Wavelet-Based Relative Prefix Sum Methods for Range Sum Queries in Data Cubes*

Daniel Lemire
National Research Council of Canada

Abstract

Data mining and related applications often rely on extensive range sum queries and thus, it is important for these queries to scale well. Range sum queries in data cubes can be achieved in time $O(1)$ using prefix sum aggregates but prefix sum update costs are proportional to the size of the data cube $O\left(n^d\right)$. Using the Relative Prefix Sum (RPS) method, the update costs can be reduced to the root of the size of the data cube $O\left(n^{d/2}\right)$. We present a new family of base b wavelet algorithms further reducing the update costs to $O\left(n^{d/\beta}\right)$ for β as large as we want while preserving constant-time queries. We also show that this approach leads to $O\left(\log^d n\right)$ query and update methods twice as fast as Haar-based methods. Moreover, since these new methods are pyramidal, they provide incrementally improving estimates.

1 Introduction

Computational scalability with respect to both data queries and data updates is a key requirement in On-Line Analytical Processing (OLAP). Moreover, for OLAP applications, it is desirable to have low query costs. However, one should distinguish between OLAP and Precomputed Analytic Processing (PAP) systems [12]: a system with very low query costs which is expensive to update or initialize might not be practical for OLAP systems where data updates are frequent and new databases are created dynamically. On the other hand, a slightly longer query time might be acceptable as long as approximations are quickly available and as long as the aggregates can be quickly updated. Reaching a good compromise with respect to the various performance measures is more important than merely having fast queries.

Data cube [9] is a common data model used by data warehouses and can be simply described as a large multidimensional array which can be implemented as such or as an interface to a (relational) database. Consider the following simple data cube example with three dimensions ($d = 3$): age, income, weight. Each cell in the data cube D (e.g. 17 years old, 10k$, 60 kg) contains a measure $D_{i,j,k}$. For example, cells may contain the number of individuals matching the criteria of the cell (e.g. $D_{17,10k\$,60kg} = 100$). A range sums query amounts to summing up the measures $\sum_{i,j,k \in I} D_{i,j,k}$ in a set of cell indices I. For example, one could ask the number m of individuals between the ages of 18 and 25, with an income above 20 k$ and a weight over 100 kg. The simplest way to do this computation is to sum over all cells. Assuming that in the age, income and weight model, we have all ages from 0 to 100 years, all incomes from 0 k$ to 100 k$ in increments of 1 k$ and all weights from 40 kg to 200 kg in increment of 1 kg, a range sum query might involve $100 \times 100 \times 160$ individual sums which amounts to well over 1 million operations. Clearly, this approach does not scale well with respect to queries.

In the context of this paper, we measure cost in terms of the number of cells we need to read or change in a data cube in the worst possible case. We assume that the data cubes have dimension d and that the size of the data cube (number of cells) is n^d for some large n and a fixed dimension d. Also, for simplicity, we assume that data cubes are indexed from 0 to n in all dimensions.

With OLAP applications in mind, "prefix sum" method was developed and it can resolve such range sum queries in time $O(1)$ [13]: one creates a new data cube $\widetilde{D}$ of the same size containing the the sums of the measures "up to", $\widetilde{D}_{i,j,k} =$

*Proc. of CASCON 2002, Toronto, Canada, October 2002. NRC 44967.

$\sum_{r\le i,s\le j,t\le k} D_{r,s,t}$ (see Tables 1 and 2 for a two-dimensional example). It can be seen that the computational cost for $\widetilde{D}$ is proportional to the size of the data cube itself. However, given a set of consecutive indices $I = [i_b^1, i_e^1] \times [i_b^2, i_e^2] \times [i_b^3, i_e^3]$, we can compute the range sum in time $O(1)$. Indeed, let $\iota_k(x) = x(i_b^k - i_e^k) + i_e^k$, then the range sum over I on D is given by

$$\sum_{i,j,k\in I} D_{i,j,k} = \sum_{r=0}^{1}\sum_{s=0}^{1}\sum_{t=0}^{1}(-1)^{r+s+t}\widetilde{D}_{\iota_1(r),\iota_2(s),\iota_3(t)},$$

and, more generally, if we note the prefix sums as

$$S_{i_1,\ldots,i_d} = \sum_{r_1\le i_i,\ldots,r_d\le i_d} D_{r_1,\ldots,r_d}, \qquad (1)$$

we have

$$S_{i_1,\ldots,i_d} = \sum_{r_1=0}^{1}\cdots\sum_{r_d=0}^{1}(-1)^{r_1+\ldots+r_d}\widetilde{D}_{\iota_1(r_1),\ldots,\iota_d(r_d)} \qquad (2)$$

so that, in general, a sum over up to n^d cells can be computed with 2^d sums irrespective of n using the prefix sum data cube $\widetilde{D}$. For static databases, this approach may well be optimal. Unfortunately, the prefix sum approach has a major drawback: updating the value of a single cell in the data cube D means updating up to n^d cells in the prefix sum data cube $\widetilde{D}$.

3	5	1	2	2	4	6	3	4
7	3	2	6	8	7	1	2	1
2	4	2	3	3	3	4	5	4
3	2	1	5	3	5	2	8	0
4	2	1	3	3	4	7	1	3
2	3	3	6	1	8	5	1	2
4	5	2	7	1	9	3	3	1
2	4	2	2	3	1	9	1	5
1	3	1	1	2	2	8	2	6

Table 1: A randomly chosen 9×9 data cube D. By convention indices start at (0,0) in the upper left corner and go up to $(8,8)$ in the lower right corner.

