Quadrarithmic-Time Geometric Exact Boolean Fast Matrix Multiplication

Zhijia (Brian) Liu / 刘之佳
Email: bzliu94@gmail.com

September 16th, 2018

Abstract We use a geometric approach to solve Boolean matrix multiplication (hereafter MM) in $\mathcal{O}(n^2 \cdot \log^3(n))$ time. We use bounds based on how far n vectors on an n-dimensional hypercube can be from each other at worst and use always-successful two-distortion-ratio approximate nearest neighbor (ANN) with approximate bichromatic closest pairs and Prim's method via Chan [C02] to determine an approximate MST and topological ordering to memoize for an overcomplete dictionary's cut-out/expiry rectangles. We use Chan's approach because it is amenable to word-packing in future; an important part is that we are happy with epsilon of one. Then, for query time, we use for each main query at most three subqueries based on dictionary. The main idea that makes the equidistance bound useful is that it decays as we add vectors. Then, we relate Euclidean and Manhattan distances. We assume perfect packing and n-dimensional hypercube surface area divided by $(n-1)$-dimensional hypercube volumes. Then, we use telescoping/prefixes (to deal with three-sided queries as opposed to having directly four-sided queries) and color carpentry on top of geometric configuration as from Kaplan [KRS08]. We draw ideas from Gupta [GJS95], as well. We use union-intersection arithmetic as from Kaplan. We have staircase pairs and introduce z-levels for rectangle arrangement. A trivial case we support is rectangular input matrices. Future ideas include number-of-non-zeroes-based sparsity exploitation (which is more ambitious than just having empty A rows or B columns, which we support), parallelism, cache exploitation. Our approach is cache-oblivious, deterministic. We compare to optimized brute-force/nearly-brute-force commonly-used approach library BLAS and observe cross-over for Boolean case at $n =$??. We note that Raz says, under certain assumptions, minimum time for MM is $\mathcal{O}(n^2 \cdot \log(n))$ [Raz02]. We acknowledge that the reason we are able to make progress on such a MM problem is that it is a special case and we are taking advantage of the concept of strength reduction.

Keywords linear algebra, matrix multiplication, complexity, computational geometry

1 Introduction

We were inspired by CS 61C machine architecture course matrix multiplication project at UC Berkeley from summer of 2011.

Also, for the future, we will have floating-point case s.t. we have exactness or numerical robustness even in the face of possible catastrophic cancellation (which we think is possible to handle via tiling and a persistent prefix tree and a secondary representation based on run-length encoding (RLE) as we can then efficiently handle binary overflow/underflow for floating-point addition or subtraction). Further, for floating-point case, we will have use of word-RAM and packing. We believe that it is possible that our Boolean MM approach can be used almost as a black box (i.e. we introduce word-packing for it and that means word-packed ANN) for floating-point case along with NTT, Zhang word-packed linear convolution [Z17], circular convolution in terms of linear convolution, Rader FFT as circular convolution [R68], bit spread/gather as via Nuetzi p-way q-bit bit interleaving [N13A, N13B], Parseval's theorem, assumption that $\log(m) = \mathcal{O}(\log(n))$. This means time for exact floating-point case of tentatively $\mathcal{O}(n^2 \cdot \log^3(n))$.

We also support semiring flavors. We are unsure as to whether our Boolean MM approach can be generalized for all semiring flavors. The time we have for Boolean case is assuming we don't use fractional cascading and lowest-level interval tree. For semiring flavors (min, $+$), (max, $+$), (min, max), (max, min), (min, $\leq$), (max, $\leq$), (min, $\times$), (max, $\times$), we use tower arrangement and critical min. distance range closest-pair query via Sharathkumar [SG07] and time is $\mathcal{O}(n^2 \cdot \log^2(n))$.

