes the included sum for each row along a specified dimension after dimension reduction. Let $\mathcal{A}$ be a d-dimensional tensor, $\mathbf{k}$ be a box size, i be the number of reduced dimensions so far,

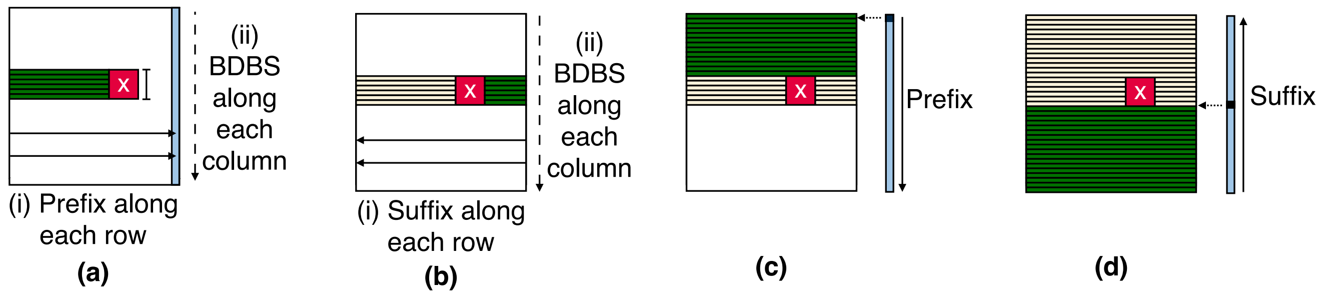
(ii) BDBS along each column
(i) Prefix along each row
(a)
(ii) BDBS along each column
(i) Suffix along each row
(b)
Prefix
(c)
Suffix
(d)
x

Figure 7: Steps for computing the excluded sum in 2 dimensions with included sums on prefix and suffix sums. The steps are labeled in the order they are computed. The 1-complement **(a)** prefix and **(b)** suffix steps perform a prefix and suffix along dimension 1 and an included sum along dimension 2. The numbers in **(a)**,**(b)** represent the order of subroutines in those steps. The 2-complement **(c)** prefix and **(d)** suffix steps perform a prefix and suffix sum on the reduced array, denoted by the blue rectangle, from step **(a)**. The red box denotes the excluded region, and solid lines with arrows denote prefix or suffix sums along a row or column. The long dashed line represents the included sum along each column.

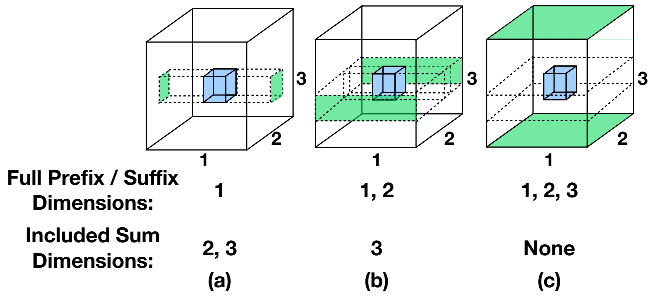

Figure 8: An example of the recursive box-complement in 3 dimensions with dimensions labeled $1, 2, 3$. The subfigures **(a)**, **(b)**, and **(c)** illustrate the 1-, 2-, and 3-complement, respectively. The blue region represents the coordinates inside the box, and the regions outlined by dotted lines represent the partitions defined by Corollary 4.1. For each partition, the face against the edge of the tensor is highlighted in green.

and j be the dimension to compute the included sum along such that $j > i$. BDBS-ALONG-DIM$(\mathcal{A}, \mathbf{k}, i, j)$ computes the included sum along the jth dimension for all rows $(n_1, \ldots, n_i, :, \ldots, :)$. That is, for each coordinate $\mathbf{x} = (n_1, \ldots, n_i, x_{i+1}, \ldots, x_d)$, the output tensor $\mathcal{A}'$ contains the included sum along dimension j:

$$\mathcal{A}'[\mathbf{x}] = \bigoplus \mathcal{A}[\underbrace{n_1, \ldots, n_i}_{i}, \underbrace{x_{i+1}, \ldots, x_j}_{j-i}, \\ x_{j+1} : x_{j+1} + k_{j+1}, \underbrace{x_{j+2}, \ldots, x_d}_{d-j-1}].$$

BDBS-ALONG-DIM$(\mathcal{A}, \mathbf{k}, i, j)$ takes $\Theta\left(\prod_{\ell=i+1}^{d} n_\ell\right)$ time because it iterates over $\left(\prod_{\ell=i+1}^{d} n_\ell\right)/n_{j+1}$ rows and runs in $\Theta(n_{j+1})$ time per row. It takes $\Theta(N)$ space using the same technique as BDBS-1D.

