November 16, 2017

Matrix multiplication in softly quadratic time

2. Real (floating-point) matrix multiplication

Approach

Challenge

There is a relatively straightforward reduction to Boolean MM. It is not actually as simple to implement as we say it is, but it is doable with an auxiliary query. We are allowed to truncate the dot product result for an output cell if the significand reaches past left-most one-bit for the dot product. We deal with indices for left-most one-bit for each component for a dot product. We solve for left-most one-bit for a dot product by using an auxiliary query for max-plus semi-ring flavor of MM. Note that the time stated for this semi-ring flavor - $O(n^2 \cdot \log^2(n))$ allows for magnitudes of size $O(n)$; to have it support magnitudes of size $O(m)$, we have time $O(n^2 \cdot \log(n) \cdot \log(m))$. We know that transdichotomous-RAM-inspired upper bound (as described later) says m' is in $O(n)$. Then, if we assume m is in $O(m')$, this gives us $O(n^2 \cdot \log^2(n))$ time, again. The same conclusion can be arrived at by having m in $O(n)$ as a straight-out requirement. This is for a single-bit-slice-to-single-bit-slice pair. Then, we have many single-to-single pairs for anchored significands for components for row from left matrix and components for column from right matrix. The reason for this is we have cross-bit-slice interaction, similar to the reason we have $O(p^2)$ for brute-force multiplication of two p-bit integers, though the primitive method that we call for pair-wise bit multiplication is different; i.e., it is a Boolean MM subproblem. We have also assumed that we can get a one-time factor speed-up of m', though we have not given a way to.

We thought that we could have Boolean MM solved via colored range counting query, but as given from various sources unchanged, it is too expensive for four-sided query.

We give an approach with better time.

Concepts

Transdichotomous RAM model and possible bounds on m' in terms of n

The transdichotomous RAM model states that m' (word size) will be in $\Omega(\log(n))$. We don't need this directly, but we will describe the chain of reasoning. Consider existence of positional number systems; having them means we have exponential relationship between number of symbols (e.g. bits) and number of representable values. To improve on this is to pursue more denseness; we could be stating that we have a lower bound for word size less than logarithmic in n. Why do we mention n? We assume that it is the largest space-use-related non-segmented variable for the problem that we have (ignoring possibly m). As it turns out, we don't need a number system that is more dense and (slightly unrelatedly) we don't need n to be representable using a constant number of words. A property of having a positional number system is also that the number of symbols needed to represent a number stays at most the same, no matter which integer it is in a certain-sized number range. An example of storing values in a segmented way is to e.g. store very large significands via superaccumulator (which we describe later); presumably we are satisfied with the different times for retrieving and mutating values for this alternate approach.

There is a second bound that may seem related, but that relies on fewer assumptions. One of the most unconcise positional number systems out there has one symbol for every distinct number, which we say we have n of. So, m' is in $O(n)$, and this is a loose upper bound.

As it turns out, we don't use either of these two assumption-laden bounds for m'.

Sanity check and bound on m in terms of n

We use the following stipulation: assuming $m = O(n^p)$, then $\log(m) = O(p \cdot \log(n))$. If $p = O(1)$ (as e.g. if $p == 1$ and $m = O(n)$; specifically $m \leq n$), then we have $\log(m) = O(\log(n))$. We believe this is fair; otherwise, there is no way to have time for q-bit integer multiplication linear in q, given that we disperse the bits for each integer.

We're only tidy when $p = O(1)$.

FFT

We use FFT multiple times, which lets us improve an aspect of our time from quadratic in m to sub-quadratic in m.

Memoizing superaccumulator and queries

For superaccumulator, bit-wise OR can be handled similarly as with addition, except it is even more well-behaved because we don't have overflow for bit-wise OR. We collapse superaccumulators towards end of each query, after which we combine using Demmel-Hida. We use superaccumulator for memoizing and query phases. We use persistence with it for both phases, as well.