The Relative Prefix Sums (RPS) method [8] ease the update burden. Instead of building a large prefix sum over all of the data, the RPS method builds local prefix sums. In order to still offer fast queries, an "overlay" is used to record the cumulative prefix sums. Let $\lfloor x\rfloor$ be the greatest integer smaller or equal to x so that $\eta\lfloor i/\eta\rfloor$ is i rounded off to a multiple of η. In the unidimensional case, given n cells x_i with $i = 1,\ldots,n$, the corresponding Prefix

3	8	9	11	13	17	23	26	30
10	18	21	29	39	50	57	62	67
12	24	29	40	53	67	78	88	97
15	29	35	51	67	86	99	117	126
19	35	42	61	80	103	123	142	154
21	40	50	75	95	126	151	171	185
25	49	61	93	114	154	182	205	220
27	55	69	103	127	168	205	229	249
28	59	74	109	135	178	223	249	275

Table 2: Prefix sum for the data cube of Table 1. Note that if the upper left value is changed in Table 1, all of this table must be updated (81 cells).

Sum (PS) cells y_i contain the sums $y_i = \sum_{k=\eta\lfloor i/\eta\rfloor}^{i} x_k$ where η is the overlay box size. On the other hand, the unidimensional overlay cube contains the sums $\sum_{k=0}^{\eta\times i} x_k$. It can be seen that the RPS method has query cost $O(1)$ and an update cost that depends on η. By choosing $\eta = \sqrt{n}$, it can be shown that the update cost is $O\left(n^{d/2}\right)$ and this is the best possible choice. An example is given in Table 4 and the RPS algorithm is given next (see [8] for more details). The current work is a generalization of this approach into a pyramidal setting (see Table 3 for a comparison).

method	Query cost	Update cost
Data Cube [9]	$O\left(n^d\right)$	$O(1)$
Prefix Sum [13]	$O(1)$	$O\left(n^d\right)$
RPS [8]	$O(1)$	$O\left(n^{d/2}\right)$
Haar[18]	$O\left(n^d\right)$	$O(\log_2 n)$
HBC [3]	$O(1)$	$O\left(n^{d-1}\right)$
PyRPS	$O(1)$	$O\left(n^{d/\beta}\right)$, $\beta = 1,2,\ldots$
PyRPS (log)	$O\left(\log_b^d n\right)$	$O\left((b-1)^d \log_b^d n\right)$

Table 3: Comparison the PyRPS method described in section 4 and the PyRPS(log) from subsection 4.1 with some other range sum aggregates for a data cube of size n^d with large n.

Algorithm 1 *(Relative Prefix Sum) Given a data cube $D_{i_1,\ldots,i_d}$ of size n^d and a set overlay box of size η, the prefix sum array PR is given by*

$$PR_{i_1,\ldots,i_d} = \sum_{k_1=\eta\lfloor i_1/\eta\rfloor}^{i_1}\cdots\sum_{k_d=\eta\lfloor i_d/\eta\rfloor}^{i_d} D_{k_1,\ldots,k_d}.$$

Let $\sigma(i) = \{0,\ldots,i\}$ if i modulo $\eta = 0$ and $\{\eta\lfloor i/\eta\rfloor + 1,\ldots,i\}$ otherwise. The overlay array

Ω *is given by*

$$\Omega_{i_1,\ldots,i_d} = \sum_{\substack{k_1 \in \sigma(i_1),\ldots,k_d \in \sigma(i_1) \\ (k_1,\ldots,k_d) \neq (i_1,\ldots,i_d)}} D_{k_1,\ldots,k_d}$$

for all tuples $(i_1,\ldots,i_d)$ *where at at least one component* i_k *is a multiple of* η. *The algorithm returns both the prefix sum array PR and the overlay array* Ω.

More recently, Hierarchical Band Cube (HBC) have been proposed [3] as an alternative to the RPS method: but HBC data updates require time $O\left(n^{d-1}\right)$. However, to our knowledge, the HBC method is the first to generalize the RPS method in terms of base b trees and it does away with the overlay. Other authors have used base 2 (dyadic) Haar wavelets [4, 10] either to compress or approximate prefix sums aggregates [14, 16, 17] or as a replacement for the prefix sum method [18]. An important limitation of these wavelet-based methods is that they have polylogarithmic query times.

3	8	9	2	4	8	6	9	13
10	18	21	8	18	29	7	12	17
12	24	29	11	24	38	11	21	30

3	5	6	5	8	13	2	10	10
7	11	13	8	14	23	9	18	21
9	16	21	14	21	38	14	24	29

4	9	11	7	8	17	3	6	7
6	15	19	9	13	23	12	16	22
7	19	24	10	16	28	20	26	38

0	0	0	9	0	0	17	0	0
0			12			33		
0			20			50		
12	12	17	46	13	27	97	10	19
0			7			17		
0			15			40		
21	19	29	86	20	51	179	20	34
0			8			14		
0			13			24		

Table 4: The Relative Prefix (RP) array and its overlay for the data cube of Table 1. The overlay size was chosen to be $\sqrt{9} = 3$.

Assuming that range sums aggregates cannot use a buffer larger than the data cube itself, we are motivated by the following questions:

1. Can wavelets allow queries in time $O(1)$ as the prefix sum methods does?
2. Assuming we require that queries be processed in time $O(1)$, how cheap can the updates be? Can we improve on the RPS method?
3. What is the best wavelet for range sum problems?