Adding in the Contribution Each dimension-reduction step must add its respective contribution to each element in the output. Given an input tensor $\mathcal{A}$ and output tensor $\mathcal{A}'$, both of size N, the function ADD-CONTRIBUTION takes $\Theta(N)$ time to add in the contribution with a pass through the tensors.

Putting It All Together Finally, we will see how to use the previously defined subroutines to describe and analyze the box-complement algorithm for excluded sums. Figure 9 presents pseudocode for the box-complement algorithm. Each dimension-reduction step has a corresponding prefix and suffix step to add in the two components in the recursive box-complement. Given an input tensor $\mathcal{A}$ of size N, the box-complement algorithm takes $\Theta(N)$ space because all of its subroutines use at most a constant number of temporaries of size N, as seen in Figure 9.

Given a tensor $\mathcal{A}$ as input, the box-complement algorithm solves the excluded-sums problem by computing the recursive box-complement components from Corollary 4.1. By construction, for dimension $i \in [1, d]$, the prefix-sum part of the ith dimension-reduction step outputs a tensor $\mathcal{A}_p$ such that for all coordinates $\mathbf{x} = (x_1, \ldots, x_d)$, we have

$$\mathcal{A}_p[x_1, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{:, \ldots, :}_{i}, :x_{i+1}, \\ \underbrace{x_{i+2} : x_{i+2} + k_{i+2}, \ldots, x_d : x_d + k_d}_{d-i-1}].$$

Similarly, the suffix-sum step constructs a tensor $\mathcal{A}_s$ such that for all $\mathbf{x}$,

$$\mathcal{A}_s[x_1, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{:, \ldots, :}_{i}, x_{i+1} + k_{i+1} :, \\ \underbrace{x_{i+2} : x_{i+2} + k_{i+2}, \ldots, x_d : x_d + k_d}_{d-i-1}].$$

We can now analyze the performance of the box-complement algorithm.

THEOREM 5.1. (TIME OF BOX-COMPLEMENT) *Given a d-dimensional tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdot \ldots \cdot n_d$,* BOX-COMPLEMENT *solves the excluded-sums problem in $\Theta(dN)$ time.*

Proof. We analyze the prefix step (since the suffix step is symmetric, it has the same running time). Let $i \in \{1, \ldots, d\}$ denote a dimension.

The ith dimension reduction step in BOX-COMPLEMENT involves 1 prefix step and $(d-i)$ included sum calls, which each take $O\left(\prod_{j=i}^{d} n_j\right)$ time. Furthermore, adding in the contribution at each dimension-reduction step takes $\Theta(N)$ time. The total time over d steps is therefore $\Theta\left(\sum_{i=1}^{d}\left((d-i+1)\prod_{j=i}^{d} n_j + N\right)\right)$. Adding in the contribution is clearly $\Theta(dN)$ in total.

Next, we bound the runtime of the prefix and included sums. In each dimension-reduction step, reducing the number of dimensions of interest exponentially decreases the size of the considered tensor. That is, dimension reduction exponentially reduces the size of the input: $\prod_{j=i}^{d} n_j \leq N/2^{i-1}$. The total time required to compute the box-complement components is therefore

$$\sum_{i=1}^{d}(d-i+1)\prod_{j=i}^{d} n_j \leq \sum_{i=1}^{d}(d-i+1)\frac{N}{2^{i-1}}$$
$$\leq 2(d+2^{-d}-1)N = \Theta(dN).$$

Therefore, the total time of BOX-COMPLEMENT is $\Theta(dN)$. □

```
BOX-COMPLEMENT($\mathcal{A}$, $\mathbf{k}$)
 1  // Input: Tensor $\mathcal{A}$ with $d$-dimensions, box size $\mathbf{k}$
    // Output: Tensor $\mathcal{A}'$ with size and dimensions
    // matching $\mathcal{A}$ containing the excluded sum.
 2  init $\mathcal{A}'$ with the same size as $\mathcal{A}$
 3  $\mathcal{A}_p \leftarrow \mathcal{A}$; $\mathcal{A}_s \leftarrow \mathcal{A}$
 4  // Current dimension-reduction step
 5  for $i \leftarrow 1$ to $d$
 6  // Saved from previous dimension-reduction step.
 7      $\mathcal{A}_p \leftarrow \mathcal{A}$ reduced up to dimension $i-1$
 8      $\mathcal{A}_s \leftarrow \mathcal{A}_p$   // Save input to suffix step
 9      // PREFIX STEP
        // Reduced up to $i$ dimensions.
10      PREFIX-ALONG-DIM along
11        dimension $i$ on $\mathcal{A}_p$.
12      $\mathcal{A} \leftarrow \mathcal{A}_p$   // Save for next round
13      // Do included sum on dimensions $[i+1, d]$.
14      for $j \leftarrow i+1$ to $d$
15          // $\mathcal{A}_p$ reduced up to $i$ dimensions
16          BDBS-ALONG-DIM on
17            dimension $j$ in $\mathcal{A}_p$
18      // Add into result
19      ADD-CONTRIBUTION from $\mathcal{A}_p$ into $\mathcal{A}'$
20
21      // SUFFIX STEP
        // Do suffix sum along dimension $i$
22      SUFFIX-ALONG-DIM along
23        dimension $i$ in $\mathcal{A}_s$
24      // Do included sum on dimensions $[i+1, d]$
25      for $j \leftarrow i+1$ to $d$
26          // $\mathcal{A}_s$ reduced up to $i$ dimensions
27          BDBS-ALONG-DIM on
28            dimension $j$ in $\mathcal{A}_s$
29      // Add into result
30      ADD-CONTRIBUTION from $\mathcal{A}_s$ into $\mathcal{A}'$
31  return $\mathcal{A}'$
```

