Computing Partial Sums in Multidimensional Arrays

(Detailed Abstract)

Bernard Chazelle *Burton Rosenberg*

Department of Computer Science
Princeton University

1 Introduction

The central theme of this paper is the complexity of the *partial-sum problem:* Given a d-dimensional array A with n entries in a semigroup and a d-rectangle $q = [a_1, b_1] \times \cdots \times [a_d, b_d]$, compute the sum

$$\sigma(A, q) = \sum_{(k_1,\ldots,k_d)\in q} A[k_1, \ldots, k_d].$$

This problem comes in two distinct flavors. In *query mode*, preprocessing is allowed and q is a query to be answered on-line. In *off-line mode*, we are given the array A and a set of d-rectangles $q_1, \ldots, q_m$, and we must compute the m sums $\sigma(A, q_i)$. *Partial-sum* is a special case of the classical orthogonal range searching problem: Given n weighted points in d-space and a query d-rectangle q, compute the cumulative weight of the points in q (see e.g. [3, 4, 5, 6, 7, 9, 10, 12, 14, 17, 18, 19, 20, 21, 22]). The dynamic version of partial-sum in query-mode was studied by Fredman [10], who showed that a mixed sequence of n insertions, deletions, and queries may require $\Omega(n \log^d n)$ time, which is optimal (Willard and Lueker [20]). This result was partially extended to groups by Willard in [19]. For the case where only insertions and queries are allowed, a lower bound of $\Omega(n \log n / \log\log n)$ was proven in the one-dimensional case (Yao [22]), and later extended to $\Omega(n(\log n/\log\log n)^d)$, for any fixed dimension d (Chazelle [8]). Regarding static one-dimensional partial-sum, Yao proved that if m units of storage are used then any query can be answered in time $O(\alpha(m, n))$, which is optimal in the arithmetic model [21]. The function $\alpha(m, n)$ is the functional inverse of Ackermann's function defined by Tarjan [15]. See also Alon and Schieber [2] for related upper and lower bounds.

Our main results are a nonlinear lower bound for one-dimensional partial-sum in off-line mode and a space-time tradeoff for partial-sum in query mode in any fixed dimension. More precisely, we prove that for any n and m, there exist m partial sums whose evaluations require $\Omega(n + m\alpha(m, n))$ time. This is a rare case where the function α arises in an off-line problem. Noticeable instances are the complexity of union-find (Tarjan [15]) and the length of Davenport-Schinzel sequences (Hart and Sharir [11], Agarwal et al. [1]). Interestingly, the proof technique we use does not involve reductions from these problems. Our result implies that, given a sequence of n numbers, computing partial sums over a well-chosen set of n intervals requires a nonlinear number of additions. This might come as a surprise in light of the fact that there is a trivial linear-time algorithm as soon as we allow subtraction. The lower bound can be regarded as a generalization of a result of Tarjan [16] concerning the off-line evaluation of functions defined over the paths of a tree. As in [16] our result also leads to an improved lower bound on the minimum depth of a monotone circuit for computing conjunctions.

The other contribution of this paper is an algorithm which can answer any partial-sum query in time $O(\alpha^d(m, n))$, where m is the amount of storage available. This generalizes Yao's one-dimensional upper bound [21] to fixed arbitrary dimension d. Since our algorithm works on a *RAM*, we can use it as the inner loop of standard multidimensional searching structures. For example, consider the classical orthogonal range searching problem on n weighted points in d-space. Lueker and Willard [20] have described a data structure of size $O(n \log^{d-1} n)$ which can answer any range query in time $O(\log^d n)$ (over a semigroup). We improve the time bound to $O(\alpha(n) \log^{d-1} n)$.

The remainder of this abstract is devoted to the proofs of the lower and upper bounds. Except for a few technical lemmas whose proofs have been omitted, our exposition is complete and self-contained.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1989 ACM 0-89791-318-3/89/0006/0131 $1.50

2 A Lower Bound for Off-line Partial-Sum

This section gives a lower bound for the one-dimensional off-line partial-sum problem. Our goal is to exhibit a family of hard instances of this problem, instances for which the amortized time needed to sum m intervals over n variables grows with n while keeping the ratio m/n fixed. The existence of this family implies the impossibility of a constant amortized time algorithm for this problem. More precisely, the time grows as inverse Ackermann. Since we desire a lower bound, we can always assume the semigroup $(S, +)$ to be commutative and idempotent, that is $a + b = b + a$ and $a + a = a$ for all $a, b \in S$. With this specialization in mind we now state precisely our model of computation.

Let $X = \{ x_i \mid i \in [0, n-1] \}$ be a set of indeterminates. The power set of X is denoted $\mathcal{P}(X)$. A set of $\mathcal{P}(X)$ of the form $\{ x_k \mid k \in [i, j] \}$ is called an *interval* and is denoted $[x_i, x_j]$. Intervals which are either empty or of the form $[x_i, x_i]$ are called *trivial*. A collection of nontrivial intervals is called a *task*. A problem instance consists of a data set X and a task $T \subseteq \mathcal{P}(X)$. A solution is a *scheme* containing T, a sequence of semigroup additions taking us from the singletons $\{x_i\}$ to the sum $\sum_{i \in I} x_i$ for each $I \in T$. Using the rules of calculation, any particular sum generated by the scheme is of the form $\sum_{i \in J} x_i$ where $J \in \mathcal{P}(X)$, and the sum of two such sums is,

$$\sum_{i \in J_1} x_i + \sum_{i \in J_2} x_i = \sum_{i \in J_1 \cup J_2} x_i.$$

Thus our model of computation limits itself to considering the construction of the intervals $I \in T$ through the iterative application of union to the elements of $\mathcal{P}(X)$ at the cost of one per union. Our formal definition of a scheme is an integer r and a map $S : [-n, r] \to \mathcal{P}(X)$ such that,

$$\begin{aligned} S(-n) &= \emptyset, \\ S(-i) &= \{x_i\} \text{ for } i \in [0, n-1], \\ \forall i \in [1, r],\ \exists j, j' < i &\quad \text{s.t.} \\ S(i) &= S(j) \cup S(j'). \end{aligned}$$

A scheme S solves a task T if its range contains T, that is, $T \subseteq \{ S(i) \mid i \in [1, r] \}$. Each $S(i)$ is called a *production*. Those $S(i)$ for $i \leq 0$ are to be thought of as cost-free productions, and are also called the *axioms* of the scheme. This being so, the integer r is intuitively the time the scheme requires to sum up all intervals in the task. For this reason, we write $r = \mathit{Cost}(S)$.

To motivate the definition we have given for a scheme, we can relate it to the notion of a *faithful* semigroup as given by Yao [21]. Briefly, a faithful semigroup requires that any sum can only be realized if all the required variables are named in the sum and only those. For example, set-theoretic union forms a faithful semigroup. Also taking the maximum of integers gives a faithful semigroup. We also note that a scheme must work for all data assignments of the variables. That is, the data structure should not adapt to the details of an assignment.

The goal of this section is to exhibit a family of *hard* tasks parameterized by two integers t and k, which are called *time* and *density*, respectively. Such a task has size roughly kn where n is the number of variables over which it is defined, and the amortized time to answer a query in the task is at least t. In addition, each task in our family will have a uniformity property which will insure the success of our construction. This uniformity property will be expressed in terms of a task's *right-degree*. It is a measure of how persistently a task uses any given variable as the left endpoint of a query. In particular, we want all variables to be used fairly equally as left endpoints of queries. As more resource, space or time, is permitted, the size of the variable set over which the hard task is defined increases according to the function $R(t, k)$ defined below. Having constructed a family of hard tasks, we investigate the growth of the function $R(t, k)$. It is this function's inverse which gives the lower bound on the time needed to solve the off-line partial-sum problem as a function of the problem size.