The answer to the first question is positive: it suffices to use wavelets in base b so that the height of the wavelet tree is $\log_b n$. $B-$adic wavelets were first introduced by Heller [11] as rank M wavelets, and are related to $b-$adic subdivision scheme [5, 6]. By choosing b large enough, we can effectively control the height of the tree. This allows us to limit the worst-case query cost to a set maximum which depends on the height of the tree. This base b approach is similar to replacing binary trees by $B-$trees as a most scalable alternative. As for the second question, we show that we can generalize the RPS method as a wavelet-like approach and improve the scalability of the method as much as we want: the update costs scales as $O\left(n^{d/\beta}\right)$ with β as a parameter. Finally, as for the third question, the dyadic Haar wavelet is probably not the best choice for many range sum problems. Instead, we present a special case of Aldroubi's *oblique wavelets* [1] as an alternative. Oblique wavelets can be described as "stripped down wavelets" that can be computed twice as fast while still providing a wavelet tree. Most wavelets found in commercial software packages are orthogonal or biorthogonal wavelets [4] (Haar, Daubechies, ...) because it assumed that the data is smooth or that the user is interested in the Euclidean norm of the wavelet coefficients and its relation to the Euclidean norm of the signal (e.g. Riesz property). For the application at hand, range sum queries, the main relevant "wavelet feature" is the "multiscale" or pyramidal approach [7]. Of course, for higher order range queries such as variance and covariance queries, other wavelets might be a better choice [15].

Our main result is a generalization of the RPS into a Pyramidal Relative Prefix Sum (PyRPS) approach which is shown to be arbitrarily scalable. We also present adapted $b-$adic oblique wavelets as a preferred wavelet transform for range sum problems.

This paper is organized as follow. We begin with a review of unidimensional wavelets. We present efficient, in-place algorithms to compute the oblique wavelet transform and we conclude with some computational complexity analysis. We then show how these results can be generalized in

the multidimensional case using the direct product. The wavelet-based Pyramidal Relative Prefix Sum approach is then presented and analyzed. We conclude with some optimization analysis for the polylogarithmic case, a discussion on variable bases, and some remarks concerning practical implementations.

2 Unidimensional Wavelets

The basic idea of a wavelet transform is to project the data on a simpler, coarser scale while measuring the error made so that the transformation can be reversed [4]. By projecting repeatedly the data on coarser and coarser scales, a wavelet tree is built and provides a "mathematical zoom". The Haar transform [10] is probably the first example of a wavelet transform. Let $\Xi = \{x_i\}$ be an array of length 2^J with indices $I_J = \{0,\ldots,2^J-1\}$. In one dimension and with a convenient normalization, the Haar transform can be described with two operators: the coarse-scale projection

$$P_{Haar}(\Xi) = \{x_{2i} + x_{2i+1}\}_{i \in I_{J-1}}$$

and the error measure (or "wavelet coefficients")

$$Q_{Haar}(\Xi) = \{x_{2i} - x_{2i+1}\}_{i \in I_{J-1}}.$$

Both P_{Haar} and Q_{Haar} downsample the data by a factor of 2 so we say that the transformation is dyadic (base 2) and given the result of both P_{Haar} and Q_{Haar}, we can recover the original data x_i. The tree structure comes in when you repeatedly apply the operators P_{Haar} and Q_{Haar} to the results of the coarse scale operator P_{Haar}. For example, given the data $\{a,b,c,d\}$, we get $P_{Haar}(\{a,b,c,d\}) = \{a+b, c+d\}$ and $Q_{Haar}(\{a,b,c,d\}) = \{a-b, c-d\}$ and then $P_{Haar}(P_{Haar}(\{a,b,c,d\})) = \{a+b+c+d\}$ and $Q_{Haar}(P_{Haar}(\{a,b,c,d\})) = \{a+b-c-d\}$. The final wavelet tree with height 2 is given by the coarse scale projection $\{a+b+c+d\}$, its corresponding wavelet coefficient $\{a+b-c-d\}$ and finer scale wavelet coefficients $\{a+b, c+d\}$ and $\{a-b, c-d\}$. Wavelet coefficients at step j are given by $Q_{Haar}(P_{Haar})^{j-1}(\Xi)$. The lowest wavelet coefficients in the tree are $Q_{Haar}(\Xi)$, then $Q_{Haar}(P_{Haar}(\Xi))$, $Q_{Haar}(P_{Haar}(P_{Haar}(\Xi)))$, and so on. Because of the downsampling, an array of length n will have $n/2$ wavelet coefficients at the bottom, then $n/4$ at the second step and $n/2^j$ at step j ($j = 1,2,\ldots$).

As an alternative, we propose to use oblique wavelets [1] defined by the following operators: the coarse-scale projection (as in Haar)

$$P_2(\Xi) = \{x_{2i} + x_{2i+1}\}_{i \in I_{J-1}}$$

and the error measure (or "wavelet coefficients")

$$Q_2(\Xi) = \{x_{2i}\}_{i \in I_{J-1}}.$$

One major drawback of such simpler transform is that it doesn't preserve the Euclidean (l_2) norm or the energy of the data unlike a properly normalized Haar transform and so, some authors [7], prefer to refer to such a transform as a "pyramidal transform" rather than a "wavelet transform".

For $b > 2$, let $\Xi = \{x_i\}$ be an array of length b^J with indices $I_J = \{0,\ldots,b^J-1\}$. We propose to generalize the operators P_2 and Q_2 to the equivalent base $b > 2$ case:

$$P_b(\Xi) = \{x_{bi} + x_{bi+1} + \ldots + x_{bi+b-1}\}_{i \in I_{J-1}}$$

and

$$\begin{gathered} Q_{b,1}(\Xi) = \{x_{bi}\}_{i \in I_{J-1}}, \\ Q_{b,2}(\Xi) = \{x_{bi} + x_{bi+1}\}_{i \in I_{J-1}}, \\ \ldots \\ Q_{b,b-1}(\Xi) = \{x_{bi} + x_{bi+1} + \ldots + x_{bi+b-2}\}_{i \in I_{J-1}}. \end{gathered}$$

These new operators downsample the original data by a factor of b and so are said to be $b-$adic but since we have b linearly independent operators, they still allow perfect reconstruction (see proof of proposition 1). Notice that subtraction is never used as the operators are effectively local prefix sums. The following algorithm can be used to efficiently compute the transform; for the sake of simplicity, we assume that n is a power of b.

