CSE 548: Analysis of Algorithms

Lecture 31
(Analyzing I/O and Cache Performance)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2015
Iterative Matrix-Multiply Variants

double Z[n][n], X[n][n], Y[n][n];

$I-J-K$

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            Z[i][j] += X[i][k] * Y[k][j];

$I-K-J$

for (int i = 0; i < n; i++)
    for (int k = 0; k < n; k++)
        for (int j = 0; j < n; j++)
            Z[i][j] += X[i][k] * Y[k][j];

$J-I-K$

for (int j = 0; j < n; j++)
    for (int i = 0; i < n; i++)
        for (int k = 0; k < n; k++)
            Z[i][j] += X[i][k] * Y[k][j];

$J-K-I$

for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            Z[i][j] += X[i][k] * Y[k][j];

$K-I-J$

for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            Z[i][j] += X[i][k] * Y[k][j];

$K-J-I$

for (int k = 0; k < n; k++)
    for (int j = 0; j < n; j++)
        for (int i = 0; i < n; i++)
            Z[i][j] += X[i][k] * Y[k][j];
**Performance of Iterative Matrix-Multiply Variants**

**Processor:** 2.7 GHz Intel Xeon E5-2680 (used only one core)

**Caches & RAM:** private 32KB L1, private 256KB L2, shared 20MB L3, 32 GB RAM

**Optimizations:** none (icc 13.0 with –O0)

\[ n = 1000 \]

\[ n = 2000 \]

\[ n = 3000 \]
For efficient computation we need

- fast processors
- fast and large (but not so expensive) memory

But memory cannot be cheap, large and fast at the same time, because of

- finite signal speed
- lack of space to put enough connecting wires

A reasonable compromise is to use a memory hierarchy.
A memory hierarchy is

- almost as fast as its fastest level
- almost as large as its largest level
- inexpensive
To perform well on a memory hierarchy algorithms must have **high locality** in their memory access patterns.
Locality of Reference

**Spatial Locality:** When a block of data is brought into the cache it should contain as much useful data as possible.

**Temporal Locality:** Once a data point is in the cache as much useful work as possible should be done on it before evicting it from the cache.
CPU-bound vs. Memory-bound Algorithms

The Op-Space Ratio: Ratio of the number of operations performed by an algorithm to the amount of space (input + output) it uses.

Intuitively, this gives an upper bound on the average number of operations performed for every memory location accessed.

CPU-bound Algorithm:
- high op-space ratio
- more time spent in computing than transferring data
- a faster CPU results in a faster running time

Memory-bound Algorithm:
- low op-space ratio
- more time spent in transferring data than computing
- a faster memory system leads to a faster running time
The two-level I/O model [Aggarwal & Vitter, CACM’88] consists of:
- an *internal memory* of size $M$
- an arbitrarily large *external memory* partitioned into blocks of size $B$.

**I/O complexity** of an algorithm

= number of blocks transferred between these two levels

Basic I/O complexities: $\text{scan}(N) = \Theta \left( \frac{N}{B} \right)$ and $\text{sort}(N) = \Theta \left( \frac{N}{B} \log_M \frac{N}{B} \right)$

Algorithms often crucially depend on the knowledge of $M$ and $B$

$\Rightarrow$ algorithms do not adapt well when $M$ or $B$ changes
The ideal-cache model [ Frigo et al., FOCS’99 ] is an extension of the I/O model with the following constraint:

- algorithms are not allowed to use knowledge of $M$ and $B$.

Consequences of this extension

- algorithms can simultaneously adapt to all levels of a multi-level memory hierarchy
- algorithms become more flexible and portable

Algorithms for this model are known as cache-oblivious algorithms.
The model makes the following assumptions:

- Optimal offline cache replacement policy
- Exactly two levels of memory
- Automatic replacement & full associativity
The Ideal-Cache Model: Assumptions

The model makes the following assumptions:

- Optimal offline cache replacement policy
  - LRU & FIFO allow for a constant factor approximation of optimal [Sleator & Tarjan, JACM’85]
- Exactly two levels of memory
- Automatic replacement & full associativity
The Ideal-Cache Model: Assumptions

The model makes the following assumptions:

- Optimal offline cache replacement policy
- Exactly two levels of memory
  - can be effectively removed by making several reasonable assumptions about the memory hierarchy [Frigo et al., FOCS’99]
- Automatic replacement & full associativity
The model makes the following assumptions:

- Optimal offline cache replacement policy
- Exactly two levels of memory
- Automatic replacement & full associativity
  - in practice, cache replacement is automatic (by OS or hardware)
  - fully associative LRU caches can be simulated in software with only a constant factor loss in expected performance [Frigo et al., FOCS’99]
The model makes the following assumptions:

- Optimal offline cache replacement policy
- Exactly two levels of memory
- Automatic replacement & full associativity

Often makes the following assumption, too:

- $M = \Omega(B^2)$, i.e., the cache is *tall*
The Ideal-Cache Model: Assumptions

The model makes the following assumptions:

- Optimal offline cache replacement policy
- Exactly two levels of memory
- Automatic replacement & full associativity

Often makes the following assumption, too:

- \( M = \Omega(B^2) \), i.e., the cache is \textit{tall}
  - most practical caches are tall
Cache-oblivious vs. cache-aware bounds:

- Basic I/O bounds (same as the cache-aware bounds):
  \[
  \text{scan}(N) = \Theta\left(\frac{N}{B}\right)
  \]
  \[
  \text{sort}(N) = \Theta\left(\frac{N}{B} \log_M \frac{N}{B}\right)
  \]

- Most cache-oblivious results match the I/O bounds of their cache-aware counterparts

- There are few exceptions; e.g., no cache-oblivious solution to the *permutation* problem can match cache-aware I/O bounds [Brodal & Fagerberg, STOC’03]
<table>
<thead>
<tr>
<th>Problem</th>
<th>Cache-Aware Results</th>
<th>Cache-Oblivious Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Scanning ((\text{scan}(N)))</td>
<td>(O\left(\frac{N}{B}\right))</td>
<td>(O\left(\frac{N}{B}\right))</td>
</tr>
<tr>
<td>Sorting ((\text{sort}(N)))</td>
<td>(O\left(\frac{N \log M}{B} \frac{N}{B}\right))</td>
<td>(O\left(\frac{N \log M}{B} \frac{N}{B}\right))</td>
</tr>
<tr>
<td>Selection</td>
<td>(O\left(\text{scan}(N)\right))</td>
<td>(O\left(\text{scan}(N)\right))</td>
</tr>
<tr>
<td>B-Trees [Am] (\text{Insert, Delete})</td>
<td>(O\left(\log B \frac{N}{B}\right))</td>
<td>(O\left(\log B \frac{N}{B}\right))</td>
</tr>
<tr>
<td>Priority Queue [Am] (\text{Insert, Weak Delete, Delete-Min})</td>
<td>(O\left(\frac{1}{B} \log M \frac{N}{B}\right))</td>
<td>(O\left(\frac{1}{B} \log M \frac{N}{B}\right))</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>(O\left(\frac{N^3}{B\sqrt{M}}\right))</td>
<td>(O\left(\frac{N^3}{B\sqrt{M}}\right))</td>
</tr>
<tr>
<td>Sequence Alignment</td>
<td>(O\left(\frac{N^2}{BM}\right))</td>
<td>(O\left(\frac{N^2}{BM}\right))</td>
</tr>
<tr>
<td>Single Source Shortest Paths</td>
<td>(O\left(\left(V + \frac{E}{B}\right) \cdot \log_2 \frac{V}{B}\right))</td>
<td>(O\left(\left(V + \frac{E}{B}\right) \cdot \log_2 \frac{V}{B}\right))</td>
</tr>
<tr>
<td>Minimum Spanning Forest</td>
<td>(O\left(\min\left(\text{sort}(E) \log_2 \log_2 V, V + \text{sort}(E)\right)\right))</td>
<td>(O\left(\min\left(\text{sort}(E) \log_2 \log_2 \frac{VB}{E}, V + \text{sort}(E)\right)\right))</td>
</tr>
</tbody>
</table>

Table 1: \(N = \#\text{elements}, V = \#\text{vertices}, E = \#\text{edges}, \text{Am} = \text{Amortized.}\)
Matrix Multiplication
Iterative Matrix Multiplication

\[ z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj} \]

Iter-MM( X, Y, Z, n )

1. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
2. \( \text{for } j \leftarrow 1 \text{ to } n \text{ do} \)
3. \( \text{for } k \leftarrow 1 \text{ to } n \text{ do} \)
4. \( z_{ij} \leftarrow z_{ij} + x_{ik} \times y_{kj} \)
Iterative Matrix Multiplication

Iter-MM( X, Y, Z, n )
1. for i ← 1 to n do
2. for j ← 1 to n do
3. for k ← 1 to n do
4. \[ z_{ij} \leftarrow z_{ij} + x_{ik} \times y_{kj} \]

Each iteration of the \textit{for} loop in line 3 incurs \( O(n) \) cache misses.

