Assuming that multiplication of n-by-n matrices can be performed by a bilinear program that uses n^ω multiplications, we present an $O(n^{1-\frac{2}{\omega}})$-round matrix multiplication algorithm for the congested clique model (with n vertices/processors). Specifically, we abstract the presentation of [1, Sec. 2], which relies on the routing scheme of [2] and on prior parallel algorithms for matrix multiplication.[1] We first present an algorithm that is based on the straightforward bilinear program for computing matrix multiplication, and next present an algorithm that uses any bilinear program. Needless to say, the first algorithm (presented in Section 2), can be derived as a special case of the second algorithm (presented in Section 3), but the derived presentation is different. In retrospect, although the first algorithm is derived more directly from the bilinear program, the second algorithm is not more complex (although it does use another conceptual insight).

1 On bilinear programs for matrix multiplication

A general bilinear program for computing the product, denoted $P = (p_{i,j})_{i,j\in[d]}$, of two d-by-d matrices, denoted $S = (s_{i,j})_{i,j\in[d]}$ and $T = (t_{i,j})_{i,j\in[d]}$, has the form

$$p_{i,j} = \sum_{w\in[m]} \gamma_{i,j}^{(w)} \left(\sum_{u,v\in[d]} \alpha_{u,v}^{(w)} s_{u,v} \right) \cdot \left(\sum_{u,v\in[d]} \beta_{u,v}^{(w)} t_{u,v} \right) \tag{1}$$

where the $\alpha_{u,v}^{(w)}, \beta_{u,v}^{(w)}, \gamma_{i,j}^{(w)}$'s are scalars and $m = m(d)$ is the number of multiplications used.

The salient feature of matrix multiplication, which we use in this memo, is that the foregoing equation (i.e., Eq. (1)) holds also when applied to a partition of n-by-n matrices to d^2 submatrices, which is each a n/d-by-n/d matrix. That is, we consider a partition of the n row (resp., columns) to d blocks, each holding n/d consecutive rows (resp., columns), and let $S_{i,j}$ (resp., $T_{i,j}$ and $P_{i,j}$) be the $(i,j)^{\text{th}}$ submatrix of S (resp., T and P). Then, for every $i, j \in [d]$, we have

$$P_{i,j} = \sum_{w\in[m]} \gamma_{i,j}^{(w)} \left(\sum_{u,v\in[d]} \alpha_{u,v}^{(w)} S_{u,v} \right) \cdot \left(\sum_{u,v\in[d]} \beta_{u,v}^{(w)} T_{u,v} \right) \tag{2}$$

where here $m = m(d)$ is a function of d. Note that this holds for any d that divides n.

2 Using the straightforward bilinear program

The straightforward program uses $m(d) = d^3$ and has the form

$$P_{i,j} = \sum_{k\in[d]} S_{i,k} \cdot T_{k,j} \tag{3}$$

for every $i, j \in [d]$. Recall that $S_{i,j}, T_{i,j}$ and $P_{i,j}$ are n/d-by-n/d matrices. We shall set $d = n^{1/3}$ and so have $m(d) = n$. We next decompose Eq. (3) as follows.

[1] We are not clear about the credits, and leave to others the task of sorting them out.

(i) $P^{(k)}_{i,j} = S_{i,k}T_{k,j}$ for every $i,j,k \in [d]$, and

(ii) $P_{i,j} = \sum_{k\in[d]} P^{(k)}_{i,j}$ for every $i,j \in [d]$.

Now, given that $d = n^{1/3}$, it is most natural to compute (i) by assigning a vertex/processor to each triplet $(i,j,k) \in [d]^3$. Each such processor has to obtain $2 \cdot (n/d)^2 = 2n^{4/3}$ bits of the $2n^2$-bit input, which is initially distributed "uniformly"/regularly among the n processors (such that each processor holds n entries of each matrix). The key (and yet simple) observation is that the distribution pattern required for performing (i) is regular; that is, each bit is required by an equal number of processors. Hence, the routing scheme theorem holds, and $O(n^{4/3}/n)$ rounds suffice.

Turning to the computation of (ii), note that there are d^2 computations, and each requires d matrices (each having $(n/d)^2 = n^{4/3}$ entries). So we can just partition each such computation among d processors (i.e., we do d^3 sub-computations, each referring to $(n/d)^2/d = n$ entries in a generic n-by-n matrix). Again, each processor needs $d \cdot n$ bits specifically, the bits in the d matrices that correspond to the entries handled by this processor. Since this is a regular/uniform partition, we can use the routing scheme again.

(If the n^2 output bits are not partitioned as required, then we can just fix that by $O(1)$ more rounds. This is avoided in the presentation of [1], which just uses the adequate partition, but this is immaterial.)