For technical reasons, the domains of t and k are,

$$t \in \Delta = \{ i/3 \mid i \geq 3 \}, \qquad k \in \Gamma = \{0, 1, 2, \ldots\}.$$

For $(t, k) \in \Delta \times \Gamma$, define $R : \Delta \times \Gamma \to \mathbf{Z}$ by the recursion,

$$\begin{aligned} R(1, k) &= 2k, \\ R(t, 0) &= 3, \\ R(t, k) &= R(t, k-1) R\big(t - 1/3, R(t, k-1)\big), \\ & \quad \text{for } t > 1, k > 0. \end{aligned}$$

We construct a family,

$$\{ \mathcal{T}_n(t, k) \mid (t, k) \in \Delta \times \Gamma,\ n = R(t, k) \},$$

where each $\mathcal{T}_n(t, k)$ is a task over n variables. The task is *dense*, in that its size, $|\mathcal{T}_n(t, k)|$, is at least $kn/2$. The task is *hard*, in that if it is solved by a scheme S, then $\mathit{Cost}(S) \geq t\,|\mathcal{T}_n(t, k)|$. Next, we limit the size of each task by placing an additional constraint on the tasks in our family. Consider the rightwards filter of x_i in $\mathcal{P}(X)$,

$$\mathit{Fil}\,(x_i) = \{ J \in \mathcal{P}(X) \mid x_i \in J, \text{ and } (\forall j < i,\ x_j \notin J) \}.$$

The *right-degree* of task T is,

$$\mathit{Deg}\, T = \max_{0 \leq i < n} |(T \cap \mathit{Fil}\,(x_i))|.$$

We constrain $\mathcal{T}_n(t, k)$ so that $\mathit{Deg}\, \mathcal{T}_n(t, k) \leq k$.

Lemma 2.1 *For all $(t, k) \in \Delta \times \Gamma$, $n = R(t, k)$, there exists a task $\mathcal{T}_n(t, k)$ on n variables such that:*

1. $|\mathcal{T}_n(t, k)| \geq kn/2$.
2. $k \geq \mathit{Deg}\, \mathcal{T}_n(t, k)$.
3. *If scheme S solves $\mathcal{T}_n(t, k)$, and $k > 0$, then $\mathit{Cost}(S) \geq t\,|\mathcal{T}_n(t, k)|$.*

We devote the next few pages to the proof of this result. For $k = 0$ we set $\mathcal{T}_3(t, 0) = \emptyset$, for this choice trivially satisfies all the conditions. For $t = 1$ and $k > 0$ we put $n = R(1, k) = 2k$, $X = \{x_0, \ldots, x_{n-1}\}$, and,

$$\mathcal{T}_n(1, k) = \bigcup_{i=0}^{n-k-1} \bigcup_{j=1}^{k} [x_i, x_{i+j}].$$

This is a collection of nontrivial intervals defined on X, hence a task. The right-degree is bounded by k and the size is $|\mathcal{T}_n(1, k)| = k(n - k) = k^2 = kn/2$. If a scheme S solves $\mathcal{T}_n(1, k)$ its range necessarily contains it and therefore $\mathit{Cost}(S) \geq |\mathcal{T}_n(1, k)|$.

We now assume that $k > 0$ and $t > 1$. By induction hypothesis, for $a = R(t, k-1)$, $b = R(t - 1/3, a)$, we have exhibited tasks $A = \mathcal{T}_a(t, k-1)$ and $B = \mathcal{T}_b(t - 1/3, a)$. Name the variables in A by $Y = \{y_0, \ldots, y_{a-1}\}$, and those in B by $Z = \{z_0, \ldots, z_{b-1}\}$. Put $n = R(t,k) = ab$ and $X = \{x_0, \ldots, x_{n-1}\}$. We construct task $Q \subseteq \mathcal{P}(X)$ from A and B. Taking b copies of A, we place them side by side in X. A copy of B is stretched over X and fringes are added to increase the complexity of the resulting intervals. Consider X divided into b blocks each containing a consecutive variables. Let the leftmost variable in each block be marked, that is x_{ia} for $i = 0, 1, \ldots, b-1$. Alter the marking by removing the mark on x_0 and placing it on x_{n-1}. Task B is stretched by considering these b marked variables as being the b variables over which B is defined, the correspondence being the obvious order-preserving association. Each interval $[z_i, z_j]$ in B lifts to the unique smallest interval $[x_{i'}, x_{j'}]$ containing all the marked variables associated with $[z_i, z_j]$. Since B has density a, at most a intervals in B have their leftmost ends on a given marked x_i. Further stretch each interval leftward, by a different amount, so that they end over x_{i-1}, x_{i-2}, etc. The point of this transformation is twofold. First, we are correcting the right-degree of the resulting task. Second, we are adding extra difficulty to the solution of the resulting task. We shall now formalize the preceding discussion and show that the resulting task Q is of the correct form, density and right-degree. Finally, we will show why any scheme solving Q is necessarily expensive.

Throughout this section we shall make use of mappings $f : \mathcal{P}(X) \to \mathcal{P}(Y)$ for which $f(A \cup B) = f(A) \cup f(B)$. Such maps are completely defined by specifying the values of f on the singleton sets $\{x\}$, for all $x \in X$. We also apply such maps to collections of elements in $\mathcal{P}(X)$. If $A \subseteq \mathcal{P}(X)$ then,

$$f(A) = \{\, f(a) \mid a \in A \,\}.$$

For $j \in [0, b-1]$ define maps,

$$\begin{array}{rccl} \varphi_j : & \mathcal{P}(Y) & \to & \mathcal{P}(X) \\ & \{y_i\} & \mapsto & \{x_{ja+i}\}, \quad i \in [0, a-1], \end{array}$$

and apply these to A, forming collections of intervals,

$$Q_j = \{\varphi_j(x) \mid x \in A\} \quad j = 0, \ldots, b-1.$$

Partition B into a subsets $B_0, \ldots, B_{a-1}$ so that the partition obeys the following restrictions:

1. $Deg\ B_i \leq 1$ for all i.
2. $[z_{b-2}, z_{b-1}] \notin B_0$.

As mentioned above, intervals in B are stretched to form "large" intervals in Q, large meaning that they span over blocks. Also, these stretched intervals are further lengthened some additional variables leftward, in fact, B_i is lengthened i additional variables leftward. The purpose of the first restriction is to control the right-degree of the resulting task Q. The purpose of the second restriction is to insure that all intervals coming from B in Q span over more than one block. The interval $[z_{b-2}, z_{b-1}]$, if present in B and allowed to fall in B_0, would be sent into Q as the interval $[x_{a(b-1)}, x_{ab-1}]$, which would lie fully inside the leftmost block. Since $Deg\ B \leq a$ and since there is only one nontrivial interval ending over z_{b-2}, it is possible to construct this partition.