When we memoize, we use superaccumulators and thereby have exact stored values. When we query, we combine collapsed superaccumulators in accordance with Demmel-Hida, which we opt to do to effectively simplify merging of superaccumulators. To do so, we take care to add summands in order of non-increasing magnitude; the price we pay is a factor of $\log_2(m)$ more bits. So, for collapsed superaccumulators, we care about left-most $m \cdot (n \cdot \log_2(m))$ bits for what we call "truncated" bit-interleaving.

Bit-interleaving

We use bit-interleaving extensively. We sometimes have ties for weighting; i.e., bits that are adjacent might be for same power of two. If we have coefficients that are multiple-bits-large (e.g. two multiple-bit coefficients are 1110_2 for 2^1 and 0110_2 for 2^0, or alternatively, $14 \cdot 2 = 28$ and $6 \cdot 1 = 6$; note that the sum is $28 + 6 = 34$), we can de-interleave and offset differently and sum to get overall sum (e.g. $10_2 \cdot 2^3 + 11_2 \cdot 2^2 + 11_2 \cdot 2^1 + 00_2 \cdot 2^0 = 2 \cdot 8 + 3 \cdot 4 + 3 \cdot 2 + 0 = 16 + 12 + 6 = 22 + 12 = 34$). Note that we don't have to jump from two distinct powers of two to distinct four powers of two and halve bit-size of a coefficient at same time; we could stay at two distinct powers of two and halve bit-size of a coefficient. The general idea is that coefficients that are multiple-bits-large become coefficients of half the bit-size with various offsets.

Truncated bit-interleaving

Specifically, truncated bit-interleaving is bit-interleaving s.t. we might need to deal with misalignment. We use left-most one bit for a collection of binary signals as anchor, we offset, truncate, and then use standard bit-interleaving.

Union-intersection

We have union-intersection arithmetic, which is borrowed from Kaplan and augmented in a novel manner.

Certain point arrangement and well-formed queries

We have two staircases as from Kaplan, except we can make sure that every point has a unique x and y value. To stay safe, we also ought to use ϵ to offset queries so that we stay off of boundaries, which have their x and y extents drawn from point x and y values. Also, queries should be sure to involve 2-d ranges that include exactly two points at a time.

Overall plan

We have three occurrences of use of idea that average subtree size for a balanced tree is $O(1)$. We use downward filtering. We have islands and assumption of well-formed point arrangement and query. We have pre-graft and post-graft; we deal with upper-layer (for x) tree and lower-layer (for y) trees. Number-theoretic transform (NTT) flavor for

FFT works with Parseval's theorem and we require that we are compatible with standard convolution. We have bit-de-interleaving for memoization. We have split vertices. We have complement and union; 3-piece break-down for lower-layer and 7-piece breakdown for upper-layer. We use x-based or y-based containment logic to determine which children we go down (for if we are at upper-layer or lower-layer, respectively). We make sure that avoiding double-counting via intersection is handled by having complement for some child. We have $O(n)$ points and $O(n)$ leaves. We find 1-2 lower-layer islands for each query. As we have $O(n)$ leaves in post-graft tree and we expect 1-2 distinct lower-layer islands, we have up to four distinct boundary leaves, and so time for finding the 1-2 lower-layer islands is $O(4 \cdot 2 \cdot \log(n)) = O(\log(n))$ as opposed to $O(\log(n)^2)$.

We note that Fourier transform is linear, so we can combine row and column source sizes and union values, then un-scale multiple times to get back to original time domain.

Misaligned significand flattening

An issue is that we have different offsets for floating-point significand contributions to memoized values for each node. Instead of needing to sort and spend time super-linear in n_s for combining n_s significand contributions, we flatten. This way, for any given power of two, we have $O(n)$ post-flatten bits for it; if we have gap between same-dot-product-term significands, we keep them separate. Merging takes time $O(\frac{m}{m'} \cdot n_s)$. We can take advantage of idea that average size of subtree for a balanced tree is $O(1)$ for the fifth time. This means that n_s is on average $O(1)$. We use flattening when we have downward memoizing (for filters) or upward memoizing (for union).