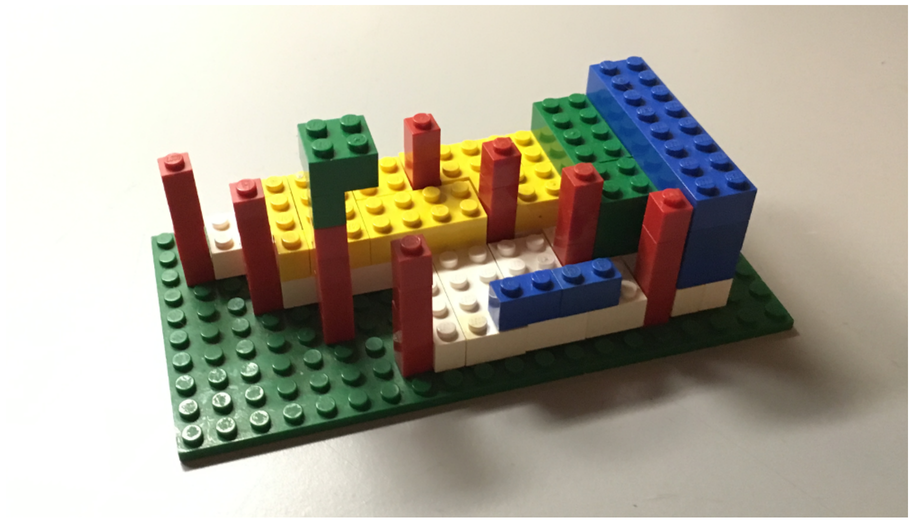

Figure 1: Pair of descending staircases with many colors for rectangle arrangement

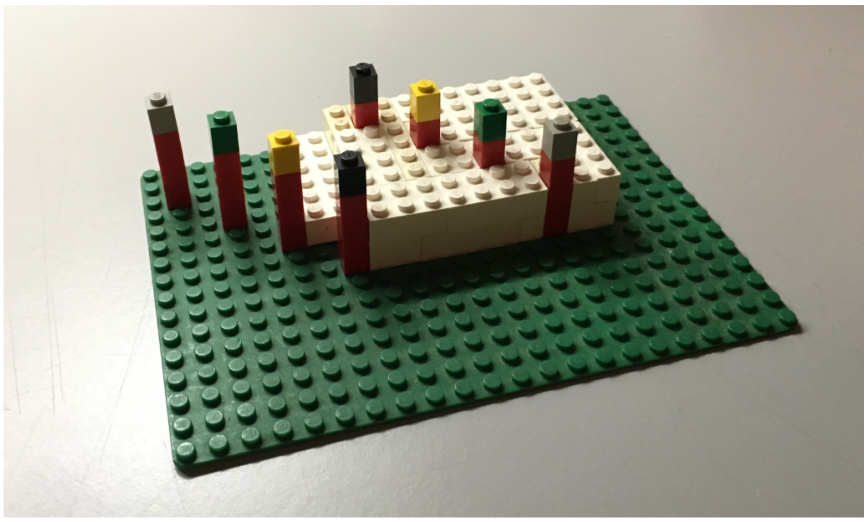

Figure 2: For fixed color, we have varying z-levels that facilitate using telescoping

We use n-dimensional hypercube surface area with (n - 1)-dimensional primitive hypercube volumes instead of with (n - 1)-dimensional primitive hypersphere volumes because the former is plausible; we don't expect for a given fixed extent (e.g. radius) that volume approaches zero as dimension increases; we want volume to be at least as large for a larger dimension.

Also, there is no need to scale unit n-dimensional hypercube or distances associated with that hypercube; length of largest diagonal sqrt(n) can in general be much larger than between one and two; this is where intuition may be different from what is really the case – i.e. particularly when n (i.e. the dimension) is quite large.

How we handle floating-point case is using word-packing for boolean subproblems s.t. boolean case is almost a black box; we use (m * n)-dimensional hypercube surface area and decay.

For approximate nearest neighbors, LSH has a success probability (i.e. it is not always correct). Also, kd-tree has large worst-case query time, even if we use a flavor such as sliding-midpoint.