For $i \in [0, a-1]$ define the map,

$$\begin{array}{rccl} \psi_i : & \mathcal{P}(Z) & \to & \mathcal{P}(X) \\ & \{z_j\} & \mapsto & \begin{cases} \{x_{(j+1)a-i}, x_{(j+1)a}\} & \text{for } j \in [0, b-2] \\ \{x_{ab-1}\} & \text{if } j = b-1. \end{cases} \end{array}$$

Note that this map does not yield intervals. Let $\chi(x)$ be the smallest interval containing the set x,

$$\chi(x) = \bigcap \{\, [y,z] \in \mathcal{P}(X) \mid x \subseteq [y,z] \,\}.$$

An image of each B_i is placed in X by,

$$Q'_i = \{\chi\psi_i(z) \mid z \in B_i\} \quad i = 0, \ldots, a-1.$$

The task Q is defined as,

$$Q = \left(\bigcup_{j=0}^{b-1} Q_j \right) \bigcup \left(\bigcup_{i=0}^{a-1} Q'_i \right).$$

We investigate the properties of this task.

By construction, Q is a collection of nontrivial intervals in X. In fact, each map φ_i and $\chi\psi_i$ is one-to-one. The distinct character of each of these maps assures that the Q_j and Q'_i are pairwise disjoint. So, we easily check that

$$\begin{aligned} |Q| &= \sum_{j \in [0,b-1]} |Q_j| + \sum_{i \in [0,a-1]} |Q'_i| \\ &= b\,|A| + |B| \geq b(k-1)a/2 + ab/2 = kab/2, \end{aligned}$$

and,

$$\begin{aligned} Deg\ Q &= \max_{i \in [0,ab-1]} |Q \cap Fil\ (x_i)| \\ &= \max_{i \in [0,ab-1]} \left| \left(Q_{\lfloor i/a \rfloor} \cup Q'_{-i \bmod a}\right) \cap Fil\ (x_i) \right| \\ &\leq (k-1) + 1 = k. \end{aligned}$$

Therefore Q is a task of the correct density and right-degree for a task in our family of hard tasks.

We will now derive a lower bound on the cost of any scheme S which solves Q. Recall that a scheme S is a map from $[-n, r]$ to $\mathcal{P}(X)$. We try to partition $[-n, r]$ according to where $S(i)$ falls, for $j \in [0, b-1]$,

$$\begin{aligned} \Lambda_j &= \{\, i \in [-n, r] \mid S(i) \subseteq [x_{ja}, x_{(j+1)a-1}] \,\}, \\ \Lambda_b &= [-n, r] \setminus \bigcup_{0 \leq j < b} \Lambda_j. \end{aligned}$$

This fails to be a partition since all Λ_j for $j \neq b$ share $-n$ as an element (the empty set is a subset of any block). The set $[-n+1, r]$ is well partitioned, however. A relabeling $L_i : [-a, r_i] \to \Lambda_i$ for $i \in [0, b-1]$ gives b distinct subsequences $S_i(j) = S(L_i(j))$ of $S(j)$. We will show that up to isomorphism each of these subsequences is a scheme solving A. Relabeling Λ_b by $L_b : [2, r_b + 1] \to \Lambda_b$, we have a sequence which is essentially isomorphic to a scheme solving B. It will be shown that, in fact, it is essentially isomorphic to a very inefficient scheme solving B. This inefficiency is the direct consequence of the fringing step in the construction, and is the key element in making a task "harder than the sum of its parts". Because we have partitioned $[1, r]$, we have the important fact,

$$\sum_{j=0}^{b} r_j = r.$$

The next two lemmas explicate the stated isomorphisms. The proofs are simple but long and tedious.

Lemma 2.2 *For $i = 0, \ldots, b-1$, the sequences $S_i : [-a, r_i] \to \mathcal{P}(X)$ are isomorphic to schemes solving A.*

Proof: Define a map, the inverse of φ_i for all i,

$$\begin{array}{rccc} \varphi^* : & \mathcal{P}(X) & \to & \mathcal{P}(Y) \\ & \{x_j\} & \mapsto & \{y_{j \bmod a}\}. \end{array}$$

We claim that for all i in $[0, b-1]$, the sequence given by the composite map $\varphi^* S_i : [-a, r_i] \to \mathcal{P}(Y)$ is a scheme in Y solving task A. Because φ^* takes the empty set to the empty set $\varphi^* S_i(-a) = \emptyset$. For $j \in [0, a-1]$ we calculate $\varphi^* S_i(-j) = \varphi^* S(-j - ai) = \varphi^*(x_{j+ai}) = y_j$. The set Λ_i includes $-a+1$ negative integers, those j for which $S(j) = \emptyset$ or $S(j) = \{x_l\}$ for $l \in [ia, (i+1)a - 1]$. Therefore, $j > 0$ implies $L_i(j) > 0$. Put $j_o = L_i(j)$. Since S is a scheme there exist integers j_o' and j_o'' strictly less than j_o and $S_i(j) = S(j_o) = S(j_o') \cup S(j_o'')$. But this gives $S(j_o') \subseteq x$ and hence $j_o' \in \Lambda_i$. Likewise for j_o''. So there are integers j' and j'' satisfying $j_o' = L_i(j')$ and $j_o'' = L_i(j'')$. The nature of the map L_i is such that $j_o', j_o'' < j_o$ implies $j', j'' < j$. But,

$$\begin{aligned} \varphi^* S_i(j) &= \varphi^*\big(S(L_i(j')) \cup S(L_i(j''))\big) \\ &= \varphi^* S_i(j') \cup \varphi^* S_i(j''), \end{aligned}$$

therefore $\varphi^* S_i$ is a scheme. Since S solves Q it also solves Q_i, which means that there exists j such that $S(j) = q$ for any $q \in Q_i$. But $Q_i = \varphi_i(A)$, hence $q = \varphi_i(a)$ for some $a \in A$. Furthermore, $q \subseteq [x_{ai}, x_{(i+1)a-1}]$ implies that $j \in \Lambda_i$. So there exists j' for which $S_i(j') = \varphi_i(a)$, and $\varphi^* S_i(j') = \varphi^* \varphi_i(a) = a$. Any $a \in A$ gives rise to a $q = \varphi_i(a)$ in Q_i for which $\varphi^*(q) = a$. Therefore $\varphi^* S_i$ solves A. □

Lemma 2.3 *There exists a map ψ^* such that $S_b : [-b, r_b + 1] \to \mathcal{P}(Z)$ is a scheme solving B and for $i \in [2, r_b + 1]$ we have $S_b(i) = \psi^* S(L_b(i))$.*

Proof: Define the map,

$$\begin{array}{rccl} \psi^* : & \mathcal{P}(X) & \to & \mathcal{P}(Z) \\ & \{x_i\} & \mapsto & \begin{cases} \{z_j\} & \text{if } i = a(j+1),\ j \in [0, b-2], \\ \{z_{b-1}\} & \text{if } i = ab - 1, \\ \emptyset & \text{otherwise.} \end{cases} \end{array}$$

This map inverts the action of $\chi\psi_i$. The effect of ψ^* is the same as setting all but the marked variables in X to zero. We show that this results in a scheme for B. We define the sequence $S_b : [-b, r_b + 1] \to \mathcal{P}(Z)$ by,

$$\begin{aligned} S_b(-b) &= \emptyset, \\ S_b(-i) &= z_i, \quad i \in [0, b-1], \\ S_b(1) &= [z_{b-2}, z_{b-1}], \\ S_b(i) &= \psi^*\big(S(L_b(i))\big), \quad i \in [2, r_b + 1]. \end{aligned}$$

We check that for all $i > 0$ there exist $j, j' < i$ such that $S_b(i) = S_b(j) \cup S_b(j')$, and $Im(S_b) \supseteq B$.