Algorithm 2 *(In-Place Base b Oblique Wavelet Transform) Given an array of values x_i, $i = 0,\ldots,n-1$, the first step is given by*

$$x_{bk} \leftarrow x_{bk},$$

$$x_{bk+1} \leftarrow x_{bk} + x_{bk+1}, \ldots,$$

$$x_{bk+b-1} \leftarrow x_{bk+b-1} + x_{bk+b-2}$$

for $k = 0,\ldots,n/b-1$ so that we have $x_i \rightarrow \sum_{k=b\lfloor (i+1)/b \rfloor}^{i} x_k$ for $i = 0,\ldots,n-1$. Similarly, successive steps at depth $j = 1,\ldots,\log_b n - 1$ are given by

$$x_{b^j k + b^j - 1} \leftarrow x_{b^j k + b^j - 1},$$

$$x_{b^j k+2b^j-1} \leftarrow x_{b^j k+2b^j-1} + x_{b^j k+b^j-1}, \ldots,$$

$$x_{b^{j+1}k-1} \leftarrow x_{b^{j+1}k-1} + x_{b^{j+1}k-b^j-1}$$

for $k = 0, \ldots, n/b^j - 1$ *so that we have*

$$x_i \rightarrow \sum_{k=\gamma(i,j)}^{i} x_k$$

for $i = b^j - 1, 2b^j - 1, \ldots, n$ *and* $\gamma(i,j) = b^j \times \left\lfloor (i+1)/b^j \right\rfloor - 1$.

The computational cost of this algorithm is $O(n)$. Indeed, the number of sums can be seen to be $\frac{b-1}{b}n + \frac{b-1}{b^2}n + \ldots + b - 1 = n - 1$ assuming that n is a power of b. Consider that the Haar transform involves $n + \frac{n}{2} + \ldots + 2 = 2n - 2$ sums and subtractions and is therefore more expensive by a factor of 2. As an example of this algorithm, suppose we apply the in-place transform to the array $D = \{1,0,2,1,2,4,3,1,3\}$ with $b = 3$. The first step gives $D_1 = \{1,1,3,1,3,7,3,4,7\}$ whereas the second and last step gives $D_2 = \{1,1,\mathbf{3},1,3,\mathbf{10},3,4,\mathbf{17}\}$ where only the sixth and the last data samples changed. Since the tree has height 2 (2-step transform), the sum of the first k cells in D can always be computed using at most 2 sums in D_2. For example, to compute the sum of the first 8 terms in D, it suffices to sum the 6^{th} and the 8^{th} term in D_2: $10 + 4 = 14$. The proof of the next proposition gives the general formula to compute such range sums. We note the transformation described by Algorithm 2 as Γ_b so that

$$\Gamma_3(1,0,2,1,2,4,3,1,3) = \{1,1,3,1,3,10,3,4,17\}.$$

Proposition 1 *Given the base b oblique transform* $y_{i=0,\ldots,n-1} = \Gamma_b(x_{i=0,\ldots,n-1})$*, then one can compute any range sum of the form* $S_k = \sum_{i=0}^{k} x_i$ *in time* $O(\log_b n)$.

Proof. The proof is constructive. Let $\gamma(k,j) = b^j \times \left\lfloor (k+1)/b^j \right\rfloor - 1$ and assume $n = 2^J$, then it suffices to observe that

$$S_k = y_k + y_{\gamma(k,1)} + \cdots + y_{\gamma(k,J-1)} \qquad (3)$$

where we used the convention that the same cells are never summed twice and negative indices are ignored e.g. $S_{2b-1} = y_{2b-1}$. We see that at most $J = \log_b n$ sums are required to compute S_k. □

In practical applications, the sum in equation 3 could be approximated by the few last terms since they involve many more cells in the original data cube and are likely to provide an incrementally improving estimate. This is particularly convenient if the topmost cells in the tree are buffered for fast access. Similarly, if one cell is updated, then no more than $1 + (b-1)\log_b n$ cells need to be updated in the transformed array as the proof of the next proposition shows. This is true for all base b wavelet transform even though we are only concerned with the transform described by Algorithm 2. We define the height (or depth) of an index in the tree by $height(k) = 1 + \max\{j \in \mathbb{N} : \gamma(k,j) = k\}$ where $\gamma(k,j) = b^j \times \left\lfloor (k+1)/b^j \right\rfloor - 1$. This definition might seem inefficient if we try to test all j values (up to $\log_b n$). Similarly, the sum of equation 3 must be evaluated while checking for redundant terms which might appear difficult. However, the following proposition shows that both problems are easily taken care of essentially because γ is monotone decreasing.

Proposition 2 *If* $\gamma(k,j) = k$*, then* $\gamma(k,\nu) = k$ *for all* $\nu < j$ *and* $\gamma(k,\alpha)$ *is monotone decreasing in* α.

Proof. Notice that $b^j \times \left\lfloor (k+1)/b^j \right\rfloor \leq k+1$, hence $\gamma(k,j) \leq k$ for $j \geq 0$. By Euler's theorem, given b and j, there exist integers $r \geq 0$ and $0 \leq s < b^j$ such that $k+1 = b^j r + s$. Assume that $\gamma(k,j) = k$ for some $j > 0$, then $b^j r - 1 = k$ and $\gamma(k,j-1) = b^j r - 1 + \left\lfloor s/b^j \right\rfloor = k + \left\lfloor s/b^j \right\rfloor$, but because $\gamma(k,j-1) \leq k$, we have $\gamma(k,j-1) = k$. The first result follows by finite induction.