Chan for ANN uses Arya's BBD-tree and uses cones.

Clarkson provides an alternative to Chan, but their approach seems at least as hard to implement.

For floating-point case, have four boolean MM subproblems that each deal with n * m * log(m) bits via point-wise multiplication. It is thus important to speed-up ANN via word-packing and a factor of m'.

Chan's approach from 2002 is deterministic (i.e. it has no failure rate, it has no randomization for running time), potentially dynamic, has good running time bound, can be word-packed.

We use for floating-point case an algorithm from Pagh; it is a Las-Vegas-style locality-sensitive hashing (LSH) algorithm associated with expected time. For Boolean case, we use Chan 1998 modified BBD-tree; the BBD-tree came from Arya and has deterministic time.

We hand-wave and tune to get neighbor-finding to be effective. We don't take integral and average and calibrate-related-scale to get logarithmic factor; we sample and calibrate-related scale to get a constant factor; this is plausible because of similar results for specific linear codes (e.g. Hamming and Reed-Solomon). We admit that this is surprising to us. This is also why we can use Pagh's Las-Vegas-style LSH (for floating-point case with word-packing and for Boolean case). Note that decay is not necessarily an overall bottleneck (though it is for us to be able to use Las-Vegas-style LSH).

We have two as average l1-norm distance with n points in n-dimensional space on a unit hypercube for smallest case (i.e. n == 2); we hover around two for larger n. We have values slightly larger than n, which we attribute to a "discretization penalty". Our predictions assume continuous coordinates. In principle, max. MST total weight is attributed to maximal equidistance. Any deviation from equidistant point arrangement should lead to lower max. MST total weight. However, this may not be true; considering hypercube and hypersphere with same surface area with their centers aligned, we have places where the hypercube protrudes from the hypersphere. This protrusion as a fraction of radius for the hypersphere (i.e. our "protrusion ratio") approaches zero as n approaches infinity, we predict. We consider sphere even though up to this point we have been using only cubes because equidistant points (if equidistance is possible) sit on the surface of a sphere, generally. However, we make up for this by considering "approved spheres", whose volume and surface area are based on those for hypercubes. Intuitively, for fixed volume or surface area as dimension increases leads radius to approach infinity. While our protrusion ratio approaches zero as n approaches infinity, we then need to consider n == 2 and n == 7 (given that surface area for a sphere peaks at n == 7, assuming n is an integer). Our continuous analysis suggests we will continue to hover for average distance for max. MST at around two. This discretization penalty then gives us a bound w.r.t. this two. Even though radius for hypersphere with fixed surface area as dimension approaches infinity approaches infinity, protrusion will be dwarfed by radius and we then have no reason to expect that our max. MST overall weight will increase via discretization. It's worth emphasizing that as dimension increases, the main hypercube has surface area that grows as well.

Volume and surface area for a unit hypercube grow in very different ways, unlike with volume and surface area for unit hypersphere. The key idea is distortion of max. MST overall weight via discretization. However, we don't need to enforce having zeros and ones for components to be able to describe trend for protrusion ratio for primitives as dimension approaches infinity. This has a better chance of working out than considering only primary hypercube and primary hypersphere because of the way volume grows for a unit hypercube. We should not feel obligated to round at this point.

Hypercube and hypersphere for primitive are NOT interchangeable; this is partly why we use concept of "approved hypersphere", which has properties based on an associated hypercube. The Wikipedia article makes radius based on fixed volume as dimension approaches infinity look as if coefficient approaches one from the left; it does not and instead grows in O(sqrt(d)). With hypercube primitive, we expect effective factor of one; with hypersphere primitive, we expect effective factor of n; this is even after noting that with hypercube, l1-norm distance is in O(n * side length) and with hypersphere, l1-norm distance is in O(sqrt(n) times radius). We note that protrusion ratio for hypercube aspect is affected more directly by hypercube diagonal distance from center of hypercube than by side length. Also, protrusion ratio for hypersphere aspect is affected by radius.