By definition $S_b(1)$ is a valid production, so let $i_o = L_b(i)$ for $i > 1$. Since S is a scheme, we have $S(i_o) = S(j_o) \cup S(j_o')$ for integers $j_o, j_o' < i_o$. If j_o is in Λ_b let j satisfy $j_o = L_b(j)$. From the nature of L_b we know that $j_o < i_o$ implies $j < i$. Otherwise, let l be such that $j_o \in \Lambda_l$. We consider two cases. If $l \neq b - 1$ then $S(j_o)$ contains at most one marked variable, and $\psi^* S(j_o) = S_b(j)$, for some $j \leq 0$. If $l = b-1$, it is possible that $S(j_o)$ contains both marked variables x_{ab-1} and $x_{a(b-1)}$, and therefore $\psi^* S(j_o) = S_b(j)$ for some $j \leq 1$. In either case $j < i$. Repeating the argument for $S(j_o')$ we have $\psi^* S(j_o') = S_b(j')$, for some $j' \leq i$. Therefore, as required, we have,

$$\begin{aligned} S_b(i) &= \psi^*\big(S(i_o)\big) = \psi^*\big(S(j_o) \cup S(j_o')\big) \\ &= \psi^* S(j_o) \cup \psi^* S(j_o') = S_b(j) \cup S_b(j'). \end{aligned}$$

Let $q \in B$ be an interval. Since q is in the partition B_i for some i, we have $\chi\psi_i(q) \in Q_i'$. Because S solves Q, for some j, we also have $S(j) = \chi\psi_i(q)$. It is clear that $j \in \Lambda_b$, so $\psi^* \chi\psi_i(q) \in Im(S_b)$. Write $q = [z_{j'}, z_{j''}]$. If $j'' \neq b - 1$ then,

$$\psi^* \chi\psi_i([z_{j'}, z_{j''}]) = \psi^*\left[x_{(j'+1)a-i}, x_{(j''+1)a}\right] = [z_{j'}, z_{j''}].$$

If $j'' = b - 1$, a similar calculation is performed. In either case, $q \in Im(S_b)$, therefore S_b solves B. □

We will show that the scheme S_b is not of minimum cost. Call a production $S_b(i)$ *redundant* if there exists $j < i$ for which $S_b(j) = S_b(i)$. Clearly, omitting a redundant production from a scheme does not invalidate it.

Lemma 2.4 *There are at least $|B| - |B_0|$ redundant productions among the productions $S_b(i)$, $2 \leq i \leq r_b + 1$.*

Proof: To find candidates for redundant productions, we identify the first time in the scheme S that a particular variable is the leftmost in a production which spans across blocks. The index for this production is in Λ_b. If the variable under scrutiny is not marked then this production, as seen from S_b, looks like the addition of 0 to a precomputed sum. Consider the witness function,

$$W(i) = \begin{cases} \min\big\{\, j \in \Lambda_b \mid \big(\min i' \text{ s.t. } x_{i'} \in S(j)\big) = i \\ \qquad \text{and } \big(\exists i',\ i'a > i, \text{ s.t. } x_{i'a} \in S(j)\big) \,\big\}, \\ \infty \quad \text{if this set is empty.} \end{cases}$$

We claim that if i is not divisible by a and $W(i) < \infty$, then $S_b(i_o)$ is redundant, where i_o is the unique integer such that $W(i) = L_b(i_o)$.

Let $j = W(i)$. Since j is finite, $j \in \Lambda_b$, and so $j > 0$. Put $S(j) = S(j') \cup S(j'')$ with $j', j'' < j$. It is immediate that for all $i' < i$, $x_{i'} \notin S(j')$ and $x_{i'} \notin S(j'')$. Without loss of generality, suppose that $S(j')$ contains x_i. By virtue of that fact $j' < j$ it follows $S(j')$ does not contain $x_{i'}$, for any i' divisible by a and larger than i. Furthermore, i is not divisible by a and $i < a(b-1)$. Therefore $S(j')$ contains no marked variables, that is,

$$\begin{aligned} \psi^* S(j) &= \psi^*\big(S(j') \cup S(j'')\big) \\ &= \psi^* S(j') \cup \psi^* S(j'') = \psi^* S(j''). \end{aligned}$$

If $j'' \in \Lambda_b$ then by $j'' < j$ we find an $i_o'' < i_o$ for which $j'' = L_b(i_o'')$ and $S_b(i_o'') = S_b(i_o)$. We conclude $S_b(i_o)$ is redundant. If $j'' \notin \Lambda_b$ then $\psi^* S(j'') = S_b(i_o'')$ for $i_o'' \leq 1 < i_o$. We have $S_b(i_o'') = S_b(i_o)$ and therefore $S_b(i_o)$ is redundant.

The map $P(q) = \min\{\, i \mid x_i \in q \,\}$ projects elements in $\cup_{i=1}^{a-1} Q_i'$ to integers which are not multiples of a (note the omission of $i = 0$ in the union). In addition, for any q in this union, since S solves Q, there exists i such that $S(i) = q$. It follows that $W(P(q))$ is always finite. Thereby, with every such q we have $i = L_b(j)$ for some j and $S_b(j)$ is redundant. We derive the composite map,

$$\bigcup_{i=1}^{a-1} Q'_i \xrightarrow{P} Im(P) \xrightarrow{W} \Lambda_b \xrightarrow{L_b^{-1}} [2, r_b + 1],$$

where any element in the rightmost target coming from the leftmost source is the index of a redundant production. The lower bound on the number of redundant productions follows from the injection of the map. Indeed, the map P fails to inject if and only if two intervals in $\cup Q'_i$ have the same left endpoint. By construction of Q'_i this is impossible. If $W(i) = W(i')$ then $x_i = x_{i'}$, or $i = i'$, and therefore W is injective. Finally, L_b was constructed to be one-to-one, hence its inverse is also injective. □

It is now a simple matter to derive a lower bound on $Cost(S)$. For each $i \in [0, b-1]$ the scheme S_i is isomorphic to one solving A, by Lemma 2.2. Therefore the cost r_i of scheme S_i is at least $t\,|A|$. Stripping S_b of its $|B| - |B_0|$ redundant productions, we obtain a scheme of size $r_b + 1 - |B| + |B_0|$, which is rich enough to solve B, by Lemmas 2.3 and 2.4. Therefore $r_b + 1 - |B| + |B_0| \geq (t - 1/3)\,|B|$. Summing up, we derive,

$$r = \sum_{i=0}^{b} r_i \geq bt\,|A| + (t - 1/3)\,|B| + |B| - |B_0| - 1.$$

Recall that each interval in B_0 ends over one of $z_0, \ldots, z_{b-3}$, and conversely each $z_0, \ldots, z_{b-3}$ has at most one interval in B_0 ending over it. Hence $|B_0| \leq b-2$. Since $|Q| = b\,|A| + |B|$, we have,

$$\begin{aligned} r &\geq bt\,|A| + (t + 2/3)\,|B| - (b-2) - 1 \\ &= t\,|Q| + (2/3)\,|B| - b + 1. \end{aligned}$$

Because $B = \mathcal{T}_b(t - 1/3, a)$ we have $|B| \geq ab/2$. Since $a \geq 3$,

$$r \geq t\,|Q| + ab/3 - b + 1 = t\,|Q| + 1.$$

We conclude that $Cost(S) = r > t\,|Q|$. Setting $Q = \mathcal{T}_n(t, k)$ the proof of Lemma 2.1 is now complete.