I/O-complexity of \textit{Iter-MM}, \( Q(n) = O(n^3) \)
Iterative Matrix Multiplication

Iter-MM( X, Y, Z, n )

1. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
2. \( \text{for } j \leftarrow 1 \text{ to } n \text{ do} \)
3. \( \text{for } k \leftarrow 1 \text{ to } n \text{ do} \)
4. \( z_{ij} \leftarrow z_{ij} + x_{ik} \times y_{kj} \)

Each iteration of the \textbf{for} loop in line 3 incurs \( O \left( 1 + \frac{n}{B} \right) \) cache misses.

I/O-complexity of \textit{Iter-MM}, \( Q(n) = O \left( n^2 \left( 1 + \frac{n}{B} \right) \right) = O \left( \frac{n^3}{B} + n^2 \right) \)
Block Matrix Multiplication

\[ \text{Block-MM}(X, Y, Z, n) \]

1. \( \text{for } i \leftarrow 1 \ \text{to } n / m \ \text{do} \)
2. \( \text{for } j \leftarrow 1 \ \text{to } n / m \ \text{do} \)
3. \( \text{for } k \leftarrow 1 \ \text{to } n / m \ \text{do} \)
4. \( \text{Iter-MM}(X_{ik}, Y_{kj}, Z_{ij}) \)
Choose \( m = \sqrt{M/3} \), so that \( X_{ik}, Y_{kj} \) and \( Z_{ij} \) just fit into the cache.

Then line 4 incurs \( \Theta \left( m \left( 1 + \frac{m}{B} \right) \right) \) cache misses.

I/O-complexity of \( \text{Block-MM} \) [assuming a \textit{tall cache}, i.e., \( M = \Omega(B^2) \)]

\[
= \Theta \left( \left( \frac{n}{m} \right)^3 \left( m + \frac{m^2}{B} \right) \right) = \Theta \left( \frac{n^3}{m^2} + \frac{n^3}{Bm} \right) = \Theta \left( \frac{n^3}{M} + \frac{n^3}{B\sqrt{M}} \right) = \Theta \left( \frac{n^3}{B\sqrt{M}} \right)
\]

( Optimal: Hong & Kung, STOC’81 )
Block Matrix Multiplication

Choose \( m = \sqrt{M/2} \), so that \( X, Y, \) and \( Z \) just fit into the cache.

Optimal for any algorithm that performs the operations given by the following definition of matrix multiplication:

\[
Z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}
\]

\[
\Theta \left( \left( \frac{n}{m} \right)^3 \left( m + \frac{m^2}{B} \right) \right) = \Theta \left( \frac{n}{m^2} + \frac{n}{Bm} \right) = \Theta \left( \frac{n^3}{M} + \frac{n^3}{B \sqrt{M}} \right) = \Theta \left( \frac{n^3}{B \sqrt{M}} \right)
\]

(Optimal: Hong & Kung, STOC’81)
Block-MM\((X, Y, Z, n)\)

1. \(\text{for } i \leftarrow 1 \text{ to } n / s \text{ do}\)
2. \(\text{for } j \leftarrow 1 \text{ to } n / s \text{ do}\)
3. \(\text{for } k \leftarrow 1 \text{ to } n / s \text{ do}\)
4. \(\text{Iter-MM}(X_{ik}, Y_{kj}, Z_{ij}, s)\)
Multiple Levels of Cache

\[
\text{Block-MM}(X, Y, Z, n) \\
1. \text{for } i_1 \leftarrow 1 \text{ to } n/s \text{ do} \\
2. \quad \text{for } j_1 \leftarrow 1 \text{ to } n/s \text{ do} \\
3. \quad \quad \text{for } k_1 \leftarrow 1 \text{ to } n/s \text{ do} \\
4. \quad \quad \quad \text{for } i_2 \leftarrow 1 \text{ to } s/t \text{ do} \\
5. \quad \quad \quad \quad \text{for } j_2 \leftarrow 1 \text{ to } s/t \text{ do} \\
6. \quad \quad \quad \quad \quad \text{for } k_2 \leftarrow 1 \text{ to } s/t \text{ do} \\
7. \quad \quad \quad \quad \quad \text{Iter-MM}(X_{i_1k_1}i_2k_2, Y_{k_1j_1}k_2j_2, X_{i_1j_1}i_2j_2, t)
\]
Multiple Levels of Cache

Block-MM( X, Y, Z, n )