To show the second result, we observe that $b^j \left\lfloor x/b^j \right\rfloor \leq \lfloor x \rfloor$ and thus, setting $x = (k+1)/b^t$ for some $t > 0$, we have $b^j \left\lfloor \frac{k+1}{b^{j+t}} \right\rfloor \leq \left\lfloor \frac{k+1}{b^t} \right\rfloor$ or $b^{j+t} \left\lfloor \frac{k+1}{b^{j+t}} \right\rfloor \leq b^t \left\lfloor \frac{k+1}{b^t} \right\rfloor$ hence $\gamma(k,j+t) \leq \gamma(k,t)$ which shows that γ is monotone. □

Finally, this $O(n)$ transform can be updated with only a logarithmic cost as the following proposition states.

Proposition 3 *The base* $b \geq 2$ *oblique transform* $y_{i=0,\ldots,n-1} = \Gamma_b(x_{i=0,\ldots,n-1})$*, can be updated in time* $O((b-1)\log_b n)$ *given a change in one cell value* x_k.

Proof. A change in cell x_k requires at most $b-1$ update at each depth in the wavelet tree except for the topmost cells where b cells might need to be updated. A maximum of $b+$

$(b-1)(\log_b n - 1)$ cells may need to be updated: $y_k, \ldots, y_{\gamma(k+b,1)-1}$; $y_{\gamma(k+b,1)}$, $y_{\gamma(k+2b,1)}, \ldots$, $y_{\gamma(k+b^2,2)-b}$; $\ldots$; $y_{\gamma(k+b^{J-1},J-1)}$, $y_{\gamma(k+2b^{J-1},J-1)}$, $\ldots$, y_{n-1}. If the value stored in cell x_k was increased (resp. decreased) by Δy, it suffices to add (resp. subtract) Δy to each cell. □

3 Multidimensional Wavelets

In the multidimensional case, we take the wavelet transform using the direct product of the unidimensional wavelet transform $\Gamma_b \otimes \cdots \otimes \Gamma_b$ on a data cube of size b^{Jd}. In practical terms, this suggests that we apply the transform on each dimension separately. For example, given a data cube $D_{i_1,\ldots,i_d}$, $0 \leq i_k < n$ of dimension d, the wavelet transform can be computed using d steps. Firstly, we apply the operator Γ_b on the first dimension n^{d-1} times: we have n^{d-1} arrays $a_{i_1}(i_2,\ldots,i_d)$ indexed by i_1 and defined by $a_{i_1}(i_2,\ldots,i_d) = D_{i_1,\ldots,i_d}$. Let $a'_{i_1} = \Gamma_b(a_{i_1})$ for all possible values of $i_2,\ldots,i_d$ and set $D^{(1)}_{i_1,\ldots,i_d} = a'_{i_1}(i_2,\ldots,i_d)$. We repeat this process with each dimension and note the result $\Gamma_b \otimes \cdots \otimes \Gamma_b(D)$. It can be seen that this algorithm has a cost of $O\left(n^d\right)$. A two-dimensional example is given in Table 5. We define the height of a cell $(i_1,\ldots,i_d)$ in the tree by $height(i_1,\ldots,i_d) = \min\{height(i_1),\ldots height(i_d)\}$.

The computational cost to range sum queries is the same as the computational cost for computing prefix sums, $S_{i_1,\ldots,i_d}$ (see equation 2). However, by proposition 1, prefix sums can be computed in time $\log_b n$ assuming we have applied the wavelet transform Γ_b. A similar result applies in the multidimensional case.

Proposition 4 *Given the base b oblique transform of a data cube of size n^d, D^{Γ_b}, then one can compute range sums $S_{i_1,\ldots,i_d}$ (prefix sums) in time $O\left(\log_b^d n\right)$.*

Proof. Let $\gamma(k,j) = b^j \times \left\lfloor (k+1)/b^j \right\rfloor - 1$ and consider the operator $\sigma_k(a) = a_k + a_{\gamma(k,1)} + \ldots + a_{\gamma(k,J-1)}$. By the proof of proposition 1, the prefix sums can be computed by the wavelet transform $\sum_{k \leq j} a_k = \sigma_k \circ \Gamma_b(a_k)$ for any array a. Thus if we note $D^{\Gamma_b} = \Gamma_b \otimes \cdots \otimes \Gamma_b(D)$, then

$$\begin{aligned} S_{i_1,\ldots,i_d} &= \sum_{r_1 \leq i_1,\ldots,r_d \leq i_d} D_{r_1,\ldots,r_d} \\ &= (\sigma_{i_1} \circ \Gamma_b) \otimes \cdots \otimes \left(\sigma_{i_d} \circ \Gamma_b\right)(D) \\ &= \sigma_{i_1} \otimes \cdots \otimes \sigma_{i_d}\left(\Gamma_b \otimes \cdots \otimes \Gamma_b(D)\right) \\ &= \sigma_{i_1} \otimes \cdots \otimes \sigma_{i_d}\left(D^{\Gamma_b}\right) \\ &= \sum_{r_1=0}^{J-1} \cdots \sum_{r_d=0}^{J-1} D^{\Gamma_b}_{\gamma(r_1,j),\ldots,\gamma(r_d,j)} \end{aligned}$$

which can be done in time $O\left(\log_b^d n\right)$ since each sum involves at most $\log_b n$ terms. □