Using this result we can derive a lower bound on $T(m, n)$, the number of operations needed to complete a task of m intervals over n variables. Define,

$$\beta(m, n) = \min\{\, t \mid R(t, \lfloor m/n \rfloor) \geq n \,\}.$$

Lemma 2.5 *$T(m,n) = \Omega(m\beta(m,n))$ for $m \geq n$, $T(m,n) = \Omega(n + m\,\beta(m,m))$ for $m < n$.*

Proof: Given m and n, let $k = \lfloor m/n \rfloor$ and $\beta = \beta(m,n)$. Assume that $m \geq n$. There are no more than $\binom{n}{2}$ intervals in a task, therefore,

$$R(1, \lfloor m/n \rfloor) = 2\lfloor m/n \rfloor \leq 2\lfloor n(n-1)/(2n) \rfloor \leq n - 1,$$

from which it follows that $\beta > 1$, and hence $\beta \geq 4/3$. Let $n_o = R(\beta - 1/3, k)$, note $n_o < n$ by definition of β. By Lemma 2.1 there exist a task $\mathcal{T}_{n_o}(\beta - 1/3, k)$ of size between $kn_o/2$ and kn_o, whose minimum derivation length is bounded below by $(\beta - 1/3)\,|\mathcal{T}_{n_o}(\beta - 1/3, k)|$. Placing $\lfloor n/n_o \rfloor$ copies of $\mathcal{T}_{n_o}(\beta - 1/3, k)$ side by side and adding enough new variables and intervals to adjust the number of variables to n and the size of the task to m, we derive,

$$\begin{aligned} T(m,n) &\geq \lfloor n/n_o \rfloor (\beta - 1/3)\,|\mathcal{T}_{n_o}(\beta - 1/3, k)| \\ &\geq \lfloor n/n_o \rfloor (\beta - 1/3) k n_o/2 \\ &\geq \lfloor n/n_o \rfloor \lfloor m/n \rfloor (\beta - 1/3) n_o/2 \\ &\geq m(\beta - 1/3)/8 = \Omega(m\beta). \end{aligned}$$

Suppose that $m < n$. Temporarily ignoring n, proceed as above to create task Q over m variables of size m whose solution time is $\Omega(m\,\beta(m,m))$. We can assume that all x_i in the set $\{x_0, \ldots, x_{m-1}\}$ appear in the resulting task. Else, we discard the unused variables, renumbering the others from 0 to $m' - 1$. Next we add variables $\{x_{m'}, \ldots, x_{n-1}\}$ and find an interval $[i, m'-1]$ in Q which we replace by $[i, n-1]$. Now Q involves all variables, so its cost is $\Omega(n)$. Hence $T(m,n) \geq \Omega(n + m\,\beta(m,m))$. □

The functions R and β are akin to Ackermann's function and its functional inverse as defined in (Tarjan [16]),

$$\begin{aligned} A(1,j) &= 2^j, \quad j \geq 1; \\ A(i+1,1) &= A(i,2), \quad i \geq 1; \\ A(i,j) &= A(i-1, A(i, j-1)), \quad i,j \geq 2, \end{aligned}$$

and,

$$\alpha(m,n) = \min\{\, i \mid A(i, \lfloor m/n \rfloor) > \log n \,\}.$$

The remainder of this section proves that Lemma 2.5 is still correct if we replace $\beta(m,n)$ by $\alpha(m,n)$. We need some technical facts about $A(i,j)$, whose proofs are omitted.

Lemma 2.6 *The function A satisfies $A(i,j) > j$ for all i, j.*

Lemma 2.7 *The function $A(i,j)$ is increasing in j.*

Lemma 2.8 *For $i > 1$, $A(i, j+1) > A(i,j)^2$.*

Lemma 2.9 *For all i and j we have $2^{A(i,j)} \leq A(i+1, j)$.*

We introduce an intermediate function to help establish the relationship between $A(i,j)$ and $R(t,k)$. Define,

$$\begin{aligned} B(1,j) &= 2^j, \quad j \geq 1; \\ B(i+1,1) &= B(i,2), \quad i \geq 1; \\ B(i,j) &= B(i, j-1) B(i-1, B(i, j-1)), \quad i,j \geq 2. \end{aligned}$$

Lemma 2.10 *For all i and j positive integers, $A(i+1, j) > B(i,j) \geq A(i,j)$. Furthermore, $B(i,j)$ is increasing in j.*

For notational convenience, we change the indices of $R(t,k)$ by writing $R'(t,k) = R((t+2)/3, k)$, and neglect the column $k = 0$ where explosive growth does not occur. That is, we define,

$$\begin{aligned} R'(1,k) &= 2k, \quad k \geq 1; \\ R'(t+1,1) &= 3R'(t,3), \quad t \geq 1; \\ R'(t,k) &= R'(t, k-1) R'(t-1, R'(t, k-1)), \\ &\qquad t, k \geq 2. \end{aligned}$$

Lemma 2.11 *For all i and j, $A(i+2, j) > R'(i,j)$ and $R'(i+1, j) > A(i,j)$.*

Lemma 2.12 *For all m and n, with $m \geq n$, $\beta(m,n) = \Theta\big(\alpha(m,n)\big)$.*

We therefore have the following lower bound on the partial-sum problem:

Theorem 2.1 *If $m \geq n$ then $T(m,n) = \Omega\big(m\,\alpha(m,n)\big)$. If $m < n$ then $T(m,n) = \Omega\big(n + m\,\alpha(m,m)\big)$.*

Proof: Combine Lemma 2.5 and Lemma 2.12. □

3 An Upper Bound for Multidimensional Partial Sums

A *scheme* is a method of precomputing sums from a given set of input variables so that any interval of variables can be summed up by adding together a small number of these precomputed sums. Informally, it consists of an algorithm which first of all specifies how to fill in a memory array with precomputed sums, and then responds to any query interval by retrieving a small number of precomputed sums, presenting their grand total as the weight of the interval. We refer to the size of the array of precomputed sums as the *space* used by the scheme, and the number of partial sums needed to express any interval sum as the *time* needed by the scheme. As it turns out, in a RAM implementation the overhead of finding the needed partial sums is negligible.

The method of filling the memory array with precomputed sums is completely general in that it assumes nothing about the addition operation or about the particular assignment of values to the input variables. It is in effect a fixed map $\mathcal{S} : X \rightarrow \mathcal{S}(X)$ taking X, an assignment of values to the input variables, to $\mathcal{S}(X)$, the sequence of values in the semigroup. Our method will have an additional favorable property: this map will be calculated at the rate of one semigroup addition per value of the sequence. That is, all intermediate results occurring in the setting up of the scheme are themselves part of the scheme.