One might think we could use significand anchors instead of flattening. By opting to use anchors, we require time super-linear in n_s for performing bit-wise OR or AND for dealing with n_s significand contributions for a node.

Again, we need to consciously allow having different offsets for different component pre-bit-interleave signals (of which there are $O(n)$).

This is assuming we are within range of the current left-most one-bit for an intermediate significand.

Complement branches

For step seven, we acknowledge that we have "splitting" s.t. for each node in a balanced tree we have for the subtree rooted at that node (of size, say, n_{curr}) another tree s.t. we have points at leaves and not at internal nodes; space needed is not $O(n_{\text{curr}})$ for each of these secondary trees; the space needed is $\Omega(n_{\text{curr}} \cdot \log(n_{\text{curr}}))$. However, this doesn't have a large impact on our overall time because we effectively have (possibly large, but) multiple number of copies of each layer in our 2-d range tree.

Handling row and column source sizes

Source sizes are used. We apply FFT multiple times to row and column signals. We transform row and column post-multiple-FFT signals in isolation in a novel manner. We note that we had to modify Kaplan's row or column source size so that we have powers of two that are not always zero (as is the case for Boolean MM).

Demmel-Hida approximation

We require $\log_2(n_{\text{terms}})$-factor more space for if we plan to sum n_{terms} terms to full floating-point accuracy assuming we add them in order of magnitude non-increasing. For our 2-d segment tree query, $n_{\text{terms}} = O(2 \cdot \log(n)) = O(\log(n))$; the factor more space we need is $O(\log(\log(n))) = O(\log(n))$. For simplicity's sake, we use the looser bound $O(\log(n))$.

Persistence, superaccumulator, and tiling for downward filter

We assume that row and column signals are size $\Omega(n)$, so when we have "narrowing" of downward filter for one child of root affecting subtree of another child of root, we can allow union for first mentioned child (for which there are $O(n)$ signals that contribute) to affect

all descendants of second mentioned child (of which there are $O(n)$) s.t. we don't spend time cubic in n for this instance of downward filtering.

By doing so for downward filtering and modifying how we behave for union, we use persistence by describing node signals via replacing aligned tiles using a graph.

On collapse time, we use tiles based on version names that act as keys. We have extra layer of indirection and point to previously existing version names if we have same pattern for a tile. We also keep track of location of left-most one bit for each superaccumulator instance.

Non-full-fledged superaccumulator

We use superaccumulator only with union (bit-wise OR) and intersection (bit-wise AND); we don't have overflow. This simplifies implementation s.t. we have to implement a non-full-fledged superaccumulator.

Avoiding bottleneck for FFT s.t. we can divide time by m'

We use Zhang approach that uses word-level parallelism for NTT flavor of FFT so that we can divide time by m'. The specific approach is called "packed convolution by in-place processing" in their paper. It involves pre-packed coefficients. We note that by using Zhang's approach, we also introduce an extra $\frac{m}{m'}$ factor, as well (in a loose manner).

Intra-component FFT passes and post-FFT signal bit-interleaving

We don't interleave at top for a row or column vector; that is, at the level at which we have n components. We do this because it is unlikely the components have aligned significands and interleaving to re-group based on power of two would just be counterproductive.

For the later FFT passes, it is up to us whether we want to interleave between FFT passes. This is possibly optional; but, doing so can make it easier mentally to ensure that we are properly aligning significands when we perform union or intersection with a union tree later on. To reiterate, having interleaving between later FFT passes is not strictly required, but can help with keeping the bookkeeping organized and the choice of doing so is subjective. Opting to interleave for later FFT passes or not to both can lead to correct output. Choosing to interleave for later FFT passes may make it easier to tell when we have post-FFT signals aligned correctly.

NTT FFT last layer

Prime is chosen s.t. it minimally surpasses vector length.

For last layer, vector length always shrinks (after rounding up) to two or better. So, for this last layer, we use prime of three.

Algorithm

Times are tentative. They are subject to changes mentioned later. Also, they may be inaccurate until an actual implementation has been made.

