| **6.895 Theory of Parallel Systems** | Lecture 20 |
|---|---|

# Permuting Data on Parallel Disks

*Lecturer: Charles Leiserson* *Scribe: Sajindra Jayasena, Sharad Ganesh*

## Lecture Summary

1. *Parallel Disk Systems*
   In this section, an overview of parallel Disk systems is given with a brief look at practical examples
2. *Problem of permuting Disk Blocks*
   In this section,we investigate how a disk access permutation is performed which would materialize high level of parallelism
3. *Conflict Graphs*
   This section looks at the theoretical aspects on how a non-conflicting schedule could be created for parallel disk access
4. *Hall's Theorem*
   Here we look at Hall's theorem and its application to a non blocking scheduling

## 1 Parallel Disk Systems

In this section we will briefly look at parallel disk systems.

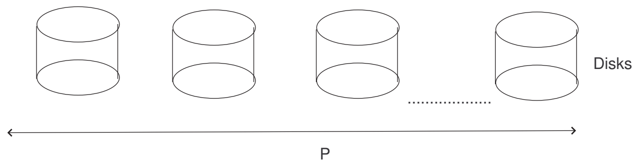


**Figure 1:** Parallel disks

One class of parallel disk systems that's used widely is RAID, which stands for **R**edundant **A**rray of **I**nexpensive **D**isks. In RAID, error correction is done across the disks and multiple disks can be accessed concurrently. RAID defines several levels of configurations that provide a combination of properties like level of performance,fault tolerance,redundancy etc that the user can select depending on his requirement.
Some example of levels are:

1. RAID 0: Striping - The controller stripes data across multiple drives in the RAID subsystem. RAID 0 has the best performance and capacity of any RAID level, but the lowest availability (no fault tolerance).
2. RAID 1: Mirroring - In RAID 1, the controller duplicates all data from one drive to another. RAID 1 provides complete data redundancy, but at the cost of doubling the required data storage capacity. RAID 1 has the highest availability but lowest capacity, since twice the number of drives are required.

3. RAID 3: Striping with Dedicated Parity - RAID level 3 provides disk striping and complete data redundancy through a dedicated parity drive. The stripe size must be 64KB if RAID 3 is used. RAID 3 handles data at the block level, not the byte level, so it is ideal for networks that often handle very large files, such as graphic images.

4. RAID 5: Striping with Rotational Parity - The controller stripes blocks of data and parity information across all drives. The parity block is distributed among the drives of the array, giving a more balanced access load across the drives. The parity information is used to recover the data if one drive fails, and explains why this method is the most popular.

Another type of configuration is JBOD which stands for **J**ust a **B**unch **O**f **D**isks.This is just a collection of disks without any high level inter disk management system.
**Current disk timings :** Generally following are the current timing for data access and data transfer in modern disk systems.

1. Data access times: $\simeq 10^{-2}$ seconds

2. Data Transfer Rate $\simeq 10^{8}$ words per seconds.

it is clear that disk access takes most of the time compared to disk transfer.Therefore the main strategy is to **reduce or parallelize the disk accesses as much as possible**.

One convenient engineering assumption is:
Instead of having free writing across the disk , the disk is broken down into large fixed size blocks. For example each block might consist of 1000+ words or possibly million words.This is to amortize the cost of data access over a larger number of words.Therfore it would make the cost per word relatively lesser.
Now we will look at an example and visualize how it is modelled in a disk.

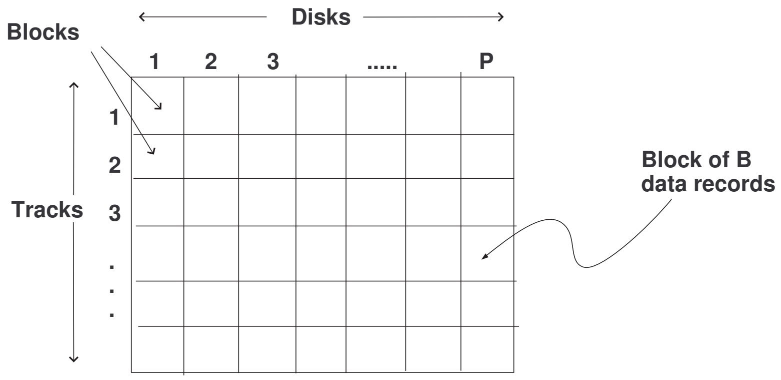


**Figure 2:** Disk block layout

In Figure 2, there are P disks and N data records.
Therefore to store N data records, the number of tracks needed are :
**Number of Tracks = N / P.B**
This is also equal to the number of IOs' to read all the data in N records.This is true because in parallel disk systems , at each unit time one block per each disk can be read. For example:
At Time $T_1$ : read track 1
At Time $T_2$ : read track 2
.......
At Time $T_n$ : read track n

This simplifies the model to a Block read , without the need to look into the intricacies of each block. Typically a computer has a certain size of memory. For example it might be able to hold M data records where the general assumption is that M $\gg$ P**.**B . ( i.e in reality the size of memory is significantly large compared to the amount of data that could be read at unit time from P parallel disks )

