February 25, 2017

Boolean and real (floating-point) matrix multiplication in softly quadratic time

First, we tackle Boolean matrix multiplication.

One way to handle it would be to use two segment trees. We use three-sided queries in plane and have a point arrangement similar to figure 3.6 from Kaplan, but arranged to descend from upper left to lower right. We avoid edge cases by offsetting our queries by ϵ in certain directions. More details about the point arrangement are in section 5.1 from Kaplan, which describes what we would do for AA^T. We switch left and right staircases (from A at left and B at right to B at left and A at right) to handle AB. We actually need four-sided query for our point arrangement to exhibit combinatorial flexibility at a cheap enough price. We can make do by noting that if the open edge is in positive y direction, we are including a prefix of B row elements (or A column elements, resp.), so we can pre-process the elements at right to subtract elements from point location immediately to left on the same side as a reaction. Then, we can live with three-sided query and we use subtraction (so we can use rings, but not semi-rings). We use color range counting query and orthant union decomposition. As per theorem 2.5 from Kaplan, we have three-sided boxes $[a, b] \times (-\infty, c]$, so $d = 2$, $k = 1$, we cut $[a, b]$ into $[a, \infty) \times [-b, \infty)$ and transform points using partial component doubling as described by Kaplan. Then, time for construction and query phases is $O(n^{\lfloor (d+k)/2 \rfloor} \cdot \log^{d+k-1}(n)) = O(n^2 \cdot \log^2(n))$ with fractional cascading. The time for construction and query phases is $O(n^2 \cdot \log^3(n))$ without fractional cascading. Our color range counting query gives us a union; we find intersection by pre-processing and storing number of one bits from an A column or B row, and using these values for a particular A column and a B row and subtracting union. Note that we have again used subtraction by turning union into intersection. This implies ω is softly two for Boolean MM for rings. Kaplan did not make this connection possibly because four-sided query appears to take time at least quadratic in number of points.

Next, we tackle real (floating-point) matrix multiplication.

This case is similar to Boolean MM, except that we have m bits per A or B matrix element. m' is the number of bits for a built-in data-type. We associate each bit with a bit-slice, i.e. a power of two. Naive time is $O(m \cdot n^2 \cdot \log^3(n))$ without fractional cascading. We rotate point arrangement ninety degrees CCW so that we have staircases headed from lower left to upper right and one of the staircases is vertically shifted w.r.t. the other. We re-group bit-slices from size one to size m', so that we have aligned groups of adjacent bits of size m' at each location for an A column or B row in the point arrangement in a satisfactory amount of time. As a result, we now have $n \cdot m/m'$ colors. We construct and perform color carpentry (see Kaplan) in bit-wise parallel. We query by taking advantage of locality. For construction, we note that along x, we have roughly repeated units with the small caveat that the units move up as we go to the right; we use placeholders and get an m'-factor speed-up, assuming we can tie these component segment trees together into one later on, and we use modular arithmetic for colors to tell which component segment tree to investigate; the component segment trees have interleaved x locations. We note that partial component doubling does not interfere with point arrangement for construction because we don't need variation in a (specifically *assuming* we are determining decomposition for a union of orthants and assuming $[a, b]$ describes y for rotated arrangement), which leaves us with two components b and c, which is same number components that we had pre-component-doubling-transform. For the pre-construction union decomposition phase that we call "color carpentry", we use generic overlaid orthant union decomposition resulting from considering m' adjacent bit-slices for a certain color, noting that if we break the groups apart, bit-wise products are still constant and one; we do this to get an m'-factor speed-up. Also, we store a color weight with each lower apex (i.e. a lower boundary-residing point). For query, we get an m'-factor speed-up, because we bit-wise-parallel-handle each color because we can perform bit-wise multiplication by using bit-wise AND. Time is $O(\frac{m}{m'} \cdot n^2 \cdot \log^2(n))$ with fractional cascading and $O(\frac{m}{m'} \cdot n^2 \cdot \log^3(n))$ without fractional cascading. We still use union query to effectively have intersection query. We also use subtraction, so we can use rings, but not semi-rings. We also have to remember to take into account power of two offsets. This implies ω is softly two for real MM for rings.

Certain MM flavors (e.g. complex, rational, small-integer-weight min-plus) lend themselves well to reduction to real MM.

We tried unsuccessfully to tackle semi-ring-supporting Boolean and real MM directly and instead defer to reductions.

We note that Strassen and Le Gall require subtraction.

Note that we can at a small penalty for time handle exact matrix multiplication by predictably increasing precision for floating-point MM. We also can support exploitation of sparsity.

References

- Kaplan et al. - Efficient colored orthogonal range counting (2008)
- Munro et al. - Range counting with distinct constraints (2015)