1. Divide into four separate matrices based on sign (i.e. negative or positive).

   Time: $O(n)$

2. Start with time-domain (unaltered) length-n row and column vectors. Apply FFT multiple times to each component of current (possibly already transformed) vector until bit count is less than or equal to two. Introduce four problems for each original problem; this results from having two bits at left and two bits at right and product of two and two being four.

   Each vector starts off with and ends with length n.

   Let $r = O(\log(m))$.

   We note that $\log(m)^{\log^*(m)} = O(\log(m)^{O(1)}) = O(\log(m))$.

Time: $O(n \cdot n \cdot m \cdot r^{\log^*(m)} \cdot \log(n)/m' \cdot \frac{m}{m'}) = O((\frac{m}{m'})^2 \cdot n^2 \cdot \log(m) \cdot \log(n)) = O((\frac{m}{m'})^2 \cdot n^2 \cdot \log^2(n))$

The logarithm of m raised to iterated logarithm of m is due to repeating FFT until we get components that have bit count less than or equal to two via inflation.

3. Combine components for each post-multiple-FFT row or column via alignment and truncated bit-interleaving.

   We have $O(n)$ rows and columns.

   Each vector has length n.

   We use Demmel-Hida and, in so doing, increase time by factor of $\log_2(n)$.

   Time: $O(\frac{m}{m'} \cdot n^2 \cdot \log^2(n) \cdot \log(n)) = O(\frac{m}{m'} \cdot n^2 \cdot \log^3(n))$

4. Prepare row/column source sizes (for union-intersection arithmetic later).

   Time: $O(n^2)$

5. Construct a 2-d range tree for each of four subproblems; have one point for each row or column and have one point per leaf.

   Time: $O(n^2 \cdot \log^2(n))$ without fractional cascading

6. Memoize s.t. we bit-wise OR or bit-wise AND post-FFT-and-truncated-bit-interleaving signal for all leaves in subtree.

   We use Demmel-Hida and, in so doing, increase time by factor of $\log_2(n)$.

   Time: $O(n \cdot \log(n) \cdot (m \cdot (n \cdot \log(m)))/m' \cdot \log(n)) = O(\frac{m}{m'} \cdot n^2 \cdot \log^3(n))$

7. Upper-layer is for x, lower-layer is for y. Have 7-piece breakdown for upper-layer: L whole, R whole, A whole, (L complement w.r.t. R), (R complement w.r.t. A), (L complement w.r.t. A), (A complement w.r.t. L and R). Have 3-piece breakdown for lower-layer: L whole, R whole, L complement. Average size of a subtree for a balanced tree is size O(1). We note that if we want a union, we must use some complement. Find filtered (non-whole) memoized values s.t. we have a permutation of at most two complements active overall for the leaf (because our boundary post-graft leaves are at most two (upper-layer or lower-layer) islands deep.

   Note: Downward filters accumulate s.t. we have a permutation of at most two filters at any given point in time.

   Note: L means left, R means right, A means auxiliary (i.e. lower-layer).

   Time: Same as for step six, multiplied by a constant factor.

8. Bit de-interleave each memoized value for each node in post-graft tree and store these values at nodes.

   We use Demmel-Hida and, in so doing, increase time by factor of $\log_2(n)$.

   Time: $O(n \cdot \frac{m}{m'} \cdot n \cdot \log^2(n) \cdot \log(n)) = O(\frac{m}{m'} \cdot n^2 \cdot \log^3(n))$

9. For each query, determine union contribution from 2-d range tree (or "union tree") by taking a well-formed four-sided query and summing the memoized values for points inside the region s.t. we have no double-counting. Query chooses which trees we go down based on logic (i.e. whether we fit in more than one child subtree). We will have at most one island for upper-layer and 1-2 distinct islands for lower-layer. So, for lower-layer, we have 1-2 distinct split vertices and overall at most four boundary vertices.