CASE ONE: For fixed volume and as dimension increases, radius for a primitive hypersphere grows and side length for a primitive hypercube shrinks (or decays). This lack of monotonically growing (i.e. we have monotonically shrinking) volume for fixed radius as dimension increases for a primitive hypersphere make hypercube a better choice for primitive shape. CASE TWO: Then, protrusion ratio is affected as dimension increases; primary SA is invariant (which gives similar behavior as with primary V being invariant), which leads to diagonal growing faster than radius grows. This happens because, similarly, for fixed SA or V for primary shape divided by n that ends up being > 1, as dimension increases, radius grows and is in O(sqrt(d)) and side length shrinks and is >= 1 and diagonal grows in O(sqrt(1 / 2 * s * d)). The orders of growth are roughly the same, except if fixed SA or V is significantly and/or consistently larger than one (which is most often the case).

From previous paragraph's analysis, we expect diverging protrusion ratio. We note that we have about an extra factor of two because protrusion ratio needs to be approximately doubled to be applicable not to points but to distances; each distance is associated with a pair of points. (Specifically, an expression we ought to use is approximately (2 * r__protrusion - 1) instead of 2 * r__protrusion.) Then, the extra factor incurred is approximately in O(r__protrusion * 2) = O(sqrt(1 / 2 * s) * 2) = O(sqrt(s)), which may be affordable enough, given that amount of volume allocated to a primitive shape is two. (Note that we have s in (1, 2], then, given that dimension is >= 2.) This, again, is for the end that is discretization penalty. s = V ˆ (1 / (n - 1)), with V == 2 and n approaching infinity. maximal s for allowed range of n in [2, infinity) is at n == 2 s.t. that maximal s is two. This gives doubled protrusion ratio upper bound of O(sqrt(1 / 2 * 2) * 2) = O(sqrt(1) * 2) = O(1 * 2) = O(2). Thus, for discretization penalty, we expect an extra upper-bound factor for time of around 2. This agrees with our small amount of data points for max. MST overall weight and associated average edge weight (which is associated with XOR one-count and ANN and effective factor for time, which comes from effectively n ˆ -1 * n = 1, which we multiply by 2 ostensibly), because e.g. 2.33 <= 2 * 2 = 4. This is good news, then, because then we have sub-log time (though not even log(log(n)) would have been enough); more to the point is that we have constant time as effective factor. Then, we can start to think about Las-Vegas-style LSH for future for word packing for floating-point case (and not just Boolean case).

Somehow, we amazingly got past the upward fluctuation for the associated average edge weight (for a max. MST overall weight for a dimension value) of around two by invoking discretization penalty. Actually, we predict a growth from and not just fluctuation around two. Still, this is handleable and is associated with a cost we can afford.

The protrusion ratio seems to approach some fixed value, which is particularly visible when we go from n in {10, 20, 1000} to n == 10000. Note that we left out constant factors that participate in Stirling's approximation or from pi or gamma function in SA or V expressions. While we have not yet formulated exactly how to get the value we converge to (which is protrusion ratio of around 0.657), the fact that we converge at all agrees with our idea that we have an upper bound for the ratio that is fixed and that is based on coefficients or terms such as sqrt(1 / 2 * s) * 2 <= sqrt(1 / 2 * 2) * 2 = 2. (Actually, put another way, we have a scale factor of around three, it seems. Of course, this is alternatively framed as a ratio of (3 - 1) : 3 == 2 : 3.) Note that 3 is different from 4.)

To recap, we seem to have an asymptote for protrusion ratio, which is essentially as good as ratio that reaches one.

Las-Vegas-style LSH will be used for f.p. case, but with what parameter values? For Boolean case, we will use BBD-tree.

We should note that n equidistant points in n dimensions tend to lie on surface of a hypersphere.

We absolutely have an asymptote s.t. as n increases we approach a fixed value for protrusion ratio, given numerical experiments via sampling s.t. for specific n values (e.g. 10, 20, 100, 10000, 100000) we calculate an estimate for the associated protrusion ratio (e.g. we approach around 0.6577). Future work can be to determine the exact formula for the value we approach.