The proof of the previous proposition gives us the formula to compute prefix sums. As an example, consider Table 5 where $n=9$, $d=2$, $b=3$ and $J=2$. Since $\gamma(7,1)=5$, the prefix sum at position $(7,7)$ is given by

$$\begin{aligned} \sum_{r_1=7,5}\sum_{r_2=7,5} D^{\Gamma_3}_{r_1,r_2} &= D_{7,7} + D_{7,5} + D_{5,7} + D_{5,5} \\ &= 16 + 45 + 42 + 126 = 229 \end{aligned}$$

and the result can be checked in Table 2. As another example, we can compute the prefix sum at position $(7,1)$ with the convention that the first index refers to the row number. We have that $\gamma(1,1) = -1$ and $\gamma(7,1) = 5$, thus the prefix sum is given by

$$\begin{aligned} \sum_{r_1=1}\sum_{r_2=7,5} D^{\Gamma_3}_{r_1,r_2} &= D_{1,7} + D_{1,5} \\ &= 15 + 40 = 55. \end{aligned}$$

At most $2 \times 2 = 4$ sums are required to compute the prefix sum at any cell as predicted by proposition 4. Similarly, if one were to change the value of cell $(0,0)$, then only 25 cells need to be updated (see Table 6). The following proposition makes this result general. As in the unidimensional case, these sums can be seen as incrementally accurate estimates: assuming we stop short of the first dL terms in the sum, the approximation $\sum_{r_1=L}^{J-1} \cdots \sum_{r_d=L}^{J-1} D^{\Gamma_b}_{\gamma(r_1,j),\ldots,\gamma(r_d,j)}$ and the actual value $S_{i_1,\ldots,i_d}$ differ at most by the sum of $dn^{d-1}(b^L-1)$ cells. If the values in the data cube are bounded in absolute value by M, the error in skipping the last dL sums is at most $Mdn^{d-1}(b^L-1)$ (exponential decay as $L \to 0$).

Proposition 5 *The base b oblique transform $\Gamma_b \otimes \cdots \otimes \Gamma_b(D)$ of a data cube of size n^d can be updated in time $O((b-1)^d \log_b^d n)$ given a change in one cell value.*

Proof. Since the operator Γ_b can be applied dimension per dimension, we can simply count the number of cells affected by the change. By the

proof of proposition 3, only $b+(b-1)(\log_b n-1)$ cells are affected after the transform on the first dimension is done. Each on these cells, in turn, impact on at most $b+(b-1)(\log_b n-1)$ cells across the second dimension. Thus after d transform, at most $(b+(b-1)(\log_b n-1))^d$ cells are impacted across all dimensions. □

3	8	9	2	4	17	6	9	30
10	18	21	8	18	50	7	12	67
12	24	29	11	24	67	11	21	97
3	5	6	5	8	19	2	10	29
7	11	13	8	14	36	9	18	57
21	40	50	25	45	126	25	45	185
4	9	11	7	8	28	3	6	35
6	15	19	9	13	42	12	16	64
28	59	74	35	74	178	45	71	275

Table 5: Oblique Transform $\Gamma_3 \otimes \Gamma_3(D)$(PyRPS) for the data cube D of Table 1. All cells are at height 1 in the wavelet tree except the 9 cells with a darker gray background which are at height 2.

3.1	8.1	9.1	2	4	17.1	6	9	30.1
10.1	18.1	21.1	8	18	50.1	7	12	67.1
12.1	24.1	29.1	11	24	67.1	11	21	97.1
3	5	6	5	8	19	2	10	29
7	11	13	8	14	36	9	18	57
21.1	40.1	50.1	25	45	126.1	25	45	185.1
4	9	11	7	8	28	3	6	35
6	15	19	9	13	42	12	16	64
28.1	59.1	74.1	35	74	178.1	45	71	275.1

Table 6: Oblique Transform $\Gamma_3 \otimes \Gamma_3(D)$ (PyRPS) for the data cube D of Table 1 modified with $D_{0,0} = 3.1$.

4 Pyramidal Relative Prefix Sum Method (PyRPS)

We are now ready to prove the next theorem which describes the main result of the proposed PyRPS method. In effect, the PyRPS method can be made as scalable as needed (assuming n large).

Theorem 1 *(Pyramidal Relative Prefix Sum) Given a data cube of size n^d, for any integer $1 \leq \beta \leq \log_2 n$ there exist a base b oblique transform $\Gamma_b \otimes \cdots \otimes \Gamma_b(D)$ which allows range sum queries in time $O(1)$ and can be updated in time $O(n^{d/\beta})$ given a change in one cell.*

Proof. We can set the height of the wavelet tree to a fix positive integer $\beta = \log_b n$ and solve for b, $b = n^{1/\beta}$ so that by proposition 4, the query cost is $O(\beta^d)$ and by proposition 5, the update cost is $O(n^{d/\beta}\beta^d)$. □

One drawback to more scalable updates is that more (fixed) steps are required to compute the range sum queries. Since increasing β slows down queries but improves the update cost, one might ask what the best compromise for β could be. Geffner et al. [8] measured the *overall complexity* by multiplying the range sum query cost with the update cost. For the PyRPS method, the overall complexity is $n^{d/\beta}\beta^{2d}$ by the proof of theorem 1. However, by choosing any fixed integer $b \geq 2$ and letting $\beta = \log_b n$, we have an overall complexity of $(b-1)^d \log_b^{2d} n$ which is clearly better for n large than any fixed β since it is logarithmic whereas $n^{d/\beta}\beta^{2d}$ is polynomial in n. In short, the overall complexity is minimized when β is large and this is discussed in the next subsection.