The space used by a scheme is the length of the sequence $\mathcal{S}(X)$. Among the subsets of the input variables are the queries, interpreted as rectangles by arranging the variables on points of a multi-dimensional integer lattice. For instance, in d dimensions consider indexing the input variables by d-tuples $i = (i_1, \ldots, i_d)$. In this situation a query, specified by a pair of vectors $j = (j_1, \ldots, j_d)$ and $k = (k_1, \ldots, k_d)$, is the set of all input variables with indices i such that $j \leq i \leq k$. The partial order $\leq$ refers to coordinate domination. The scheme provides a map T from queries to subsets of the sequence $\mathcal{S}(X)$ such that for any query q,

$$\sum_{v \in T(q)} \mathcal{S}(X)(v) = \sum_{x \in q} x,$$

this equality being true for any assignment X. The *time* used by a scheme is the maximum size of $T(q)$ over all permissible queries q. Given the input variable set of size n, a scheme using space kn and time t is denoted $\mathcal{S}_n(t,k)$. By abuse of terminology, the list $\mathcal{S}_n(t,k)(X)$ will also be denoted by $\mathcal{S}_n(t,k)$. The suggestion is that a scheme should always be thought of in its evaluated form — since cells can only store elements from the semigroup, it is only in its evaluated form that the scheme can be represented in memory. However, one must consider the assignment X as completely undetermined and general so that the universal nature of the scheme is not undermined.

Schemes are constructed recursively. A query rectangle is decomposed into a small number of slices, some fat and some thin, by considering the problem as a one-dimensional array of $(d-1)$-dimensional arrays. The set of $(d-1)$-dimensional arrays over a semigroup, with addition defined componentwise, is itself a semigroup. The one-dimensional scheme gives a list of $(d-1)$-dimensional subproblems, each of which is solved recursively. The construction for the one-dimensional case is given by a two path recursion — one path solving "small" queries which fit within carefully calculated blocks, the other path solving "large" queries which span over our defined blocks, but which therefore allows us to treat the entire block as a single value.

Given parameters t and k, there is a maximum number of input variables within which a scheme with these parameters can be constructed. We give growth functions, $R_d(t,k)$, indexed by dimension, which both guide the choice of block sizes and guarantee the construction of schemes. If

$$\max\{\, n_i \mid i = 1, \ldots, d \,\} \leq R_d(t,k)$$

then we can construct a scheme of size $kn_1 \times \cdots \times n_d$ which solves arbitrary queries in time at most t.

The following function is the model for all the R_d's. By appropriate change of variables, each function R_d can be expressed in this form. For $t \in \{1,3,5,\ldots\}$ and $k \in \{1,4,7,\ldots\}$,

$$\begin{aligned} R(1,k) &= 1, \quad k \geq 1; \\ R(t,1) &= 4, \quad t > 1; \\ R(t,k) &= R(t,k-3)R\big(t-2, R(t,k-3)\big), \quad t,k > 1. \end{aligned}$$

Remark that all images of R are integers congruent to 1 mod 3.

Lemma 3.1 *For any (t,k) in the domain of R and any $n \leq R(t,k)$, there exists a one-dimensional scheme $\mathcal{S}_n(t,2k)$ solving any query in time t using $2kn$ cells. The scheme can be constructed in $2kn$ semigroup additions.*

Proof: The construction proceeds by a double induction and is similar to (Yao [21]). Let the set of n input variables be denoted by $\{\, x_i \mid i = 0, \ldots, n-1 \,\}$. The cases $t \leq 3$ and $k \leq 1$ are trivial. We therefore assume that $t > 3$ and $k > 1$. Let $a = R(t,k-3)$ and $b = R(t-2,a)$. By induction, schemes $\mathcal{S}_{a'}(t, 2(k-3))$ and $\mathcal{S}_{b'}(t-2, 2a)$ can be constructed for any $a' \leq a$ and $b' \leq b$. We show the construction of $\mathcal{S}_n(t,2k)$ for any $n \leq ab$. Without loss of generality we assume that $a < n$. Let $\hat{b} = \lceil n/a \rceil$. We have the inequalities,

$$a \leq a(\hat{b}-1) < n \leq a\hat{b} \leq ab.$$

The $\hat{b}$ intervals $[0,a-1], [a,2a-1], \ldots, \big[(\hat{b}-1)a, n\big]$ partition the variable set $\{x_i\}$ into blocks of at most a variables each. The first $\hat{b}-1$ blocks contain exactly a variables and the last block contains $a' \leq a$ variables. Schemes $\mathcal{S}_a(t, 2(k-3))$ and $\mathcal{S}_{a'}(t, 2(k-3))$ are constructed inductively. These will solve any query falling fully inside one of the blocks. For use by these schemes we reserve memory cells $B_{i,j}$ with $i \in [0, n-1]$ and $j \in \left[0, 2(k-3)\right]$.

Queries that span across blocks are handled in two steps. First the query is shortened so that its ends coincide with block boundaries. We precalculate and store in cells C_i^1 and C_i^2 what it is possible to remove in this manner. That is, for $i \in [0, n-1]$:

$$C_i^1 = \begin{cases} x_i & \text{for } i \equiv -1 \bmod a, \text{ or if } i = n-1; \\ x_i + C_{i+1}^1 & \text{else.} \end{cases}$$

$$C_i^2 = \begin{cases} x_i & \text{for } i \equiv 0 \bmod a; \\ x_i + C_{i-1}^2 & \text{else.} \end{cases}$$

The shortened query can now be considered as a query over variables $C_{ai}^1 = x_{ai} + x_{ai+1} + \cdots + x_{a(i+1)-1}$. We seek to answer this query in time $t-2$. With the two remaining time units we add the appropriate C_i^l to recover the original query. We reserve cells D for a scheme to answer this query over reduced variables C_i^1. Since $b = R(t-2, a)$ and $\hat{b} \leq b$, there exists a scheme $\mathcal{S}_{\hat{b}}(t-2, 2a)$ which can solve the shortened query. Since we have $2a\hat{b}$ cells available for D, the induction is complete. Indeed,

$$\begin{aligned} 2kn - |B| - |C^1| - |C^2| - |D| &= \\ 2kn - 2(k-3)n - n - n - 2a\hat{b} & \\ = \quad 4n - 2a(\hat{b}-1) - 2a > 2(n-a) > 0. & \end{aligned}$$

Discussion now turns to the linear-time constructibility of these schemes. We put the additions used to construct the scheme in one-to-one correspondence with memory cells. We also need to consider how values for a and $\hat{b}$ are selected at each level of the recursion. For this selection we refer to the table R, and therefore must build that section of R which contains values less than n. We shall show that $R(t,k)$ is increasing in both t and k, so that the required section can be constructed by the program,

```
$i := 5$ ;
while $(R(i,4) < n)$ begin
  $j := 7$ ;
  while $(R(i,j) < n)$ $j := j + 3$ ;
  $i := i + 2$ ;
end ;
```

Note that we have omitted the calculation of $R(i,j)$ as well as the calculation of $R(i,j)$ in its constant rows and column.