1. for $i_1 \leftarrow 1$ to $n / s$ do
2. for $j_1 \leftarrow 1$ to $n / s$ do
3. for $k_1 \leftarrow 1$ to $n / s$ do
4. for $i_2 \leftarrow 1$ to $s / t$ do
5. for $j_2 \leftarrow 1$ to $s / t$ do
6. for $k_2 \leftarrow 1$ to $s / t$ do
7. Iter-MM( $(X_{i_1 k_1})_{i_2 k_2}$, $(Y_{k_1 j_1})_{k_2 j_2}$, $(X_{i_1 j_1})_{i_2 j_2}$, $t$ )
Recursive Matrix Multiplication

\[
\begin{bmatrix}
  Z_{11} & Z_{12} \\
  Z_{21} & Z_{22}
\end{bmatrix}
\begin{bmatrix}
  X_{11} & X_{12} \\
  X_{21} & X_{22}
\end{bmatrix}
\begin{bmatrix}
  Y_{11} & Y_{12} \\
  Y_{21} & Y_{22}
\end{bmatrix}
= 
\begin{bmatrix}
  X_{11} Y_{11} + X_{12} Y_{21} & X_{11} Y_{12} + X_{12} Y_{22} \\
  X_{21} Y_{11} + X_{22} Y_{21} & X_{21} Y_{12} + X_{22} Y_{22}
\end{bmatrix}
\]
**Recursive Matrix Multiplication**

**Rec-MM** $(Z, X, Y)$

1. *if* $Z$ is a $1 \times 1$ matrix *then* $Z \leftarrow Z + X \cdot Y$

2. *else*

3. **Rec-MM** $(Z_{11}, X_{11}, Y_{11})$, **Rec-MM** $(Z_{11}, X_{12}, Y_{21})$

4. **Rec-MM** $(Z_{12}, X_{12}, Y_{12})$, **Rec-MM** $(Z_{12}, X_{12}, Y_{22})$

5. **Rec-MM** $(Z_{21}, X_{21}, Y_{11})$, **Rec-MM** $(Z_{21}, X_{22}, Y_{21})$

6. **Rec-MM** $(Z_{22}, X_{21}, Y_{12})$, **Rec-MM** $(Z_{22}, X_{22}, Y_{22})$
Recursive Matrix Multiplication

\[ \text{Rec-MM( } Z, X, Y \text{ )} \]

1. \textit{if } \( Z \equiv 1 \times 1 \text{ matrix } \text{then } Z \leftarrow Z + X \cdot Y \)
2. \textit{else}
3. \text{Rec-MM( } Z_{11}, X_{11}, Y_{11} \text{ ), Rec-MM( } Z_{11}, X_{12}, Y_{21} \text{ )}
4. \text{Rec-MM( } Z_{12}, X_{12}, Y_{12} \text{ ), Rec-MM( } Z_{12}, X_{12}, Y_{22} \text{ )}
5. \text{Rec-MM( } Z_{21}, X_{21}, Y_{11} \text{ ), Rec-MM( } Z_{21}, X_{22}, Y_{21} \text{ )}
6. \text{Rec-MM( } Z_{22}, X_{21}, Y_{12} \text{ ), Rec-MM( } Z_{22}, X_{22}, Y_{22} \text{ )}

I/O-complexity (for all \( n > M \)), \( Q(n) = \begin{cases} 
0 \left( n + \frac{n^2}{B} \right), & \text{if } n^2 \leq \alpha M \\
8Q \left( \frac{n}{2} \right) + O(1), & \text{otherwise}
\end{cases} \)

\[ = O \left( \frac{n^3}{M} + \frac{n^3}{B \sqrt{M}} \right) = O \left( \frac{n^3}{B \sqrt{M}} \right), \text{when } M = \Omega(B^2) \]

I/O-complexity (for all \( n \)) \( = O \left( \frac{n^3}{B \sqrt{M}} + \frac{n^2}{B} + 1 \right) \) (why?)
Recursive Matrix Multiplication with Z-Morton Layout
Recursive Matrix Multiplication with Z-Morton Layout

\[ Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \]
**Recursive Matrix Multiplication with Z-Morton Layout**

\[
\begin{array}{cccc}
Z_{1111} & Z_{1112} & Z_{1211} & Z_{1212} \\
Z_{1121} & Z_{1122} & Z_{1221} & Z_{1222} \\
Z_{2111} & Z_{2112} & Z_{2211} & Z_{2212} \\
Z_{2121} & Z_{2122} & Z_{2221} & Z_{2222} \\
\end{array}
\]

\[
\begin{array}{cccc}
Z_{1111} & Z_{1112} & Z_{1121} & Z_{1122} \\
Z_{1211} & Z_{1212} & Z_{1221} & Z_{1222} \\
Z_{2111} & Z_{2112} & Z_{2121} & Z_{2122} \\
Z_{2211} & Z_{2212} & Z_{2221} & Z_{2222} \\
\end{array}
\]

\[
\begin{array}{cccc}
Z_{11} & Z_{12} & Z_{21} & Z_{22} \\
\end{array}
\]
Recursive Matrix Multiplication with Z-Morton Layout

