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)

---

**$n = 1000$**

- **Running Times**
- **L1 Cache Misses**
- **L2 Cache Misses**

---

**$n = 2000$**

- **Running Times**
- **L1 Cache Misses**
- **L2 Cache Misses**

---

**$n = 3000$**

- **Running Times**
- **L1 Cache Misses**
- **L2 Cache Misses**
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.
The 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 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 Ideal-Cache Model: Assumptions

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 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 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 tall
  - most practical caches are tall
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 \textit{permutation} problem can match cache-aware I/O bounds [Brodal & Fagerberg, STOC’03].
### Some Known Cache Aware / Oblivious Results

<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 N}{B}\right))</td>
<td>(O\left(\frac{N \log_M N}{B}\right))</td>
</tr>
<tr>
<td>Selection</td>
<td>(O(\text{scan}(N)))</td>
<td>(O(\text{scan}(N)))</td>
</tr>
<tr>
<td>B-Trees [Am] ((\text{Insert, Delete}))</td>
<td>(O\left(\log_B N\right))</td>
<td>(O\left(\log_B N\right))</td>
</tr>
<tr>
<td>Priority Queue [Am] ((\text{Insert, Weak Delete, Delete-Min}))</td>
<td>(O\left(\frac{1}{B} \log_M N\right))</td>
<td>(O\left(\frac{1}{B} \log_M N\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(\frac{V + E}{B}\right) \cdot \log_2 \frac{V}{B}\right))</td>
<td>(O\left(\left(\frac{V + 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, \frac{V + \text{sort}(E)}{B}\right)\right))</td>
<td>(O\left(\min\left(\text{sort}(E) \log_2 \log_2 \frac{VB}{E}, \frac{V + \text{sort}(E)}{B}\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. for i ← 1 to n do
2. for j ← 1 to n do
3. for k ← 1 to n do
4. \( z_{ij} \leftarrow z_{ij} + x_{ik} \times y_{kj} \)
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 for loop in line 3 incurs \( O(n) \) cache misses.

I/O-complexity of Iter-MM, \( Q(n) = O(n^3) \)
Each iteration of the `for` loop in line 3 incurs $O \left( 1 + \frac{n}{B} \right)$ cache misses.

I/O-complexity of `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

\[ \text{cache ( size } = M \text{ )} \]

\[
\begin{array}{c|c|c}
\frac{M}{3} & \frac{M}{3} & \frac{M}{3} \\
\end{array}
\]

\[
\begin{array}{c}
\frac{m}{3} \\
\end{array}
\]

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

1. for \( i \leftarrow 1 \) to \( n / m \) do
2. for \( j \leftarrow 1 \) to \( n / m \) do
3. for \( k \leftarrow 1 \) to \( n / m \) 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 \textit{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 \( m = \sqrt[3]{M/2} \), 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:

\[
\mathbf{z}_{ij} = \sum_{k=1}^{n} \mathbf{x}_{ik} \mathbf{y}_{kj}
\]

I/O-complexity:

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

( Optimal: Hong & Kung, STOC’81 )
Multiple Levels of Cache

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

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

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

1. for $i_1 ← 1$ to $n / s$ do
2. for $j_1 ← 1$ to $n / s$ do
3. for $k_1 ← 1$ to $n / s$ do
4. for $i_2 ← 1$ to $s / t$ do
5. for $j_2 ← 1$ to $s / t$ do
6. for $k_2 ← 1$ to $s / t$ do
7. $Iter-MM( (X_{i_1k_1})_{i_2k_2}, (Y_{k_1j_1})_{k_2j_2}, (X_{i_1j_1})_{i_2j_2}, t )$
Multiple Levels of Cache

One Parameter Per Caching Level!

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

1. for $i_1 ← 1$ to $n / s$ do
2. for $j_1 ← 1$ to $n / s$ do
3. for $k_1 ← 1$ to $n / s$ do
4. for $i_2 ← 1$ to $s / t$ do
5. for $j_2 ← 1$ to $s / t$ do
6. for $k_2 ← 1$ to $s / t$ do
7. Iter-MM( (X_{i_1k_1})_{i_2k_2}, (Y_{k_1j_1})_{k_2j_2}, (X_{i_1j_1})_{i_2j_2}, t )
Recursive Matrix Multiplication

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

Rec-MM( Z, X, Y )

1. if $Z \equiv 1 \times 1$ matrix then $Z \leftarrow Z + X \cdot Y$

2. else

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

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

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

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

Rec-MM( Z, X, Y )

1. if Z ≡ 1 × 1 matrix then Z ← Z + X · 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} )

I/O-complexity ( for 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 \left( B^2 \right)

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

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

Source: wikipedia
Recursive Matrix Multiplication with Z-Morton Layout

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

1. \textit{if} \ Z \equiv 1 \times 1 \text{ matrix 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}) \\
4. \text{Rec-MM}(Z_{12}, X_{12}, Y_{12}), \text{Rec-MM}(Z_{12}, X_{12}, Y_{22}) \\
5. \text{Rec-MM}(Z_{21}, X_{21}, Y_{11}), \text{Rec-MM}(Z_{21}, X_{22}, Y_{21}) \\
6. \text{Rec-MM}(Z_{22}, X_{21}, Y_{12}), \text{Rec-MM}(Z_{22}, X_{22}, Y_{22}) \\