Clearly, the nonconstant rows of $R(t,k)$ are increasing. Furthermore we have $R(2i+1, 4) = 4^i$ and hence $R(2i+1, k) \geq 4^i$, for $k \geq 4$. Note that, for $i > 0$,

$$\begin{aligned} R(2i+1, k+3) &= R(2i+1,k)R(2i-1,k') \\ &\geq R(2i+1,k)4^{i-1}, \end{aligned}$$

since $k' \geq 4$. We derive the lower bound,

$$R(2i+1, 3j+1) \geq 4^{(i-1)j},$$

for all $i, j \geq 0$. So, for $i, j > 1$,

$$\begin{aligned} & R(2(i+1)+1, 3j+1) > \\ & \qquad R\big(2i+1, R(2(i+1)+1, 3(j-1)+1)\big) \\ & \geq \quad R\left(2i+1, 4^{i(j-1)}\right) \\ & \geq \quad R(2i+1, 6j+4), \end{aligned}$$

from which it immediately follows that the number of values less than n in the nonconstant rows is at most proportional to,

$$|\{\, k \mid R(5,k) \leq n \,\}| = O(\log n).$$

Given $2k$ storage cells and n variables, where k is congruent to 1 mod 3 and $k \geq 4$, find the smallest t such that $R(t,k) \geq n$. If the inequality is strict, we have left the constructed segment of the table, and should reassign k to be the smallest appropriate integer such that $R(t,k) \geq n$ for this t. Now we are assured that $R(t, k-3) < n$ and hence lies in the constructed segment of the table. As a result, the values of a, a' and $\hat{b}$ are obtainable from the above construction. The remainder of the construction uses up one cell of memory for every semigroup addition performed, hence the number of additions is the number of cells. □

Schemes for multidimensional variable sets are now considered. In d dimensions, the variables are indexed by vectors of integers. Queries are d-rectangles with integer coordinates. For each dimension, a function $R_d(t,k)$ is defined for all positive integers t and k. We have,

$$R_1(t,k) = \begin{cases} R\big(\lfloor t \rfloor_{1(2)}, \lfloor k/2 \rfloor_{1(3)}\big) & \text{if } t > 0, k > 1, \\ 0 & \text{else,} \end{cases}$$

where $\lfloor s \rfloor_{i(j)}$ denotes the greatest integer $\leq s$ that is congruent to i mod j. In the d-dimensional case, $d > 1$, let

$$\{\, x_{(i_1,\ldots,i_d)} \mid i_j \in [0, n_j - 1] \,\}$$

be the variable set, that is, a d-dimensional array. If $n_j \leq R_d(t,k)$ for $j = 1, \ldots, d$, then there exists a scheme using an k cells per variable, amortized, and which answers any query in time t. We construct this scheme, having defined,

$$R_d(t,k) = R_1\left(\lfloor \sqrt[d]{t} \rfloor, \lfloor \sqrt[d]{k} \rfloor\right).$$

Lemma 3.2 *Let $n = (n_1, \ldots, n_d)$ where $n_j < R_d(t,k)$ for all j. There exists a d-dimensional scheme $\mathcal{S}_n(t,k)$ for the set of input variables x_i for $i = (i_1, \ldots, i_d)$ in the range $0 \leq i < n$. The scheme can be constructed by using $kn_1 \times \cdots \times n_d$ semigroup additions.*

Proof: We assume $t = \tau^d$ and $k = \kappa^d$ for integers τ and κ. If this is not true, replace t or k by a smaller integer which is a perfect d-th power. Lemma 3.1 deals with the case $d = 1$. For $d > 1$, we reduce the dimension by one, and by a recursive construction the result follows. That is, we show how a d-dimensional problem is essentially the product of two problems of dimensions 1 and $d-1$.

Since $n_d \leq R_d(t,k) = R_d(\tau^d, \kappa^d) = R_1(\tau, \kappa)$, we construct a one-dimensional scheme over n_d variables which uses κ cells per variable, amortized, and answers any query in time τ. The semigroup over which this construction proceeds is the semigroup of $(d-1)$-dimensional arrays, of dimension $(n_1, \ldots, n_{d-1})$, with entries taken from the original semigroup and addition defined componentwise. Thus each "cell" of this construction is itself an array of $n_1 \times \cdots \times n_{d-1}$ cells, and each addition of the construction involves $n_1 \times \cdots \times n_{d-1}$ additions. Recursively we construct a scheme $\mathcal{S}_{n'}(\tau^{d-1}, \kappa^{d-1})$ for each array given by the one-dimensional construction, which is possible because $n_i \leq R_d(\tau^d, \kappa^d) = R_{d-1}(\tau^{d-1}, \kappa^{d-1})$ for $i = 1, \ldots, d$. Remark that a scheme necessarily contains all singletons in its domain, because a singleton is a valid rectangle. Therefore, the cells used by the one-dimensional scheme are fully absorbed by the recursive constructions.

Hence the total space is,

$$\begin{aligned}|\mathcal{S}_{n_d}(\tau,\kappa)|\,|\mathcal{S}_{n'}(\tau^{d-1},\kappa^{d-1})| &= (\kappa n_d)(\kappa^{d-1}n_1\cdots n_{d-1})\\ &= kn_1\cdots n_d.\end{aligned}$$

Each $(d-1)$-dimensional scheme returns a collection of at most τ^{d-1} semigroup values in response to a query. The one dimensional query has divided the original query into at most τ subproblems, hence any query is answered in time at most $\tau^d = t$. □

The remainder of this section expresses the time bound for the constructed d-dimensional scheme in terms of the inverse Ackermann function. Define,

$$\beta_d(k,N) = \min\{\, t \mid R_d(t,k) \geq N \,\}.$$

Lemma 3.3 *For $k \geq 14^d$, we have $\beta_d(k,N) = \Theta(\beta_1(k,N)^d)$.*

Proof: From $R_d(t^d,k^d) = R_1(t,k)$ we know $\beta_d(k^d,N) = (\beta_1(k,N))^d$. Because $R_d(t,k)$ increases with k, we have $\beta_1(k,N) \geq \beta_1(k^d,N)$. To establish the inequality in the other direction we prove that given d, there exists a T such that for all $k \geq 14$, $R_1(t+T,k) \geq R_1(t,k^d)$. From this it follows that $\beta_1(k,N) \leq \beta_1(k^d,N) + T$. By,

$$\begin{aligned}R_1(t+T,k) &= R(t+T,k/2)\\ &\geq R\big(t+T-2, R(t+T,(k/2)-3)\big)\\ &\geq R_1\big(t, 2R(T,(k/2)-3)\big),\end{aligned}$$

we reduce the problem to showing that for T large enough, $2R(T,(k/2)-3) \geq k^d$. Writing $T = 2\tau+1$ and $(k/2)-3 = 3\kappa+1$, our hypothesis on k give $\kappa \geq 1$. In Lemma 3.1 we derived $R(2\tau+1, 3\kappa+1) \geq 4^{(\tau-1)\kappa}$. Chosing T so that $4^{\tau-1} \geq 14^d$, the inequality is assured. □

Lemma 3.4 *For all $i,j \geq 1$, $R(2(i+1)+1, 3(j+1)+1) \geq A(i,j) \geq R(2(i-1)+1, 3(j-1)+1)$.*

Proof: We actually have, $R(2(i+1)+1, 3(j+1)+1) \geq 3A(i,j)+4$. Recall the lower bound $R(2i+1,3j+1) \geq 4^{(i-1)j}$ from the proof of Lemma 3.1. For $i=1$,

$$R(5, 3(j+1)+1) \geq 4^{j+1} \geq 3\cdot 2^j + 4 = 3A(1,j)+4.$$

For $i > 1$ and $j = 1$,

$$\begin{aligned}R(2(i+1)+1, 7) &\geq R\big(2i+1, R(2(i+1)+1,4)\big)\\ &\geq R(2i+1,10) \geq 3A(i-1,2)+4\\ &= 3A(i,1)+4.\end{aligned}$$