Source: wikipedia
Recursive Matrix Multiplication with Z-Morton Layout

Rec-MM (Z, X, Y)

1. if $Z \equiv 1 \times 1$ matrix then $Z \leftarrow Z + X \cdot Y$
2. else
3. Rec-MM (Z₁₁, X₁₁, Y₁₁), Rec-MM (Z₁₁, X₁₂, Y₂₁)
4. Rec-MM (Z₁₂, X₁₂, Y₁₂), Rec-MM (Z₁₂, X₁₂, Y₂₂)
5. Rec-MM (Z₂₁, X₂₁, Y₁₁), Rec-MM (Z₂₁, X₂₂, Y₂₁)
6. Rec-MM (Z₂₂, X₂₁, Y₁₂), Rec-MM (Z₂₂, X₂₂, Y₂₂)

I/O-complexity (for $n > M$), $Q(n) = \begin{cases} 0 \left(1 + \frac{n^2}{B}\right), & \text{if } n^2 \leq \alpha M \\ 8Q\left(\frac{n}{2}\right) + O(1), & \text{otherwise} \end{cases}$

$= O\left(\frac{n^3}{M\sqrt{M}} + \frac{n^3}{B\sqrt{M}}\right) = O\left(\frac{n^3}{B\sqrt{M}}\right)$, when $M = \Omega(B)$

I/O-complexity (for all $n$) $= O\left(\frac{n^3}{B\sqrt{M}} + \frac{n^2}{B} + 1\right)$
### Recursive Matrix Multiplication with Z-Morton Layout

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000000</td>
<td>000001</td>
<td>000100</td>
<td>000101</td>
<td>010000</td>
<td>010001</td>
<td>010100</td>
<td>010101</td>
</tr>
<tr>
<td>001</td>
<td>000010</td>
<td>000011</td>
<td>000110</td>
<td>000111</td>
<td>010100</td>
<td>010101</td>
<td>011010</td>
<td>011011</td>
</tr>
<tr>
<td>010</td>
<td>001000</td>
<td>001001</td>
<td>001100</td>
<td>001101</td>
<td>011000</td>
<td>011001</td>
<td>011100</td>
<td>011101</td>
</tr>
<tr>
<td>011</td>
<td>001010</td>
<td>001011</td>
<td>001110</td>
<td>001111</td>
<td>011010</td>
<td>011011</td>
<td>011110</td>
<td>011111</td>
</tr>
<tr>
<td>100</td>
<td>100000</td>
<td>100001</td>
<td>100100</td>
<td>100101</td>
<td>110000</td>
<td>110001</td>
<td>110100</td>
<td>110101</td>
</tr>
<tr>
<td>101</td>
<td>100010</td>
<td>100011</td>
<td>100110</td>
<td>100111</td>
<td>110100</td>
<td>110101</td>
<td>111010</td>
<td>111011</td>
</tr>
<tr>
<td>110</td>
<td>101000</td>
<td>101001</td>
<td>101100</td>
<td>101101</td>
<td>111000</td>
<td>111001</td>
<td>111100</td>
<td>111101</td>
</tr>
<tr>
<td>111</td>
<td>101010</td>
<td>101011</td>
<td>101110</td>
<td>101111</td>
<td>111100</td>
<td>111101</td>
<td>111110</td>
<td>111111</td>
</tr>
</tbody>
</table>

*Source: wikipedia*
Searching
( Static B-Trees )
A perfectly balanced binary search tree

- Static: no insertions or deletions
- Height of the tree, $h = \Theta(\log_2 n)$

$h = \Theta(\log_2 n)$
- A perfectly balanced binary search tree
- Static: no insertions or deletions
- Height of the tree, $h = \Theta(\log_2 n)$
- A search path visits $O(h)$ nodes, and incurs $O(h) = O(\log_2 n)$ I/Os
I/O-Efficient Static B-Trees

Each node stores $B$ keys, and has degree $B + 1$

Height of the tree, $h = \Theta(\log_B n)$
Each node stores $B$ keys, and has degree $B + 1$

Height of the tree, $h = \Theta(\log_B n)$

A search path visits $O(h)$ nodes, and incurs $O(h) = O(\log_B n)$ I/Os
Cache-Oblivious Static B-Trees?
van Emde Boas Layout

a binary search tree
If the tree contains $n$ nodes, each subtree contains $\Theta(2^{h/2}) = \Theta(\sqrt{n})$ nodes, and $k = \Theta(\sqrt{n})$. 
van Emde Boas Layout