We have rectangle case. If number of A rows or B columns is more than number of colors, we have more points than dimensions, which means good packing. If number of A rows or B columns is less than number of colors, we break rectangular case into many small square problems; these problems are independent of each other for (+, x) ring.

With f.p. case, we also must have m * d points in m * d dimensions and this also means we have good packing.

Parameters for Las Vegas LSH for f.p. case (with word packing) are c = 1 / 4 * log_2(n) (which, notably, may be much larger than 2), r = 4, w = m'. Limit as n approaches infinity of rho is limit as n approaches infinity of 4 / log_2(n) is zero. Size is O(d * n ^ (1 + rho)) = O(n * n ^ (1 + 0)) = O(n * n) = O(n ^ 2) bits. Query time is O(n ^ rho * (1 + d / w)) = O(n ^ 0 * (1 + n)) = O(1 * (1 + n)) = O(n). The factor for approximation may be larger than two, which will impact running time for ANN application, which is memoizing for dictionary.

Hypercube and hyperspheres are in principle interchangeable for primitives. However, spheres do not pack perfectly and cubes tile almost perfectly. We consider packing constant. We take a detour and use hypercube primitives because as dimension increases, max. density for hypersphere shrinks quickly. This explains why we had gut feeling that decay is justifiable if we use hypercube primitives. To use hypersphere primitives, we would need a good formula for max. density based on dimension, which does not currently seem to be well-known. Presumably if we knew this formula, decay would occur as well as opposed to growth when solving for radius and we have fixed volume and increasing dimension. For example, in 24 dimensions, hyperspheres have max. average density (or packing constant) of around 0.000000471. From this perspective, we use hypercube primitives to take a more well-beaten path and avoid currently not-well-known max. packing for hyperspheres in large dimension. Protrusion uses hypersphere because equidistant points tend to lie on surface of a sphere. As long as we use appropriate scaling to go to l1-norm and we are happy with bound tightness, we can choose either hypercube or hypersphere for primitive (i.e. for hypercube we scale by n and for hypersphere we scale by sqrt(n)).

2 References

Chan, Timothy M. 1998. "Approximate Nearest Neighbor Queries Revisited." *Discrete & Computational Geometry* 20 (3). Springer: 359–73.

———. 2002. "Closest-Point Problems Simplified on the RAM." In *In Proc. 13th ACM-SIAM Sympos. on Discrete Algorithms*. Citeseer.

Gupta, Prosenjit, Ravi Janardan, and Michiel Smid. 1995. "Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization." *Journal of Algorithms* 19 (2). Elsevier: 282–317.

Kaplan, Haim, Natan Rubin, Micha Sharir, and Elad Verbin. 2008. "Efficient Colored Orthogonal Range Counting." *SIAM Journal on Computing* 38 (3). SIAM: 982–1011.

Nuetzi, Gabriel. 2013a. "How to Compute a 3-d Morton Number (Interleave the Bits of 3 Ints)." https://stackoverflow.com/questions/1024754.

———. 2013b. "Produce Interleaving Bit Patterns (Morton Keys) for 32-Bit, 64-Bit, and 128-Bit." https://stackoverflow.com/questions/18529057.

Rader, Charles M. 1968. "Discrete Fourier Transforms When the Number of Data Samples Is Prime." *Proceedings of the IEEE* 56 (6). IEEE: 1107–8.

Raz, Ran. 2002. "On the Complexity of Matrix Product." In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, 144–51. ACM.

Sharathkumar, R, and Prosenjit Gupta. 2007. "Range-Aggregate Proximity Queries." *IIIT Hyderabad, Telangana* 500032.

Zhang, Meng. 2017. "Fast Convolutions of Packed Strings and Pattern Matching with Wildcards." *International Journal of Foundations of Computer Science* 28 (03). World Scientific: 289–307.