For $i > 1$ and $j > 1$,

$$\begin{aligned}&R(2(i+1)+1, 3(j+1)+1) \geq\\ &\qquad R\big(2i+1, R(2(i+1)+1, 3j+1)\big)\\ &\geq\ 3A(i-1,j')+4,\end{aligned}$$

where $3(j'+1)+1 = R(2(i+1)+1, 3j+1) \geq 3A(i,j-1)+4$. Therefore $j' \geq A(i,j-1)$ and,

$$\begin{aligned}&R(2(i+1)+1, 3(j+1)+1) \geq\\ &\qquad 3A\big(i-1, A(i,j-1)\big)+4\\ &= 3A(i,j)+4.\end{aligned}$$

This establishes the first inequality. For $i=1$ or $j=1$ we directly verify that

$$A(i,j) \geq R(2(i-1)+1, 3(j-1)+1).$$

Recall that for all $i > 1$, $A(i,j+1) > A(i,j)^2$ (Lemma 2.8). Using this and the lower bound on $R(t,k)$ we derive for $i,j > 1$,

$$\begin{aligned}&R(2(i-1)+1, 3j+1) =\\ &\qquad R\big(2(i-1)+1, 3(j-1)+1\big)\\ &\qquad\quad R\left[2(i-2)+1, R\big(2(i-1)+1, 3(j-1)+1\big)\right]\\ &< R\left[2(i-2)+1, R\big(2(i-1)+1, 3(j-1)+1\big)\right]^2\\ &< R\left[2(i-2)+1, 3\left(R\big(2(i-1)+1,\right.\right.\\ &\qquad \left.\left.3(j-1)+1\big)-2\right)+1\right]^2\\ &< A\left[i-1, R\big(2(i-1)+1, 3(j-1+1)\big)-1\right]^2\\ &< A\left[i-1, R\big(2(i-1)+1, 3(j-1)+1\big)\right]\\ &< A\big(i-1, A(i,j)\big) = A(i,j+1).\end{aligned}$$

□

Lemma 3.5 *For $k \geq 14$, $\beta_1(k,n) = O(\alpha(kn,n))$.*

Proof: Recall that $A(i+1,j) \geq 2^{A(i,j)}$ (Lemma 2.9). Given i such that $A(i,k) > \log n$ we have,

$$\log n < n < 2^{A(i,k)} \leq A(i+1,k) \leq R(2(i+2)+1, 3(k+1)+1),$$

hence $2(\alpha(kn,n)+2)+1 \geq \beta_1(6k+8,n)$. If $R(2i+1, 3(k-1)+1) > n$ then,

$$\log n < n < R(2i+1, 3(k-1)+1) \leq A(i+1,k),$$

and therefore $(1/2)(\beta_1(6k-4,n)-1)+1 \geq \alpha(kn,n)$. Clearly,

$$R_1(i+2,k) > R_1\big(i, R_1(i+2,k-6)\big) > R_1(i, 6k+d),$$

for sufficiently large i, provided that we are in the third column of R_1, that is, $k \geq 14$. Therefore, $\beta_1(6k+d,n) + 2 \geq \beta_1(k,n)$. Also, $\beta_1(k,n) \geq \beta_1(k',n)$ for $k' \geq k$, by the monotonicity of R. In conclusion,

$$\begin{aligned}\beta_1(k,n) &\leq \beta_1(6k+8,n) \leq 2\alpha(kn,n)+5\\ &\leq \beta_1(6k-4,n)+6 \leq \beta_1(k,n)+6.\end{aligned}$$

□

We combine the constructions of Lemmas 3.1 and 3.2 with the above results to state this section's result.

Theorem 3.1 *In every dimension d, given a problem of size $n = (n_1,\ldots,n_d)$ and any $k \geq 14^d$, there exists a scheme using k cells per variable, amortized, which solves any query in time $O(\alpha(kN,N)^d)$. The scheme can be constructed in time proportional to its size.*

Proof: We select a t for which $R_d(t,k) \geq N$. Using Lemmas 3.2, 3.3 and 3.5, we deduce $t = O\big(\alpha(kN,N)^d\big)$. □

To summarize this section, we have given an algorithm which prestores partial sums of a multidimensional array such that only a small amount of additional computation need be done to compute any rectangular sum in the array. This is true whenever the elements have a commutative semigroup structure. Omitted from this abstract is the proof that the method can be implemented on a RAM without adding any asymptotically significant overhead. Proving the optimality of our solution is left as an open problem.

Acknowledgments. This research was supported in part by the National Science Foundation under Grant CCR-8700917.

References

[1] Agarwal, P., Sharir, M., Shor, P. *Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences*, manuscript, 1988.

[2] Alon, N., Schieber, B. *Optimal preprocessing for answering on-line product queries*, TR 71/87, The Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University, 1987.

[3] Bentley, J.L. *Decomposable searching problems*, Info. Proc. Lett. 8 (1979), 244-251.

[4] Bentley, J.L. *Multidimensional divide and conquer*, Comm. ACM, 23 (1980), 214-229.

[5] Bentley, J.L., Maurer, H.A. *Efficient worst-case data structures for range searching*, Acta Informatica 13 (1980), 155-168.

[6] Chazelle, B. *Filtering search: a new approach to query-answering*, SIAM J. Comput. 15 (1986), 703-724.

[7] Chazelle, B. *A functional approach to data structures and its use in multidimensional searching*, SIAM J. Comput. 17 (1987), 1988, 427-462.

[8] Chazelle, B. *Lower bounds on the complexity of multidimensional searching*, Tech. Rep. CS-TR-055-86, Dept. Computer Science, Princeton University. Abridged version in Proc. 27th Annu. IEEE Symp. on Foundat. of Comput. Sci. (1986), 87-96.

[9] Chazelle, B., Guibas, L.J. *Fractional cascading: II. Applications*, Algorithmica 1 (1986), 163-191.

[10] Fredman, M.L. *A lower bound on the complexity of orthogonal range queries*, J. ACM 28 (1981), 696-705.

[11] Hart, S., Sharir, M. *Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes*, Combinatorica 6 (1986), 151-177.

[12] Lueker, G.S. *A data structure for orthogonal range queries*, Proc. 19th Annu. IEEE Symp. on Foundat. of Comput. Sci. (1978), 28-34.

[13] Mehlhorn, K. *Data structures and algorithms 3: multidimensional searching and computational geometry*, Springer-Verlag (1984).

[14] Overmars, M.H. *The design of dynamic data structures*, LNCS, Vol. 156, Springer-Verlag, 1983.

[15] Tarjan, R.E. *Efficiency of a good but not linear set union algorithm*, J. ACM, 22 (1975), 215-225.

[16] Tarjan, R.E. *Complexity of monotone networks for computing conjunctions*, Annals Discrete Math. 2 (1978), 121-133.

[17] Vaidya, P.M., *Space-time tradeoffs for orthogonal range queries*, Proc. 17th Annu. ACM Symp. on Theory of Comput. (1985), 169-174.

[18] Willard, D.E. *New data structures for orthogonal range queries*, SIAM J. Comput. 14 (1985), 232-253.

[19] Willard, D.E. *Lower bounds for dynamic range query problems that permit subtraction*, Proc. 13th Internat. Coll. on Autom., Langu. and Program. (1986)

[20] Willard, D.E., Lueker, G.S. *Adding range restriction capability to dynamic data structures*, J. ACM 32 (1985), 597-617.

[21] Yao, A.C. *Space-time tradeoff for answering range queries*, Proc. 14th Annu. ACM Symp. on Theory of Comput. (1982), 128-136.

[22] Yao, A.C. *On the complexity of maintaining partial sums*, SIAM J. Comput. 14 (1985), 277-288.