Recursive Subdivision

If the tree contains $n$ nodes, each subtree contains $\Theta(2^{h/2}) = \Theta(\sqrt{n})$ nodes, and $k = \Theta(\sqrt{n})$. 
van Emde Boas Layout

If the tree contains $n$ nodes, each subtree contains $\Theta(2^{h/2}) = \Theta(\sqrt{n})$ nodes, and $k = \Theta(\sqrt{n})$. 
If the tree contains $n$ nodes, each subtree contains $\Theta(2^{h/2}) = \Theta(\sqrt{n})$ nodes, and $k = \Theta(\sqrt{n})$. 

Here is a diagram illustrating the van Emde Boas Layout:

![Diagram of van Emde Boas Layout](image)

- **Recursive Subdivision**
- **A**
- **B_1**
- **B_2**
- **B_k**

The layout is recursive, with each subtree containing a binary search tree. The height $h$ of the tree is divided into $k$ subtrees, each of height $\lfloor h/2 \rfloor$. The layout is designed to optimize storage and retrieval operations.
If the tree contains \( n \) nodes, each subtree contains \( \Theta\left(2^{h/2}\right) = \Theta(\sqrt{n}) \) nodes, and \( k = \Theta(\sqrt{n}) \).
I/O-Complexity of a Search

- The height of the tree is $\log n$.
- Each $\triangle$ has height between $\frac{1}{2} \log B$ & $\log B$.
- Each $\triangle$ spans at most 2 blocks of size $B$. 
The height of the tree is $\log n$.

Each triangle has height between $\frac{1}{2}\log B$ & $\log B$.

Each triangle spans at most 2 blocks of size $B$.

$p = \text{number of triangles visited by a search path}$

Then $p \geq \frac{\log n}{\log B} = \log_B n$, and $p \leq \frac{\log n}{\frac{1}{2}\log B} = 2\log_B n$.

The number of blocks transferred is $\leq 2 \times 2 \log_B n = 4\log_B n$.
Sorting
( Mergesort )
**Merge Sort**

\[ \text{Merge-Sort} ( A, p, r ) \quad \{ \text{sort the elements in } A[ p \ldots r ] \} \]

1. *if* \( p < r \) *then*
2. \( q \leftarrow \lfloor ( p + r ) / 2 \rfloor \)
3. \( \text{Merge-Sort} ( A, p, q ) \)
4. \( \text{Merge-Sort} ( A, q + 1, r ) \)
5. \( \text{Merge} ( A, p, q, r ) \)
Merging $k$ Sorted Sequences

- $k \geq 2$ sorted sequences $S_1, S_2, \ldots, S_k$ stored in external memory
- $|S_i| = n_i$ for $1 \leq i \leq k$
- $n = n_1 + n_2 + \cdots + n_k$ is the length of the merged sequence $S$
- $S$ (initially empty) will be stored in external memory
- Cache must be large enough to store
  - one block from each $S_i$
  - one block from $S$

Thus $M \geq (k + 1)B$
Merging k Sorted Sequences

- Let $B_i$ be the cache block associated with $S_i$, and let $B$ be the block associated with $S$ (initially all empty).
- Whenever a $B_i$ is empty fill it up with the next block from $S_i$.
- Keep transferring the next smallest element among all $B_i$s to $B$.
- Whenever $B$ becomes full, empty it by appending it to $S$.
- In the Ideal Cache Model the block emptying and replacements will happen automatically $\Rightarrow$ cache-obliviuous merging.

I/O Complexity

- Reading $S_i$: #block transfers $\leq 2 + \frac{n_i}{B}$.
- Writing $S$: #block transfers $\leq 1 + \frac{n}{B}$.
- Total #block transfers $\leq 1 + \frac{n}{B} + \sum_{1 \leq i \leq k} \left( 2 + \frac{n_i}{B} \right) = O \left( k + \frac{n}{B} \right)$.
Cache-Oblivious 2-Way Merge Sort

```
Merge-Sort ( A, p, r )         \{ sort the elements in A[ p .. r ] \}

1. if p < r then
2.    q ← ⌊ ( p + r ) / 2 ⌋
3.    Merge-Sort ( A, p, q )
4.    Merge-Sort ( A, q + 1, r )
5.    Merge ( A, p, q, r )
```

I/O Complexity: \[ Q(n) = \begin{cases} O \left( 1 + \frac{n}{B} \right), & \text{if } n \leq M, \\ 2Q \left( \frac{n}{2} \right) + O \left( 1 + \frac{n}{B} \right), & \text{otherwise.} \end{cases} \]

\[ = O \left( \frac{n}{B} \log \frac{n}{M} \right) \]

How to improve this bound?
Cache-Oblivious $k$-Way Merge Sort

I/O Complexity: $Q(n) = \begin{cases} 
O \left(1 + \frac{n}{B}\right), & \text{if } n \leq M, \\
 k \cdot Q \left(\frac{n}{k}\right) + O \left(k + \frac{n}{B}\right), & \text{otherwise.}
\end{cases}$

\[ = O \left( k \cdot \frac{n}{M} + \frac{n}{B} \log_k \frac{n}{M} \right) \]

How large can $k$ be?

