2 External Memory Algorithms

The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on ones desk or by taking an airplane to the other side of the world and using a sharpener on someone elses desk.

(D. Comer)

2.1 Disk Storage

2.2 The External Memory Model

2.3 Back of the Envelope Calculations for External Memory

2.4 Standard Matrix Multiplication

Let X, Y, and Z be three $n \times n$ matrices. We assume that the matrices are stored on disk in row-major order, and we want to compute in Z the product $X \times Y$. Recall that $Z[i][j] = \sum_{k=1}^{n} X[i][k] \cdot Y[k][j]$. The standard code for matrix multiplication can be adapted to the external memory setting as shown in Figure 2.1.

If we assume that disk accesses require $O(1)$ time, the running time of this algorithm is $\Theta(n^3)$. We now analyze the number of I/Os. The number of write operations is n^2, one for each item of the output matrix, while the number of read operations is $2n^3$. Hence, the total number of I/Os is $n^2 + 2n^3 = \Theta(n^3)$.

The number of I/Os can be reduced by exploiting the fact that items can be moved in blocks of size B between main memory and disk. Consider first matrix X. Items of X are laid out on disk in the following order:

$$x_{11}\ x_{12}\ x_{13}\ x_{14} \mid x_{21}\ x_{22}\ x_{23}\ x_{24} \mid x_{31}\ x_{32}\ x_{33}\ x_{34} \mid x_{41}\ x_{42}\ x_{43}\ x_{44}$$

and the algorithm accesses X according to the following pattern:

$$\begin{array}{ll} x_{11}\ x_{12}\ x_{13}\ x_{14} \mid x_{11}\ x_{12}\ x_{13}\ x_{14} \mid x_{11}\ x_{12}\ x_{13}\ x_{14} \mid x_{11}\ x_{12}\ x_{13}\ x_{14} \mid & (i = 1, j \in [1, 4]) \\ x_{21}\ x_{22}\ x_{23}\ x_{24} \mid x_{21}\ x_{22}\ x_{23}\ x_{24} \mid x_{21}\ x_{22}\ x_{23}\ x_{24} \mid x_{21}\ x_{22}\ x_{23}\ x_{24} \mid & (i = 2, j \in [1, 4]) \end{array}$$

```
algorithm standardMatMul(matrix $X$, matrix $Y$, matrix $Z$, int $n$)
    for $i = 1$ to $n$
        for $j = 1$ to $n$
            $z = 0$
            for $k = 1$ to $n$
                load from disk item $X[i][k]$
                load from disk item $Y[k][j]$
                $z = z + X[i][k] \times Y[k][j]$
            write to disk value $z$ as item $Z[i][j]$
```

Figure 2.1: Standard matrix multiplication algorithm adapted to the external memory setting.

$$x_{31}\ x_{32}\ x_{33}\ x_{34} \mid x_{31}\ x_{32}\ x_{33}\ x_{34} \mid x_{31}\ x_{32}\ x_{33}\ x_{34} \mid x_{31}\ x_{32}\ x_{33}\ x_{34} \mid \quad (i = 3, j \in [1,4])$$

$$x_{41}\ x_{42}\ x_{43}\ x_{44} \mid x_{41}\ x_{42}\ x_{43}\ x_{44} \mid x_{41}\ x_{42}\ x_{43}\ x_{44} \mid x_{41}\ x_{42}\ x_{43}\ x_{44} \quad (i = 4, j \in [1,4])$$

Items of X are accessed by row, which is exactly the same order in which they are laid out on disk. Hence, instead of storing only one item per disk access we could use a buffer of size B and store B consecutive items. In this way the algorithm would incur in one I/O every B accesses to matrix X, obtaining n^3/B I/Os to load X throughout the computation. In the example above, if $B = 2$ the first I/O would load $x_{11}\ x_{12}$, the second I/O $x_{13}\ x_{14}$, the third I/O again $x_{11}\ x_{12}$, and so on. Moreover, each item is accessed exactly n times, and its reuse distance (i.e., the number of steps between two consecutive accesses to the same item) is exactly n.