   Time: $O(\frac{m}{m'} \cdot \log(n))$ per query for $O(n^2)$ queries

10. For each query, combine collapsed superaccumulator values for each of four subproblems via Demmel-Hida and sorting by magnitude.

Time: $O(\frac{m}{m'} \cdot \log^2(n) \cdot \log(n)) = O(\frac{m}{m'} \cdot \log^3(n))$ per query for $O(n^2)$ queries

We use Demmel-Hida and, in so doing, increase time by factor of $\log_2(n)$.

11. For each query, we take source size sum for row and column and subtract contribution from union tree to get intersection (i.e. dot product forward-scaled however many times we applied FFT). This is where we perform an analogue of Kaplan's Boolean union-intersection arithmetic.

    Time: $O(\frac{m}{m'})$ per query for $O(n^2)$ queries

12. For each query, apply Parseval's theorem multiple times to un-scale intermediate dot product by various vector lengths to be in time domain.

    Time: $O(\frac{m}{m'})$ per query for $O(n^2)$ queries

Necessary word-level-parallel operations

- p-way q-bit bit-interleaving

  Assume $p = O(n)$ and $q = O(m \cdot \log(m))$.

  Time is $O(\frac{m}{m'} \cdot n \cdot \log^2(n))$.

- p-way q-bit bit-de-interleaving

  Time is same as for p-way q-bit bit-interleaving plus p, which leaves the complexity unchanged.

- Number-theoretic transform (NTT) flavor of FFT

  Assume vector has number of elements $O(m)$.

  Note that we choose this flavor of Fourier transform because it supports standard convolution and deals with clean integers. While the algorithm would be simpler if we chose not to pack coefficients, packing is crucial for our running time. We approach packing s.t. we can divide time by m' via Zhang NTT FFT.

  We note again that the specific approach is called "packed convolution by in-place processing".

  Time is $O((\frac{m}{m'})^2 \cdot \log^2(m)) = O((\frac{m}{m'})^2 \cdot \log^2(n))$.

Caveats

We avoid mixing of sign via having four subproblems: +/+, +/-, -/+, -/-.

Digression on integer/floating-point approaches and large-precision

We don't deal with "variable precision". It doesn't make sense to talk about exact floating-point. Integer can be handled in exact manner, as integer is a special case of floating-point (i.e. fixed-point). Rational is handled in a satisfactory way in a different document. We can increase desired precision for floating-point approach arbitrarily (and get floating-point bignum) and for integer approach arbitrarily (and get integer bignum).

We indirectly use convolution, because we are having sub-quadratic number of bit-pairs to consider for dot product (which deals with multiplication) and we have FFT. As a result, we use NTT flavor of FFT.

The following shows that if we have binary sequences locally packed (but not globally packed), we can still get reasonable asymptotic space and time expression:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{m_i}{m'} = \frac{m}{m'}$$

$$\Rightarrow \frac{1}{n}\sum_{i=1}^{n}\left\lceil\frac{m_i}{m'}\right\rceil < \frac{1}{n}\sum_{i=1}^{n}\left(\frac{m_i}{m'}+1\right) = \frac{m}{m'}+1 = O\left(\frac{m}{m'}\right) \blacksquare$$

$$\left\lceil\frac{m}{m'}\right\rceil + 1 \leq 2 \cdot \left\lceil\frac{m}{m'}\right\rceil$$

$$\Rightarrow 1 \leq \left\lceil\frac{m}{m'}\right\rceil \Rightarrow m > 0 \blacksquare$$

The equations above show that we are allowed to round each individual significand contribution up from m_i to nearest larger multiple of m'. They also show that we can locally round space used for significand contributions into integer number of words and still be good enough for our target overall time that involves m. It is easier for us to allocate space for significand contributions now. i being in $[1, n]$ involves n arbitrarily; the upper limit can be less than n, the number of rows and columns.

Time implication

This implies ω is softly two for real (floating-point) MM for rings.