Recall that for $k$-way merging, we must ensure

\[ M \geq (k + 1)B \Rightarrow k \leq \frac{M}{B} - 1 \]
Cache-Aware \( \left( \frac{M}{B} - 1 \right) \)-Way Merge Sort

I/O Complexity: 
\[
Q(n) = \begin{cases} 
O \left( 1 + \frac{n}{B} \right), & \text{if } n \leq M, \\
 k \cdot Q \left( \frac{n}{k} \right) + O \left( k + \frac{n}{B} \right), & \text{otherwise.} 
\end{cases}
\]

\[
= O \left( k \cdot \frac{n}{M} + \frac{n}{B} \log_k \frac{n}{M} \right)
\]

Using \( k = \frac{M}{B} - 1 \), we get:
\[
Q(n) = O \left( \left( \frac{M}{B} - 1 \right) \frac{n}{M} + \frac{n}{B} \log_{M/B} \left( \frac{n}{M} \right) \right) = O \left( \frac{n}{B} \log_{M/B} \left( \frac{n}{M} \right) \right)
\]
Sorting
(Funnelsort)
**k-Merger (k-Funnel)**

- **k ≥ 2 sorted input sequences**
- **√k linking buffers** (√k of them)
- **√k - merger** (one)
- **√k - mergers** (√k of them)
- **one merged output sequence**

Memory layout of a k-merger:

<table>
<thead>
<tr>
<th>R</th>
<th>L₁</th>
<th>B₁</th>
<th>L₂</th>
<th>B₂</th>
<th>L_{√k}</th>
<th>B_{√k}</th>
</tr>
</thead>
</table>
Space usage of a $k$-merger: $S(k) = \begin{cases} 
\Theta(1), & \text{if } k \leq 2, \\
(\sqrt{k} + 1)S(\sqrt{k}) + \Theta(k^2), & \text{otherwise.}
\end{cases}$

$= \Theta(k^2)$

A $k$-merger occupies $\Theta(k^2)$ contiguous locations.
Each invocation of a $k$-merger

- produces a sorted sequence of length $k^3$

- incurs $O\left(1 + k + \frac{k^3}{B} + \frac{k^3}{B} \log_M \left(\frac{k}{B}\right)\right)$ cache misses provided $M = \Omega(B^2)$
**k-Merger (k-Funnel)**

$k \geq 2$ sorted input sequences

- $\sqrt{k}$ linking buffers (each of size $2k^2$)
- $\sqrt{k}$-mergers ($\sqrt{k}$ of them)
- $\sqrt{k}$-merger (one)

Memory layout of a $k$-merger:

\[
\begin{array}{cccccc}
R & L_1 & B_1 & L_2 & B_2 & L_{\sqrt{k}} & B_{\sqrt{k}} \\
\end{array}
\]

Cache-complexity:

\[
Q'(k) = \begin{cases} 
O\left(1 + k + \frac{k^3}{B}\right), & \text{if } k < \alpha\sqrt{M}, \\
\left(2k^2 + 2\sqrt{k}\right)Q'(\sqrt{k}) + \Theta(k^2), & \text{otherwise.} 
\end{cases}
\]

\[
= O\left(\frac{k^3}{B} \log_M \left(\frac{k}{B}\right)\right), \quad \text{provided } M = \Omega(B^2)
\]
**k-Merger (k-Funnel)**

$k \geq 2$ sorted input sequences

$k$ linking buffers (each of size $2k^{3\over 2}$)

$\sqrt{k}$ - merger (one)

$\sqrt{k}$ - mergers ($\sqrt{k}$ of them)

One merged output sequence

---

Let $r_i$ be #items extracted the $i$-th input queue. Then $\Sigma_{i=1}^k r_i = O(k^3)$.

Since $k < \alpha \sqrt{M}$ and $M = \Omega(B^2)$, at least $M \over B = \Omega(k)$ cache blocks are available for the input buffers.

