1. CSE 6220: HW5

Note: This assignment is submitted by: **Somdut Roy (GTID: sroy86)**

1. The proposed algorithm requires the following two steps:
   (a) Partial dot products of the subsets of the matrix and the vector on each processor and then store the local $y[j]$ in the processor. This step requires no communication.
   (b) All-to-all reduction to add the local local $y[j]$ vectors onto each processor. Each step in this involves all-to-all permutation folowed by pairwise addition (reduction) of the elements. The communication time taken in this is $O(\tau log(p) + \mu mp(\frac{1}{2} + \frac{1}{4} + ...)) = O(\tau log(p) + \mu mp)$.

   Runtime:
   (a) Computation: $O(\frac{n^2}{p})$.
   (b) Computation: $O(n)$
   Communication: With $m = \frac{n}{p}$, $O(\tau log(p) + \mu.\frac{n}{\not{p}}.\not{p}) = O(\tau log(p) + \mu n)$.

2. To make sure a processor holds a constant number of matrix elements at all times, it is essential that the mess can be assumed to be a combination of concentric squares holding alayer of one element at the edge. As we go to the inner square the length of the side is reduced by 2 (with exclusion of one vertex on both ends as compared to the outer square). The outermost square with the largest perimeter will dominate the run time.The numbers are shifted one at a time in required directions for making the transpose possible in each squares independently. the elements move in one of four directions based on its desired final location.

   Runtime: Most of the time will be taken by the elements in $(0, n-1)$ and

$(n-1,0)$ positions to swap places. Both the elements will move $2(n-1)$ places along the outermost square to swap places. So runtime will $O(2(n-1)) = O(n)$.

3. The dividend $d(x)$, divisor $p(x)$, quotient $q(x)$ and remainder $r(x)$ are four polynomials of degrees m, n, $(m-n)$ and atmost $(n-1)$ respectively for $0 \leq n \leq m$ that are related by $d(x) = p(x) * q(x) + r(x)$.

   Now, $\frac{d(x)}{p(x)} = \frac{\sum_{k=0}^{m} d_k x^k}{p(x)} = (\sum_{k=0}^{m} \frac{d_k}{x^{m-k}}) \frac{x^m}{p(x)}$

   Now we write x^m in the quotient-remainder form: $x^m = p(x) * q_1(x) + r_1(x)$, where r_1 has a degree less than n.

   Hence, $\frac{d(x)}{p(x)} = (\sum_{k=0}^{m} \frac{d_k}{x^{m-k}})(q_1(x) + \frac{r_1(x)}{p(x)})$.

   Now, to solve this problem, a efficient alogrithm is needed to divide x^m by $p(x)$. For that, we need to find the value of a reciprocal polynomial $s(x)$ which is given by $\lfloor \frac{x^{2n}}{p(x)} \rfloor$. This calculation is doable if n is of the form of that it is one less than a perfect power of two. If that not the case, we multiply both $d(x)$ and $p(x)$ so that the divisor of the new problem has a degree that is exactly one less than a perfect power of two. Once we have that, we have a recursive way to calculate the quantity $s(x)$. With $s(x)$ calculated, for $m \leq 2n$, we obtain $\lfloor \frac{d(x)}{p(x)} \rfloor$ by calculating $d(x)s(x)$ using FFT, dividing it by x^{2n} and taking only the terms with non-negative power of x. If $m > 2n$, we will caculate a polynomial $t(x) = x^{2n} - p(x)s(x)$.

   We can write, $\frac{d(x)}{p(x)} = \frac{d(x)(p(x)s(x)+t(x))}{x^{2n}p(x)}$.

   Therefore, $\lfloor \frac{d(x)}{p(x)} \rfloor = \lfloor \frac{d(x)s(x)}{x^{2n}} \rfloor + \lfloor \frac{\lfloor \frac{d(x)t(x)}{x^{2n}} \rfloor}{p(x)} \rfloor$.

   The first part can be calculated as shown for $m < 2n$ case. That will give us a part of the required $q(x)$. In the otehr term, $\lfloor \frac{d(x)t(x)}{x^{2n}} \rfloor$, we compute $d(x)t(x)$ using FFT, divide it by x^{2n} and take only non-negative powers of x. Then $\lfloor \frac{d(x)t(x)}{x^{2n}} \rfloor$ is to be divided by $p(x)$ for the next iteration using the method as shown before. The calculated quotient in each part is added to the previously existing quotient. Once the quotient $q(x)$ is made, we can find $r(x) = d(x) - p(x)q(x)$ where $p(x)q(x)$ is computed using FFT.

4. An $n \times n$ Toeplitz matrix T is given by:

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \dots & t_{-n+2} \\ t_2 & t_1 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \dots & t_1 & t_0 \end{bmatrix}$$

   Now we know that a circulant matrix of size n can be mutiplied to a vec-

tor using FFT with a runtime od $O(nlog(n))$. To use that to our benefit, we create a circulant matrix $C = \left[\begin{array}{c|c} T & A \\ \hline A & T \end{array}\right]$ where matrix A is given by:

$$A = \begin{bmatrix} 0 & t_{n-1} & t_{n-2} & \dots & t_1 \\ t_{-n+1} & 0 & t_{n-1} & \dots & t_2 \\ t_{-n+2} & t_{-n+1} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & t_{n-1} \\ t_{-1} & t_{-2} & \dots & t_{-n+1} & 0 \end{bmatrix}.$$

For a vector v, we can perform Tv using, $Tv = \begin{bmatrix} 0_n & I_n \end{bmatrix} C \begin{bmatrix} v \\ 0_n \end{bmatrix}$, where matrix product involving C is performed using FFT.