On second thought, computation (ii) may be rewritten as (ii') $P = \sum_{k\in[d]} P^{(k)}$, where $P^{(k)} = (P^{(k)}_{i,j})_{i,j\in[d]}$ which suggests that it can be implemented using any n-way equi-partition of the n-by-n matrix, where each party holding a bit of $P^{(k)}$ sends this bit to the designator processor.

3 Using any bilinear program

Like in the special case (of the straightforward program), we shall set d such that $m(d) = n$. Our overall plan is to compute P (by decomposing Eq. (2)) as follows.

(i) For every $w \in [n]$, we compute the matrix $R^{(w)} = A^{(w)}B^{(w)}$, where $A^{(w)} = \sum_{u,v\in[d]} \alpha^{(w)}_{u,v} S_{u,v}$ and $B^{(w)} = \sum_{u,v\in[d]} \alpha^{(w)}_{u,v} T_{u,v}$.

(ii) For every $i,j \in [d]$, we compute $P_{i,j} = \sum_{w\in[n]} \gamma^{(w)}_{i,j} R^{(w)}$.

Assuming that we can compute all $A^{(w)}$'s and $B^{(w)}$'s, and that these entries are partitioned regularly among n processors, we assign a vertex/processor to each $w \in [n]$ and let it compute the product $R^{(w)}$. This requires sending two n/d-by-n/d matrices to each processor, which can be done in $O(\frac{(n/d)^2}{n}) = O(n/d^2)$ rounds, using the routing scheme. However, to implement (i) we also have to show how to compute all the $A^{(w)}$'s and $B^{(w)}$'s, where in each case we need to compute n different linear combinations of the same d^2 matrices, where each matrix holds $(n/d)^2$ entries.

We can actually afford to compute n different linear combinations of n different $(n/d)^2$-bit long strings, by assigning to each processor $\frac{(n/d)^2}{n} = n/d^2$ entries (i.e., equi-partition $[(n/d)^2]$ to n parts), and letting it compute the corresponding linear combinations. Note that providing each processors with the $n \cdot n/d^2$ relevant bits can be done in $O(n/d^2)$ rounds. The foregoing also handles the computation of (ii), which calls for computing d^2 different linear combinations of some n matrices (i.e., the $R^{(w)}$'s).

Conclusion. We have shown that Eq. (2) can be computed in $O(n/d^2)$ rounds, where d was set such that $m(d) = n$ (i.e., d-by-d matrix multiplication can be computed via Eq. (2) while using n multiplications). Assuming that $m(d) = d^\omega$ for some $\omega \in [2,3]$, we use $d = n^{1/\omega}$, and get round complexity $O(n^{1-\frac{2}{\omega}})$. Note that $\omega < 2.373$ yields $1 - \frac{2}{\omega} < 0.1572$.

Digest. Note that, unlike in the simple algorithm (of Section 2), the current algorithm is based on the observation that a processor can compute *several* Boolean functions of the *same* inputs at the same cost as computing a single Boolean function of these inputs, where the cost is the number of rounds required to provide this processors with the relevant inputs. Equivalently, the cost of computing a function is independent of the range of the function; it only depends on the size of its domain.

Specifically, in Step (i) we computed n different pairs of (n/d)-by-(n/d) matrices, which all depend on the same pair of n-by-n matrices such that each output bit depends on d^2 input bits, and then computed the product of each pair of matrices. Hence, in the first substep (of Step (i)) we partition the $(n/d)^2$ relevant entries among n processors such that each processor is responsible for n/d^2 entries, provide each processor with the relevant $2 \cdot d^2 \cdot n/d^2$ input bits, and let it compute the relevant $2 \cdot n \cdot n/d^2$ output bits. In the second substep, the latter bits are redistributed so that each processor gets a pair of (n/d)-by-(n/d) matrices, and just computes their product. The key observation here is the first distribution of bits refer to $2n$ bits per processor, whereas the second refers to $2n^2/d^2$ bits per processor, while both distribution schemes are regular (and so can be performed in $O(1)$ and $O(n/d^2)$ rounds, respectively).

Likewise, in Step (ii) we computed an n-by-n matrix (i.e., d^2 different (n/d)-by-(n/d) matrices) such that each output bit depends on n input bits (which are taken from n different (n/d)-by-(n/d) matrices). Hence, the n^3/d^2 bits of the n different (n/d)-by-(n/d) matrices are redistributed (according to a regular pattern), and this is performed in $O(n/d^2)$ rounds.

References

[1] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Comput.*, Vol. 32 (6), pages 461–478, 2019.

[2] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *32nd PODC*, pages 42–50, 2013.