Running time

$O((\frac{m}{m'})^2 \cdot n^2 \cdot \log^3(n))$ if $m = O(n^p)$ and $p = O(1)$

The bottlenecks are related to FFT and anticipating use of collapse of superaccumulators for Demmel-Hida. Secondarily, bottlenecks are 2-d range tree creation, memoizing, bit-interleaving and bit-de-interleaving.

While on its own, it would not be enough by itself to affect overall running time, we note that an area for improvement would be to somehow avoid truncating post-bit-interleave signals and possibly save a factor of $\log(n)$ by no longer needing Demmel-Hida, which culminates in summing based on magnitude in query phase. A working approach might involve a modified superaccumulator that deals with dead space, efficient merging, and possibly persistence.

Note

m is the size of a significand in terms of number of bits.

m' is the size of a primitive float in terms of number of bits.

Full example

Not complete yet; the hardest part of this will be showing NTT flavor of FFT correctly. The most important part will be to show source size determination for rows and columns in isolation. We will give higher priority to implementing Boolean or floating-point MM first, however, as the way to calculate source sizes would then be self-evident.

$S_r = \langle 3, 5, 0, 0\rangle = \langle 11_2, 101_2, 0, 0\rangle$

$S_c = \langle 2, 1, 0, 0\rangle = \langle 10_2, 1_2, 0, 0\rangle$

$n = 4$

$S_r \cdot S_c = 6 + 5 = 11$

$S_r^0 = \langle 1, 1, 0, 0 \rangle$

$S_r^1 = \langle 1, 0, 0, 0 \rangle$

$S_r^2 = \langle 0, 1, 0, 0 \rangle$

$S_c^0 = \langle 0, 1, 0, 0 \rangle$

$S_c^1 = \langle 1, 0, 0, 0 \rangle$

$S_c^2 = \langle 0, 0, 0, 0 \rangle$

$i = 0, j = 0 \to k = 0 \Rightarrow S_r^0 \cup S_c^0 = \langle 1, 1, 0, 0 \rangle \Rightarrow |S_r^0 \cup S_c^0| = 2$

$i = 0, j = 1 \to k = 1 \Rightarrow S_r^0 \cup S_c^1 = \langle 1, 1, 0, 0 \rangle \Rightarrow |S_r^0 \cup S_c^1| = 2$

$i = 0, j = 2 \to k = 2 \Rightarrow S_r^0 \cup S_c^2 = \langle 1, 1, 0, 0 \rangle \Rightarrow |S_r^0 \cup S_c^2| = 2$

$i = 1, j = 0 \to k = 1 \Rightarrow S_r^1 \cup S_c^0 = \langle 1, 1, 0, 0 \rangle \Rightarrow |S_r^1 \cup S_c^0| = 2$

$i = 1, j = 1 \to k = 2 \Rightarrow S_r^1 \cup S_c^1 = \langle 1, 0, 0, 0 \rangle \Rightarrow |S_r^1 \cup S_c^1| = 1$

$i = 1, j = 2 \to k = 3 \Rightarrow S_r^1 \cup S_c^2 = \langle 1, 0, 0, 0 \rangle \Rightarrow |S_r^1 \cup S_c^2| = 1$

$i = 2, j = 0 \to k = 2 \Rightarrow S_r^2 \cup S_c^0 = \langle 0, 1, 0, 0 \rangle \Rightarrow |S_r^2 \cup S_c^0| = 1$

$i = 2, j = 1 \to k = 3 \Rightarrow S_r^2 \cup S_c^1 = \langle 1, 1, 0, 0 \rangle \Rightarrow |S_r^2 \cup S_c^1| = 2$

$i = 2, j = 2 \to k = 4 \Rightarrow S_r^2 \cup S_c^2 = \langle 0, 1, 0, 0 \rangle \Rightarrow |S_r^2 \cup S_c^2| = 1$