Figure 9: Pseudocode for the box-complement algorithm. For ease of presentation, we omit the exact parameters to the subroutines and describe their function in the algorithm.

6 Experimental Evaluation

This section presents an empirical evaluation of strong and weak excluded-sums algorithms. In 3 dimensions, we compare strong excluded-sums algorithms: specifically, we evaluate the box-complement algorithm and variants of the corners algorithm and find that the box-complement outperforms the corners algorithm given similar space. Furthermore, we compare weak excluded-sums algorithms in higher dimensions. Lastly, to simulate a more expensive operator than numeric addition when reducing, we introduce an artificial slowdown in the binary associative operator in 3D. The full version of the paper includes details of the experimental setup and additional results [16].

Strong Excluded Sums in 3D Figure 3 summarizes the results of our evaluation of the box-complement and corners algorithm in 3 dimensions with a box length of $k_1 = k_2 = k_3 = 4$ (for a total box volume of $K = 64$) and number of elements $N = 681472 = 88^3$. We tested with varying N but found that the time and space per element were flat. We found that the box-complement algorithm outperforms the corners algorithm by about 1.4× when given similar amounts of space, though the corners algorithm with 2× the space as the box-complement algorithm was 2× faster.

We explored two different methods of using extra space in the corners algorithm based on the computation tree of prefixes and suffixes: (1) storing the spine of the computation tree to asymptotically reduce the running time, and (2) storing the leaves of the computation tree to reduce passes through the output. Although storing the

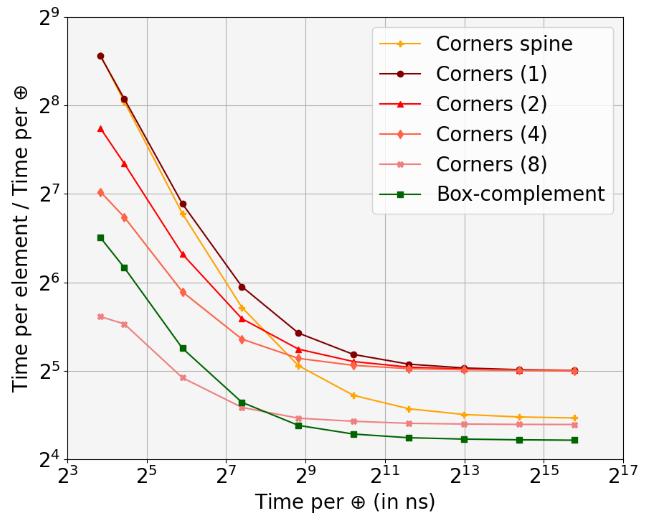

Figure 10: The scalability of excluded-sum algorithms as a function of the cost of operator $\oplus$ on a 3D domain of $N = 4096$ elements. The horizontal axis is the time in nanoseconds to execute $\oplus$. The vertical axis represents the time per element of the given algorithm divided by the time for $\oplus$. We inflated the time of $\oplus$ using increasingly large arguments to the standard recursive implementation of a Fibonacci computation.

leaves does not asymptotically affect the behavior of the corners algorithm, we found that reducing the number of passes through the output has significant effects on empirical performance. Storing the spine did not improve performance, because the runtime is dominated by the number of passes through the output.

Excluded Sums With Different Operators Most of our experiments used numeric addition for the $\oplus$ operator. Because some applications, such as FMM, involve much more costly $\oplus$ operators, we studied how the excluded-sum algorithms scale with the cost of $\oplus$. To do so, we added a tunable slowdown to the invocation of $\oplus$ in the algorithms. Specifically, they call an unoptimized implementation of the standard recursive Fibonacci computation. By varying the argument to the Fibonacci function, we can simulate $\oplus$ operators that take different amounts of time.