4.1 Optimization of the polylogarithmic case: PyRPS(log)

The PyRPS method with $\beta = \log_b n$ might still be interesting even though it no longer offers $O(1)$ queries since it minimizes the overall complexity. By propositions 4 and 5, we have that the overall complexity is $(b-1)^d \log_b^{2d} n$ and so the best choice for b arises when $(b-1)^d / \ln^{2d} b$ is a minimum. Since b must be an integer, this minimum is reached at $b = 5$ (see Fig. 1).

Lemma 1 *In a polylogarithmic PyRPS approach, the overall computational cost defined as the product of the query cost with the update cost is minimized when the base $b = 5$.*

4.2 Variable Bases

One of the assumptions made so far was that the chosen base b was constant. Other authors have experimented with variable bases [3] thus generating a large family of algorithms with various properties. This is especially applicable in the case of an algorithm such as PyRPS where we fix the height of the tree a priori. We begin by stating the variable base equivalent to Algorithm 2.

Algorithm 3 *(In-Place Oblique Wavelet Transform with Variable Base) Suppose we are given* β

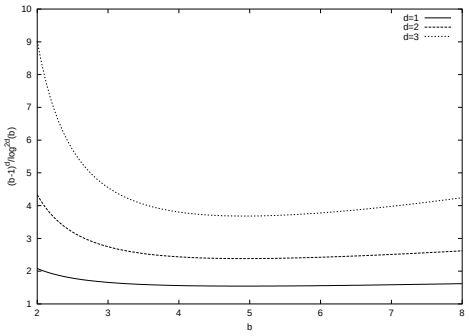

Figure 1: The estimated overall computational cost of a wavelet-based range sum system depends on the base b as $(b-1)^d/\ln^{2d} b$. For $d=1$, the minimum is reached at $b=5$.

integers $b_i > 1$ *such that* $\prod_{i=1}^{\beta} b_i = n$ *and an array of values* x_i, $i = 0,\ldots,n-1$. *Let* $B_k = \prod_{i=1}^{k} b_i$ *for* $k = 0,\ldots,\beta$ *with* $B_0 = 1$, *the transforms at depth* $j = 0,\ldots,\beta$ *are given by*

$$x_{B_j k+B_j-1} \leftarrow x_{B_j k+B_j-1},$$

$$x_{B_j k+2B_j-1} \leftarrow x_{B_j k+2B_j-1} + x_{B_j k+B_j-1},\ldots,$$

$$x_{B_{j+1} k-1} \leftarrow x_{B_{j+1} k-1} + x_{B_{j+1} k-B_j-1}$$

for $k = 0,1,\ldots,n/B_j - 1$.

Using Algorithm 3, queries still take time β^d since the depth of the tree is β. However, updates now require time $\left(\sum_{i=1}^{\beta} b_i\right)^d$ instead of $n^{d/\beta}\beta^d$ as in the proof of Theorem 1. Since the query time remains the same, the question is whether having a variable base can improve the update cost. To answer this question, we find the minimum of $\sum_{i=1}^{\beta} b_i$ over $b_1,\ldots,b_\beta$ given $\prod_{i=1}^{\beta} b_i = n$. The Lagrangian of this problem is given by $L = \sum_{i=1}^{\beta} b_i - \lambda\left(\prod_{i=1}^{\beta} b_i - n\right)$ and thus $\frac{\partial L}{\partial b_k} = 1 - \lambda\prod_{i=1,i\neq k}^{\beta} b_i$. Setting $\frac{\partial L}{\partial b_k} = 0$, we have $\lambda\prod_{i=1}^{\beta} b_i = b_k\,\forall k$. It follows that $b_1 = \ldots = b_\beta$ and thus $b_1 = \ldots = b_\beta = n^{1/\beta}$. In other words, for large n, the best choice is not to use a variable base.

5 Practical Implementation and Future Work

In implementing the PyRPS method for high performance purposes, several issues arise. For example, it might be desirable to buffer some of the topmost cells of the tree since a given cell is used much more often in queries and updates. In practice, it will often be convenient because a large fraction, $1-1/b^d$, of the wavelet coefficients are at the lowest level. It could also justify the use of a variable base approach to tailor the tree to the amount of memory available for caching the upper part of the tree. For example, if $b=5$ and $d=3$, 99% of all coefficients are at the lowest level of the tree. Second of all, the lowest level coefficients are set in blocks of size b^d-1 often updated at the same time which suggests that an efficient implementation would allow for fast "block updates". Again, the need to control the size of these blocks should be considered in the choice of the base b.

The RPS method suggests we keep the overlay in memory which implies a memory cost of $(n/b)^d$ cells where b is the size of the overlay or $n^{d/2}$ when we choose $b=\sqrt{n}$ to optimize the update complexity [13] and as pointed out by the authors, a smaller b might be chosen to optimize to overlay given the memory available. The next proposition makes precise how much of a buffer size we need to reduce queries processing by up to a fraction ξ: we make the assumption here that buffered cells can be accessed without noticeable cost when compared to permanent storage retrieval.

Proposition 6 *Whether we consider PyRPS or PyRPS(log), assuming the tree has height* β, *to buffer the first* $\xi\beta$ *levels of the tree requires storage space of* $n^{d\xi}$ *cells.*

Proof. We have $n = b^\beta$ and the first $\xi\beta$ levels of the tree have $\left(b^{\xi\beta}\right)^d = n^{d\xi}$ cells. The result is the same no matter what b is and applies to all base b trees. □

For example, if the tree has height 10 and we want to buffer 25% ($\xi = 0.25$), then we need to buffer $n^{d/4}$ cells. This tells us that to keep the memory requirements constant, we need to set $\xi = \frac{\kappa}{\ln\left(n^d\right)}$ where κ is some constant and therefore, unsurprisingly, for very large data cubes with respect to the memory available, buffering won't provide significant help.