$|S_r \cup S_c| = (2 \cdot 2^0 + 2 \cdot 2^1 + 2 \cdot 2^2) + (2 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3) + (1 \cdot 2^2 + 2 \cdot 2^3 + 1 \cdot 2^4) =$
$(2{\cdot}1{+}2{\cdot}2{+}2{\cdot}4){+}(2{\cdot}2{+}1{\cdot}4{+}1{\cdot}8){+}(1{\cdot}4{+}2{\cdot}8{+}1{\cdot}16) = (2{+}4{+}8){+}(4{+}4{+}8){+}(4{+}16{+}16) =$
$14 + 16 + 36 = 30 + 36 = 66$

$|S_r| = 2 \cdot 111_2 + 1 \cdot 1110_2 + 1 \cdot 11100_2 = 2 \cdot 7 + 1 \cdot 14 + 1 \cdot 28 = 14 + 14 + 28 = 28 + 28 = 56$

$|S_c| = 1 \cdot 111_2 + 1 \cdot 1110_2 + 0 \cdot 11100_0 = 1 \cdot 7 + 1 \cdot 14 + 0 \cdot 28 = 7 + 14 + 0 = 21$

$\Rightarrow |S_r| + |S_c| = 56 + 21 = 77$

$|S_r \cap S_c| = (|S_r| + |S_c|) - |S_r \cup S_c| = 77 - 66 = 11$ ■

$|S_r| = \sum_{i=0}^{n} 1$

$\text{ones}(x) = \underbrace{11 \ \ldots \ 11_2}_{x \text{ ones}} = 2^x - 1$

$\text{shiftedOnes}(x, y) = \text{ones}(x) \cdot 2^y = (2^x - 1) \cdot 2^y$

$m_{\text{eff,r}} = 3$ (however many cases we consider for union for row powers of two)

$m_{\text{eff,c}} = 3$ (however many cases we consider for union for column powers of two)

$|S_r| = \sum_{i=1}^{n} (S_r[i-1]) \cdot \text{shiftedOnes}(m_{\text{eff,r}}, 0) = (3 + 5) \cdot 111_2 = 8 \cdot 7 = 56$ ■

$|S_c| = \sum_{i=1}^{n} (S_c[i-1]) \cdot \text{shiftedOnes}(m_{\text{eff,c}}, 0) = (2 + 1) \cdot 111_2 = 3 \cdot 7 = 21$ ■

Experiments

Nothing yet; we will pursue Boolean MM and floating-point MM first. We will compare to brute-force and Strassen.

References

Main inspiration

- Gupta et al. - Further results on generalized intersection searching problems: counting, reporting, and dynamization (1995)
- Kaplan et al. - Efficient colored orthogonal range counting (2008)
- Munro et al. - Range counting with distinct constraints (2015)
- Raz - On the complexity of matrix product (2002)

Klee contour

- Boissonnat et al. - Voronoi diagrams in higher dimensions under certain polyhedral distance functions (1998)
- Chan - Klee's measure problem made easy (2013)

Four-sided colored range counting query

- Arge et al. - Cache-oblivious planar orthogonal range searching and counting (2005)
- Larsen et al. - Near-optimal range reporting structures for categorical data (2013)
- JaJa et al. - Space-efficient and fast algorithms for multidimensional dominance reporting and counting (2004)
- Patil et al. - Categorical range maxima queries (2014)

Grids

- Nekrich - Orthogonal range searching on discrete grids (2014)

Catastrophic cancellation and floating-point numbers

- Demmel et al. - Accurate and efficient floating point summation (2004)
- Goodrich et al. - Parallel algorithms for summing floating-point numbers (2016)
- Neal - Fast exact summation using small and large superaccumulators (2015)
- Zhu et al. - Correct rounding and a hybrid approach to exact floating-point summation (2009)
- Zhu et al. - Algorithm 908: Online exact summation of floating-point streams (2010)

Bit-interleaving

- Anderson - Bit Twiddling Hacks - Interleave bits by binary magic numbers (2005)
- Warren - Hacker's Delight - Section 7-7 - General permutations, sheep and goats operation (2012)
- Skilling - Programming the Hilbert curve (2004)
- Lawder - Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curve (2000)

NTT FFT with word-level parallelism

- Zhang - Fast convolutions of packed strings and pattern matching with wildcards (2017)