## 2 Problem of permuting Disk Blocks

Now we will look at how an actual disk access works and how it should be scheduled in such a manner in obtaining maximum parallelism in disk access.
Here we will look at a permutation which is :

1. Offline ( Fixed in advance )
2. Not In Place

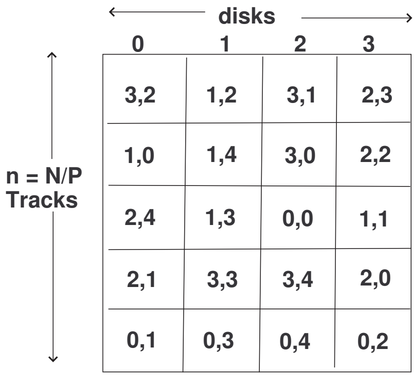


**Figure 3:** Disk access permutation

Here in figure 3, each block would hold 1 record. Therefore the number of tracks required to hold N data records is **N/P** where P is the number of Disks in the parallel system. But when there are multiple data records per block the calculation would be slightly more complex.
In the diagram ,each block specifies where it should be written to .For example the top left block specifies that the data in that block should be written to the second track of third Disk.
**Objective of scheduling :**
The objective of selecting a proper access sequence for the given permutation is to be able to direct the set of disks to read a set of disk blocks in a column and write those disk blocks in such a manner that the disk blocks that are read would allow the disks to write them back in parallel , thus achieving the optimal parallelism.
For example as in the diagram, if the last track is read in parallel, when it comes to writing them , they all have to be written to a single column.But the issue here is they can only be written one at a time ,since all those blocks belong to same disk.Therefore the objective is to find an access sequence where a set of blocks could be read at the one unit of time in parallel and could be written at in one unit of time in parallel. i.e in fewer IOs' as possible.
Now let's look at the following schedule.For simplicity lets assume that the read is performed in a linear fashion to a file and would be written back to a different location in the disks.

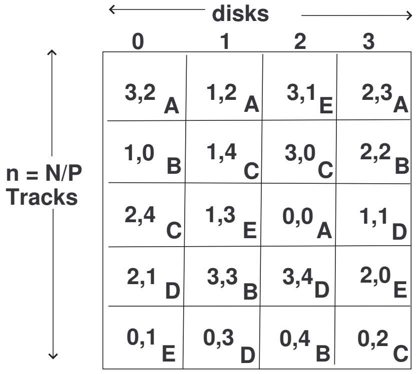


**Figure 4:** Disk block access sequence

By simulating through the schedule it is evident that reads and writes are scheduled in a fashion that all writes can also be performed in parallel. The permutation is done in exactly **N/P** parallel IOs'.
How well could we perform this in general? The following section presents the theorem for it.

**Theorem 1** *It should be possible to permute access of N data records with P disks with* ***O(N/P)*** *parallel IOs.*
*No matter what permutation is given this can be scheduled to guarantee that it could be performed in N/P parallel Disk IOs'.*
*note : The**O(N/P)** is given since it needs to perform a read as well as a write.*

Now let's look at how we could model and analyze this using Graph theory .

# 3 Conflict graph

**Idea:**

Set up a source disk and a destination disk. There is a vertex for every disk. The graph models the permutations required and helps us see what the conflicts are.

The degree of this graph is $n = N/P$. The smallest degree of any vertex is also $n = N/P$.

In fact all degrees are the same n = N/P.
This is a bi-partite graph which is n-regular, since all degrees are the same(=$n$).

**Claim:**

Any d-regular bi-partite multigraph can be edge colored with $d$ colors.

**Definition 2 (Edge coloring)** *Color the edges in such a way that all the edges coming into a given vertex have a different color.*

**Claim:** Each vertex will require at least d-colors and we argue that it is sufficient.

In general you can show that if $d$ is the degree of any arbitrary graph it can be colored with at most