Matrix Z be can treated similarly, resulting in n^2/B write operations. Unfortunately, the same technique cannot be applied to matrix Y, which is accessed by column. Items of Y are also laid out on disk in row-major order:

$$y_{11}\ y_{12}\ y_{13}\ y_{14} \mid y_{21}\ y_{22}\ y_{23}\ y_{24} \mid y_{31}\ y_{32}\ y_{33}\ y_{34} \mid y_{41}\ y_{42}\ y_{43}\ y_{44}$$

and the algorithm accesses y according to the following pattern:

$$y_{11}\ y_{21}\ y_{31}\ y_{41} \mid y_{12}\ y_{22}\ y_{32}\ y_{42} \mid y_{13}\ y_{23}\ y_{33}\ y_{43} \mid y_{14}\ y_{24}\ y_{34}\ y_{44} \mid$$

$$y_{11}\ y_{21}\ y_{31}\ y_{41} \mid y_{12}\ y_{22}\ y_{32}\ y_{42} \mid y_{13}\ y_{23}\ y_{33}\ y_{43} \mid y_{14}\ y_{24}\ y_{34}\ y_{44} \mid$$

$$y_{11}\ y_{21}\ y_{31}\ y_{41} \mid y_{12}\ y_{22}\ y_{32}\ y_{42} \mid y_{13}\ y_{23}\ y_{33}\ y_{43} \mid y_{14}\ y_{24}\ y_{34}\ y_{44} \mid$$

$$y_{11}\ y_{21}\ y_{31}\ y_{41} \mid y_{12}\ y_{22}\ y_{32}\ y_{42} \mid y_{13}\ y_{23}\ y_{33}\ y_{43} \mid y_{14}\ y_{24}\ y_{34}\ y_{44}$$

In this case the first I/O loads $y_{11}\ y_{12}$, but y_{12} goes unused, the second I/O loads $y_{21}\ y_{22}$, but y_{22} goes unused, the third I/O loads $y_{31}\ y_{32}$, but y_{32} goes unused, and so on. Hence, the algorithm has poor spatial locality with respect to matrix Y. Moreover, the reuse distance of any item is equal to n^2, which means that the algorithm has also poor temporal locality. In the worst case where $n > B$, only one of the items loaded from Y in a single I/O operation can be immediately used: the next item is indeed in the successive row, and is not stored in the loaded block when $n > B$. Hence, the algorithm still incurs in n^3 I/Os to load matrix Y.

```
algorithm blockedMatMul(matrix $X$, matrix $Y$, matrix $Z$, int $n$, int $s$)
    for $i = 1$ to $n/s$
        for $j = 1$ to $n/s$
            initialize $Z_{ij}$ to 0
            for $k = 1$ to $n/s$
                load from disk quadrant $X_{ik}$
                load from disk quadrant $Y_{kj}$
                $Z_{ij} = Z_{ij} \boxplus (X_{ik} \boxtimes Y_{kj})$
            write $Z_{ij}$ to disk
```

Figure 2.2: Blocked matrix multiplication algorithm with blocking parameter s. Symbols $\boxplus$ and $\boxtimes$ denote standard addition and multiplication between matrices: any internal memory algorithm can be used to implement $\boxplus$ and $\boxtimes$.

2.5 Blocked Matrix Multiplication

A well-known technique to improve temporal locality is *blocking*, in which data is partitioned into chunks of appropriate size that are completely processed before loading the next chunk. Let s be a blocking parameter (we assume for simplicity that s divides the matrix side n). For any pair of indexes a and b such that $1 \le a, b \le n/s$, we denote with X_{ab} the quadrant of matrix X consisting of elements in rows $(a-1)\cdot s+1, ..., a\cdot s$ and in columns $(b-1)\cdot s+1, ..., b\cdot s$. The blocked implementation of the matrix multiplication algorithm can be written as shown in Figure 2.2.

It is worth noticing that loading a $s \times s$ submatrix X_{ij} requires to access s different rows of matrix X. For each row, items are stored contiguously on disk and can be loaded in chunks of size B with one I/O. Hence, the total number of I/Os per load operation is $\Theta(s + s^2/B)$. Write operations are similar. The entire execution of the blocked matrix multiplication algorithm therefore incurs in

$$\sum_{i=1}^{n/s}\sum_{j=1}^{n/s}\left(\Theta\left(s+\frac{s^2}{B}\right)+\sum_{k=1}^{n/s}\Theta\left(s+\frac{s^2}{B}\right)\right)=\Theta\left(\frac{n^3}{s^3}\left(s+\frac{s^2}{B}\right)\right)=\Theta\left(\frac{n^3}{s^2}+\frac{n^3}{sB}\right)$$

Choosing $s = \sqrt{M}/3$, three entire submatrices fit in main memory and the number of I/Os is

$$\Theta\left(\frac{n^3}{M}+\frac{n^3}{B\sqrt{M}}\right)=\Theta\left(\frac{n^3}{B\sqrt{M}}\right)$$

when $M \ge B^2$. This number is asymptotically optimal, since matches a lower bound proved by Hong and Kung (ACM Symposium on Theory of Computing, 1981).