Figure 10 summarizes our findings. For inexpensive $\oplus$ operators, the box-complement algorithm is the second fastest, but as the cost of $\oplus$ increases, the box-complement algorithm dominates. The reason for this outcome is that box-complement performs approximately 12 $\oplus$ operations per element in 3D, whereas the most efficient corners algorithm performs about 22 $\oplus$ operations. As $\oplus$ becomes more costly, the time spent executing $\oplus$ dominates the other bookkeeping overhead.

Weak Excluded Sums in Higher Dimensions Figure 4 presents the results of our evaluation of weak excluded-sum algorithms in higher dimensions. For all dimensions $i = 1, 2, \ldots, d$, we set the box length $k_i = 8$ and chose a number of elements N to be a perfect power of the dimension i. Table 1 presents the asymptotic runtime of the different excluded-sum algorithms.

The weak naive algorithm for excluded sums with nested loops outperforms all of the other algorithms up to 2 dimensions because its runtime is dependent on the box volume, which is low in smaller dimensions. Since its runtime grows exponentially with the box length, however, we limited it to 5 dimensions.

The summed-area table algorithm outperforms the BDBS and box-complement algorithms up to 6 dimensions, but its runtime scales exponentially in d.

Finally, the BDBS and box-complement algorithms scale linearly in the number of dimensions and outperform both naive and summed-area table methods in higher dimensions. Specifically, the box-complement algorithm outperforms the summed-area table algorithm by between 1.3× and 4× after 6 dimensions. The BDBS algorithm demonstrates an advantage to having an inverse: it outperforms the box-complement algorithm by 1.1× to 4×. Therefore, the BDBS algorithm dominates the box-complement algorithm for weak excluded sums.

7 Conclusion

In this paper, we introduced the box-complement algorithm for the excluded-sums problem, which improves the running time of the state-of-the-art corners algorithm from $\Omega(2^d N)$ to $\Theta(dN)$ time. The space usage of the box-complement algorithm is independent of the number of dimensions, while the corners algorithm may use space dependent on the number of dimensions to achieve its running-time lower bound.

Acknowledgments

The idea behind the box-complement algorithm was originally conceived jointly with Erik Demaine many years ago, and we thank him for helpful discussions. Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37, 1997.

[2] Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.

[3] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, 12(2):13–21, 2007.

[4] Hongwei Cheng, William Y. Crutchfield, Zydrunas Gimbutas, Leslie F. Greengard, J. Frank Ethridge, Jingfang Huang, Vladimir Rokhlin, Norman Yarvin, and Junsheng Zhao. A wideband fast multipole method for the Helmholtz equation in three dimensions. *Journal of Computational Physics*, 216(1):300–325, 2006.

[5] Ronald Coifman, Vladimir Rokhlin, and Stephen Wandzura. The fast multipole method for the wave equation: A pedestrian prescription. *IEEE Antennas and Propagation Magazine*, 35(3):7–12, 1993.

[6] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, page 207–212, New York, NY, USA, 1984. Association for Computing Machinery.

[7] Eric Darve. The fast multipole method: numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

[8] E. D. Demaine, M. L. Demaine, A. Edelman, C. E. Leiserson, and P. Persson. Building blocks and excluded sums. *SIAM News*, 38(4):1–5, 2005.

[9] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 135(2):280–292, August 1997.

[10] Nail A. Gumerov and Ramani Duraiswami. *Fast multipole methods for the Helmholtz equation in three dimensions*. Elsevier, 2005.

[11] Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast summed-area table generation and its applications. In *Computer Graphics Forum*, volume 24, pages 547–555. Wiley Online Library, 2005.

[12] Nicholas J. Higham. The accuracy of floating point summation. *SIAM J. Scientific Computing*, 14:783–799, 1993.

[13] William B March and George Biros. Far-field compression for fast kernel summation methods in high dimensions. *Applied and Computational Harmonic Analysis*, 43(1):39–75, 2017.

[14] Xiaobai Sun and Nikos P Pitsianis. A matrix version of the fast multipole method. *Siam Review*, 43(2):289–300, 2001.

[15] Ernesto Tapia. A note on the computation of high-dimensional integral images. *Pattern Recognition Letters*, 32(2):197–201, 2011.

[16] Helen Xu, Sean Fraser, and Charles E. Leiserson. Multidimensional included and excluded sums, 2021. `https://arxiv.org/abs/2106.00124`.

[17] Gernot Ziegler. Summed area computation using ripmap of partial sums, 2012. GPU Technology Conference (talk).