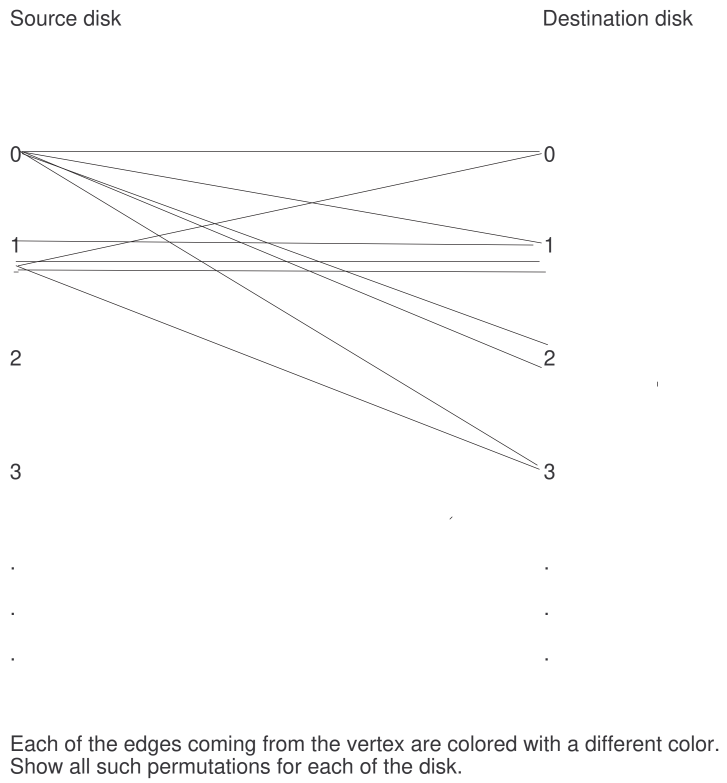


**Figure 5:** Conflict Graph

$d + 1$ colors. The fact that, whether you can color it with d-colors is NP-Complete. When the graph is colored, each coloring matches the vertices on the left with the vertices on the right.

```
Color = step at which the block is moved.
```

**Method of edge coloring:**

```
Find a perfect matching(everything on the left is matched to
something on the right).

Color the edges in the matching using color 1.

Remove all edges of color 1. (Now we have a $d-1$ regular
bi-partite multigraph. The degree of every vertex has dropped by
exactly 1).

Recur.
```

To know that a perfect matching exists in a d-regular bi-partite graph, we present Hall's theorem.

## 4 Hall's theorem -(General bi-partite graph)

**Theorem 3** *Claim: If it is a d-regular bi-partite graph, a perfect matching always exists.*

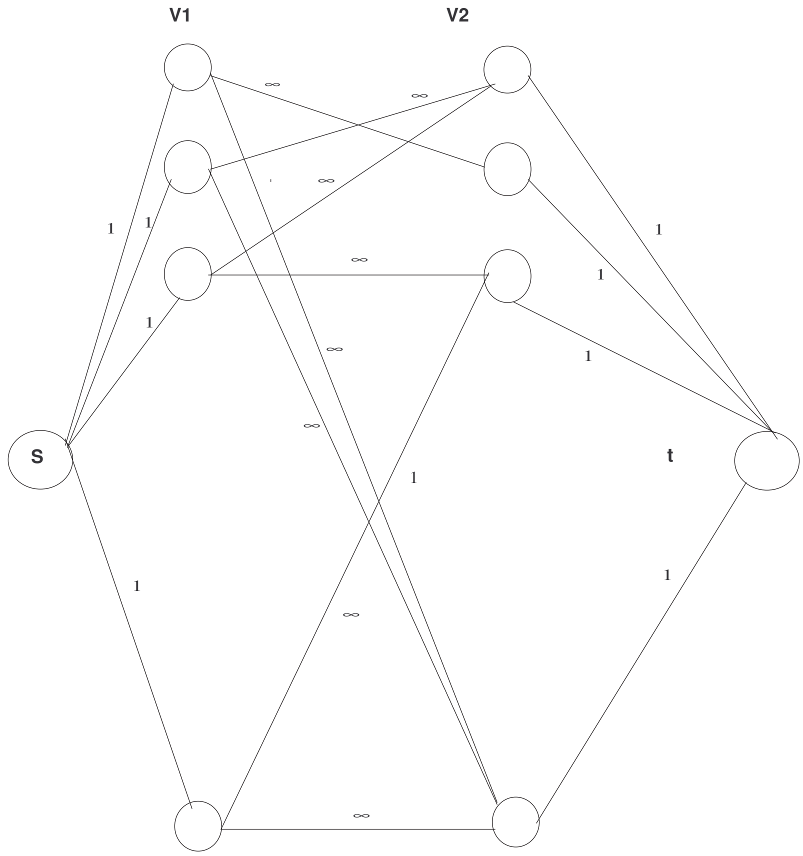


**Figure 6:** Flow graph

For any particular bi-partite graph, we have,

A set of vertices $A \subseteq V_1$, and let $N(A) \subseteq V_2$ be the set of neighbors of A. For any bi-partite graph a perfect matching exists if:

$$|N(A)| \geq |A| \forall A \subseteq V_1.$$

**Proof of Hall's theorem**

To find a maximum matching, one way to do that is to build a flow network shown in Figure: 6. If the value of the flow is $n$, then we have a perfect matching.
**Proof**
Let f be the max flow.

$$|f| = C(S, T)$$

For some cut (S, T) by the max-flow min-cut theorem. S has to be in one set and T in another, according to the max-flow min-cut theorem. If you have a maximum flow, there must be a cut, whose capacity is equal to the cut. If there is none, then you can find another path to route the flow.

Let, $A = S \cap V_1$, since the edges from vertices in $V_1$ to vertices in $V_2$ have infinite capacity.

$$N(A) \subseteq S$$

The max possible flow on this network is n. Flow across any cut in the graph is n. At the same time, the minimum cut must have the value of the flow, and cannot be infinite.
Also,

$$N(V_1 - A) \subseteq T$$

$$C(S,T) \geq |V_1 - A| + |N(A)|$$

$$|V_1 - A| + |A|$$

$$= |V_1| = n$$

□

The flow value is $n$, everyone gets matched up, which proves Hall's theorem.