Hence, #cache-misses for accessing the input queues (assuming circular buffers) $= \Sigma_{i=1}^k O \left(1 + {r_i \over B}\right) = O \left(k + {k^3 \over B}\right)$

---

Cache-complexity:

$$Q'(k) = \begin{cases} O \left(1 + k + {k^3 \over B}\right), & \text{if } k < \alpha \sqrt{M}, \\ (2k^{3\over 2} + 2\sqrt{k})Q'(\sqrt{k}) + \Theta(k^2), & \text{otherwise}. \end{cases}$$

$$= O \left(\frac{k^3}{B} \log_M \left(\frac{k}{B}\right)\right), \text{ provided } M = \Omega(B^2)$$
\( k \)-Merger ( \( k \)-Funnel)

Memory layout of a \( k \)-merger:

\[
\begin{array}{cccc|c}
R & L_1 & B_1 & L_2 & B_2 & L_\sqrt{k} & B_\sqrt{k} \\
\end{array}
\]

Cache-complexity:

\[
Q'(k) = \begin{cases} 
O\left(1 + k + \frac{k^3}{B}\right), & \text{if } k < \alpha\sqrt{M}, \\
(2k^{\frac{3}{2}} + 2\sqrt{k})Q'(\sqrt{k}) + \Theta(k^2), & \text{otherwise.}
\end{cases}
\]

\[
= O\left(\frac{k^3}{B} \log_M\left(\frac{k}{B}\right)\right), \quad \text{provided } M = \Omega(B^2)
\]

\( k < \alpha\sqrt{M} \): 
\[ Q'(k) = O\left(1 + k + \frac{k^3}{B}\right) \]

- \#cache-misses for accessing the input queues = \( O\left(k + \frac{k^3}{B}\right) \)
- \#cache-misses for writing the output queue = \( O\left(1 + \frac{k^3}{B}\right) \)
- \#cache-misses for touching the internal data structures = \( O\left(1 + \frac{k^2}{B}\right) \)
- Hence, total \#cache-misses = \( O\left(1 + k + \frac{k^3}{B}\right) \)
**k-Merger (k-Funnel)**

- Each call to $R$ outputs $k^2$ items. So, #times merger $R$ is called $= \frac{k^3}{3} = k^2$.

- Each call to an $L_i$ puts $k^2$ items into $B_i$. Since $k^3$ items are output, and the buffer space is $\sqrt{k} \times 2k^2 = 2k^2$, #times the $L_i$’s are called $\leq k^2 + 2\sqrt{k}$.

- Before each call to $R$, the merger must check each $L_i$ for emptiness, and thus incurring $O(\sqrt{k})$ cache-misses. So, #such cache-misses $= k^2 \times O(\sqrt{k}) = O(k^2)$.

---

**Memory layout of a k-merger:**

- $R | L_1 | B_1 | L_2 | B_2 | L_\sqrt{k} | B_\sqrt{k}$

**Cache-complexity:**

$$Q'(k) = \begin{cases} O\left(1 + \frac{k^3}{B}\right), & \text{if } k < \alpha \sqrt{M}, \\ \left(2k^2 + 2\sqrt{k}\right)Q'(\sqrt{k}) + \Theta(k^2), & \text{otherwise.} \end{cases}$$

$$= O\left(\frac{k^3}{B} \log_M \left(\frac{k}{B}\right)\right), \quad \text{provided } M = \Omega(B^2).$$

**k ≥ α√M:**

$$Q'(k) = \left(2k^2 + 2\sqrt{k}\right)Q'(\sqrt{k}) + \Theta(k^2)$$
Funnelsort

- Split the input sequence $A$ of length $n$ into $n^3$ contiguous subsequences $A_1, A_2, \ldots, A_{n^3}$ of length $\frac{n^2}{n^3}$ each
- Recursively sort each subsequence
- Merge the $n^3$ sorted subsequences using a $n^3$-merger

Cache-complexity:

$$Q(n) = \begin{cases} 
O \left( 1 + \frac{n}{B} \right), & \text{if } n \leq M, \\
\frac{1}{n^3} Q \left( \frac{n^2}{n^3} \right) + Q' \left( \frac{1}{n^3} \right), & \text{otherwise}.
\end{cases}$$

$$= \begin{cases} 
O \left( 1 + \frac{n}{B} \right), & \text{if } n \leq M, \\
\frac{1}{n^3} Q \left( \frac{n^2}{n^3} \right) + O \left( \frac{n}{B \log_M \left( \frac{n}{B} \right)} \right), & \text{otherwise}.
\end{cases}$$

$$= O \left( 1 + \frac{n}{B \log_M n} \right)$$