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), \text{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>x: 0 1 2 3 4 5 6 7</th>
<th>y: 0 1 2 3 4 5 6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</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, $h = \Theta(\log_2 n)$
A Static Search Tree

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

**Recursive Subdivision**
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 $k = \Theta(\sqrt{n})$. 

Recursive Subdivision
van Emde Boas Layout

Recursive Subdivision

If the tree contains $n$ nodes, each subtree contains $\Theta(2^{h/2}) = \Theta(\sqrt{n})$ nodes, and $k = \Theta(\sqrt{n})$. 
I/O-Complexity of a Search

- The height of the tree is $\log n$
- Each △ has height between $\frac{1}{2}\log B$ & $\log B$.
- Each △ spans at most 2 blocks of size $B$. 
I/O-Complexity of a Search

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

- $p =$ number of △‘s 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

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

1. \textit{if} \: p < r \: \textit{then}
2. \quad q \leftarrow \left\lfloor \frac{p + r}{2} \right\rfloor
3. \quad \text{Merge-Sort} \left( A, p, q \right)
4. \quad \text{Merge-Sort} \left( A, q + 1, r \right)
5. \quad \text{Merge} \left( A, p, q, r \right)
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-oblivious 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

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

1. if \( p < r \) then
2. \( q \leftarrow \lfloor (p + r) / 2 \rfloor \)
3. \text{Merge-Sort} \left( A, p, q \right)
4. \text{Merge-Sort} \left( A, q + 1, r \right)
5. \text{Merge} \left( A, p, q, r \right)

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:

$$Q(n) = O\left(\left(\frac{M}{B} - 1\right)\frac{n}{M} + \frac{n}{B} \log_M \left(\frac{n}{M}\right)\right) = O\left(\frac{n}{B} \log_M \left(\frac{n}{M}\right)\right)$$
Sorting ( Funnelsort )
$k$-Merger ($k$-Funnel)

$k \geq 2$ sorted input sequences

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

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

Memory layout of a $k$-merger:

<table>
<thead>
<tr>
<th>$R$</th>
<th>$L_1$</th>
<th>$B_1$</th>
<th>$L_2$</th>
<th>$B_2$</th>
<th>$L_{\sqrt{k}}$</th>
<th>$B_{\sqrt{k}}$</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 $O\left(1 + \frac{k}{B} + \frac{k^3}{B \log M} \left(\frac{k}{B}\right)^2\right)$ cache misses provided $M = \Omega(B^2)$
**k-Merger ( k-Funnel)**

$k \geq 2$ sorted input sequences

\[
\sqrt{k} \text{ linking buffers (each of size } 2k^2)\]

\[
\sqrt{k} \text{ - merger (one)}\]

\[
\sqrt{k} \text{ - mergers (} \sqrt{k} \text{ of them)}\]

\[
\text{Memory layout of a } k\text{-merger:}
\]

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

**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

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

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

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

one merged output sequence

Let $c_i$ be #items extracted the $i$-th input queue. Then $\sum c_i = O(1 + k^3)$. Since $k < \alpha \sqrt{M}$ and $M = \Omega(B^2)$, at least $\frac{M}{B} = \Omega(k)$ cache blocks are available for the input buffers.

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

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

provided $M = \Omega(B^2)$

Memory layout of a $k$-merger:

<table>
<thead>
<tr>
<th>$R$</th>
<th>$L_1$</th>
<th>$B_1$</th>
<th>$L_2$</th>
<th>$B_2$</th>
<th>$L_{\sqrt{k}}$</th>
<th>$B_{\sqrt{k}}$</th>
</tr>
</thead>
</table>

$k < \alpha \sqrt{M}$: $Q'(k) = O \left( 1 + k + \frac{k^3}{B} \right)$
**$k$-Merger ($k$-Funnel)**

- **Memory layout of a $k$-merger:**
  
  $\begin{array}{c|cccc}
  R & L_1 & B_1 & L_2 & B_2 \\
  \hline
  L_{\sqrt{k}} & B_{\sqrt{k}} \\
  \end{array}$

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

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

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

- **#cache-misses for accessing the input queues**
  
  $= O\left(\frac{k + k^3}{B}\right)$

- **#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**
  
  $= O\left(1 + k + \frac{k^3}{B}\right)$
\[ k \geq \alpha \sqrt{M}: \quad Q'(k) = \left(2k^{\frac{3}{2}} + 2\sqrt{k}\right) Q'\left(\sqrt{k}\right) + \Theta(k^2) \]

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

- Each call to an \( L_i \) puts \( k^2 \) items into \( B_i \). Since \( k^3 \) items are output, and the buffer space is \( \sqrt{k} \times 2k^{\frac{3}{2}} = 2k^2 \), \#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 \( O\left(\sqrt{k}\right) \) cache-misses. So, \#such cache-misses = \( k^2 \times O\left(\sqrt{k}\right) = \)
Funnelsort

- Split the input sequence $A$ of length $n$ into $\frac{1}{n^3}$ contiguous subsequences $A_1, A_2, \ldots, A_{\frac{1}{n^3}}$ each
- Recursively sort each subsequence
- Merge the $\frac{1}{n^3}$ sorted subsequences using a $\frac{1}{n^3}$-merger

Cache-complexity:

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

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

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