Next, we present some early experimental results to elaborate on how a change of basis impact the range sum queries and updates performance. A data cube with $n = 256$ and $d = 3$ is generated using random data. A 32 Bytes integer value is stored in each cell so that the data cube requires 64 Megabytes of storage. Using the PyRPS method, we choose $\beta = 2$ to get $b = 16$ (shallow tree), and $\beta = 8$ to get $b = 2$ (deep tree). We also use the parameters $b = 4$ ($\beta = 4$) as a compromise. Average time for prefix sum queries (see equation 1) as well as average time for one-cell updates is given (Table 7). The one-time cost for the pyramidal transform is also given. The RPS method is not included but is computationally equivalent to the case $\beta = 2$. The benchmarking was done using Sun Java 1.4 on a Pentium 3 (1.133 GHz) laptop running Windows 2000. A flat binary file with no buffering was used for storage. These two settings provide very different performances with either faster queries ($\beta = 2$), faster updates ($\beta = 8$), or a good compromise ($\beta = 4$). Clearly, in a system where queries are much more frequent than updates, we might want to choose $b > 2$.

b	β	transform (min)	prefix sum (μs)	update (μs)
16	2	43.9	1.6	3105
4	4	44.0	3.9	274
2	8	43.9	7.9	98

Table 7: Average processing time for prefix sums and one-cell updates for different PyRPS parameters with $n = 256$ and $d = 3$ using Java on a Pentium 3 processor.

6 Conclusion

$B-$adic wavelets [11, 6] are useful in building pyramidal aggregates as the base size can be used to control the height of the tree or to optimize it overall (lemma 1). The wavelet tree itself proves to be a valuable paradigm even though the usual spectral analysis properties are of little use for simple range sum queries and thus, oblique or "stripped-down" wavelets [1] are a good choice.

The source code to reproduce the tables of this paper is freely available from the author.

About the Author

Daniel Lemire holds a Ph.D. in Engineering Mathematics from the École Polytechnique de Montréal and his B.Sc. and M.Sc. from the University of Toronto. Currently, he is a research officer at the Institute for Information Technology (NRC) in the e-Business Research Group. Previously, he has been a professor at Acadia University, an industry consultant, and a postdoctoral fellow at the Institut de génie biomédical. His research background includes wavelets, data processing, telehealth, and subdivision schemes (CAGD). He can be reached at lemire@ondelette.com.

References

[1] A. Aldroubi, Oblique Multiwavelet Bases: Examples. SPIE Conference on Mathematical Imaging: Wavelet Applications in Signal and Image Processing IV, vol. 2825, Part I, pages 54-64, Denver CO, USA, August 1996.

[2] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate Query Processing Using Wavelets. In Proc. VLDB, pages 111-122, Cairo, Egypt, September 2000.

[3] C. Y. Chan, Y. E. Ioannidis. Hierarchical Prefix Cubes for Range-Sum Queries. In Proc. VLDB, pages 675-686, Edinburgh, Scotland, UK, September 1999.

[4] I. Daubechies. Ten lectures on wavelets; CBMS-NSF Regional Conference Series in Appl. Math. 61, 1992.

[5] G. Deslauriers and S. Dubuc, Symmetric Iterative Interpolation Processes, Constructive Approximation 5, pages 49-68, 1989.

[6] G. Deslauriers, S. Dubuc, and D. Lemire, Derivatives of the Lagrange Iterative Interpolation and $b-$adic Cohen-Daubechies-Feauveau Wavelets, Technical Report EPM/RT-97/28, École Polytechnique de Montréal, Montreal, April 1997. (http://www.ondelette.com/lemire/-documents/publications/echap2.zip)

[7] D. Donoho , P. Yu. Deslauriers-Dubuc: Ten Years After; in Deslauriers G.,Dubuc S. (Eds), CRM Proceedings and Lecture Notes Vol. 18, 1999.

[8] S. Geffner, D. Agrawal, A. E. Abbadi, T. R. Smith. Relative Prefix Sums: An Efficient Approach for Querying Dynamic OLAP Data Cubes. In Proc. of ICDE, pages 328-335, Sydney, Australia, March 1999.

[9] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tabs and subtotals. In Proc. ICDE 1996, pages 131-139, 1996.

[10] A. Haar, Zur Theorie der orthogonalen Funktionen-Systeme, Math. Ann., 69, pages 331-371, 1910.

[11] P. N. Heller, Rank *M* wavelets with *N* vanishing moments, SIAM J. Math. Analysis 16, 1995.

[12] J. M. Hellerstein. Online Processing Redux, IEEE Data Engineering Bulletin, September 1997

[13] C. Ho, R.Agrawal, N. Megiddo, R. Srikant. Range Queries in OLAP Data Cubes. In Proc. ACM SIGMOD, June 1996.

[14] Y. Matias, J. S. Vitter, M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. In proc. VLDB, pages 101-110, Cairo, Egypt, September 2000.

[15] R. R. Schmidt and C. Shahabi. ProPolyne: A Fast Wavelet-based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries (extended version), VIII. Conference on Extending Database Technology, Prague, March 2002.

[16] J. S. Vitter, M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In Proc. ACM CIKM, pages 193-204, Washington D.C., November 1998.

[17] J. Vitter, M. Wang, B. R. Iyer. Data Cube Approximation and Histograms via Wavelets. In Proc. of ACM CIKM, pages 96-104, Bethesda, Maryland, USA, November 1998.

[18] Y. Wu, D. Agrawal, A. E. Abbadi. Using Wavelet Decomposition to Support Progressive and Approximate Range-Sum Queries over Data Cubes. In Proc. ACM CIKM, pages 414-421, McLean, VA, USA, November 2000.