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 \( k = 0; k < n; k++ \) )
      \( Z[i][j] += X[i][k] \ast Y[k][j] \);

**I-K-J**

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

**J-I-K**

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

**J-K-I**

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

**K-I-J**

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

**K-J-I**

for (int \( k = 0; k < n; k++ \) )
  for (int \( j = 0; j < n; j++ \) )
    for (int \( i = 0; i < n; i++ \) )
      \( Z[i][j] += X[i][k] \ast Y[k][j] \);
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
- capacitance of long connecting wires, etc.

A reasonable compromise is to use a **memory hierarchy**.
A *memory hierarchy* is intended to be

- 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 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 lower running time

Memory-bound Algorithm:
- low op-space ratio
- more time spent in transferring data than computing
- a faster memory system leads to a lower 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 $\text{sort}(N) = \Theta \left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$

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 Ideal-Cache Model: Assumptions

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 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
The Ideal-Cache Model: I/O Bounds

Cache-oblivious vs. cache-aware bounds:

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

- 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]
# 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 (scan(N))</td>
<td>$O\left(\frac{N}{B}\right)$</td>
<td>$O\left(\frac{N}{B}\right)$</td>
</tr>
<tr>
<td>Sorting (sort(N))</td>
<td>$O\left(\frac{N \log \frac{M}{B}}{B} \cdot \frac{N}{B}\right)$</td>
<td>$O\left(\frac{N \log \frac{M}{B}}{B} \cdot \frac{N}{B}\right)$</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] (Insert, Delete)</td>
<td>$O\left(\log \frac{N}{B}\right)$</td>
<td>$O\left(\log \frac{N}{B}\right)$</td>
</tr>
<tr>
<td>Priority Queue [Am] (Insert, Weak Delete, Delete-Min)</td>
<td>$O\left(\frac{1}{B} \cdot \log \frac{M}{B} \cdot \frac{N}{B}\right)$</td>
<td>$O\left(\frac{1}{B} \cdot \log \frac{M}{B} \cdot \frac{N}{B}\right)$</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>$O\left([\frac{V + E}{B}] \cdot \log_2 \frac{V}{B}\right)$</td>
<td>$O\left([\frac{V + E}{B}] \cdot \log_2 \frac{V}{B}\right)$</td>
</tr>
<tr>
<td>Minimum Spanning Forest</td>
<td>$O\left(\min\left(\text{sort}(E) \cdot \log_2 \log_2 V, \frac{V}{B} + \text{sort}(E)\right)\right)$</td>
<td>$O\left(\min\left(\text{sort}(E) \cdot \log_2 \log_2 \frac{VB}{E}, \frac{V}{B} + \text{sort}(E)\right)\right)$</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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. \(z_{ij} \leftarrow z_{ij} + x_{ik} \times y_{kj}\)

\[\begin{array}{cccc}
  z_{11} & z_{12} & \ldots & z_{1n} \\
  z_{21} & z_{22} & \ldots & z_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  z_{n1} & z_{n2} & \ldots & z_{nn}
\end{array}\]

\[\begin{array}{cccc}
  x_{11} & x_{12} & \ldots & x_{1n} \\
  x_{21} & x_{22} & \ldots & x_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{n1} & x_{n2} & \ldots & x_{nn}
\end{array}\]

\[\begin{array}{cccc}
  y_{11} & y_{12} & \ldots & y_{1n} \\
  y_{21} & y_{22} & \ldots & y_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  y_{n1} & y_{n2} & \ldots & y_{nn}
\end{array}\]

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

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

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 $\text{Block-MM}$ [assuming a 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 \( X_{ik}, Y_{kj} \), and \( Z_{ij} \) just fit into the cache.

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

\[
Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}
\]

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

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

( 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 )
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. $\text{Iter-MM}( (X_{i_1,k_1})_{i_2,k_2}, (Y_{k_1,j_1})_{k_2,j_2}, (X_{i_1,j_1})_{i_2,j_2}, t )$
Multiple Levels of Cache

One Parameter Per Caching Level

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

1. for \( i_1 \leftarrow 1 \) to \( n / s \) do
2. \( \text{for } j_1 \leftarrow 1 \) to \( n / s \) do
3. \( \text{for } k_1 \leftarrow 1 \) to \( n / s \) do
4. \( \text{for } i_2 \leftarrow 1 \) to \( s / t \) do
5. \( \text{for } j_2 \leftarrow 1 \) to \( s / t \) do
6. \( \text{for } k_2 \leftarrow 1 \) 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

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

\[
\begin{align*}
\text{Rec-MM}(Z, X, Y) & \\
1. & \text{ if } Z \equiv 1 \times 1 \text{ matrix then } Z \leftarrow Z + X \cdot Y \\
2. & \text{ else } \\
3. & \text{ Rec-MM}(Z_{11}, X_{11}, Y_{11}), \text{ Rec-MM}(Z_{11}, X_{12}, Y_{21}) \\
4. & \text{ Rec-MM}(Z_{12}, X_{12}, Y_{12}), \text{ Rec-MM}(Z_{12}, X_{12}, Y_{22}) \\
5. & \text{ Rec-MM}(Z_{21}, X_{21}, Y_{11}), \text{ Rec-MM}(Z_{21}, X_{22}, Y_{21}) \\
6. & \text{ Rec-MM}(Z_{22}, X_{21}, Y_{12}), \text{ Rec-MM}(Z_{22}, X_{22}, Y_{22})
\end{align*}
\]
Recursive Matrix Multiplication

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

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. \( \text{Rec-MM}(Z_{22}, X_{21}, Y_{12}), \text{Rec-MM}(Z_{22}, X_{22}, Y_{22}) \)

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

\[ = 0\left(\frac{n^3}{M} + \frac{n^3}{B\sqrt{M}}\right) = 0\left(\frac{n^3}{B\sqrt{M}}\right), \text{when } M = \Omega(B^2) \]

I/O-complexity (for all \( n \)) = \( 0\left(\frac{n^3}{B\sqrt{M}} + \frac{n^2}{B} + 1\right) \) (why?)
Recursive Matrix Multiplication with Z-Morton Layout
Recursive Matrix Multiplication with Z-Morton Layout

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_{1111}$</td>
<td>$Z_{1112}$</td>
<td>$Z_{1211}$</td>
<td>$Z_{1212}$</td>
<td></td>
</tr>
<tr>
<td>$Z_{1121}$</td>
<td>$Z_{1122}$</td>
<td>$Z_{1221}$</td>
<td>$Z_{1222}$</td>
<td></td>
</tr>
<tr>
<td>$Z_{2111}$</td>
<td>$Z_{2112}$</td>
<td>$Z_{2211}$</td>
<td>$Z_{2212}$</td>
<td></td>
</tr>
<tr>
<td>$Z_{2121}$</td>
<td>$Z_{2122}$</td>
<td>$Z_{2221}$</td>
<td>$Z_{2222}$</td>
<td></td>
</tr>
</tbody>
</table>

$Z_{11}$ | $Z_{12}$ | $Z_{21}$ | $Z_{22}$ |   |
Recursive Matrix Multiplication with Z-Morton Layout

Source: wikipedia
Recursive Matrix Multiplication with Z-Morton Layout

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

1. \( \text{if } Z \equiv 1 \times 1 \text{ matrix then } Z \leftarrow Z + X \cdot Y \)
2. \( \text{else} \)
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, \( Q(n) = \begin{cases} 
0 \left(1 + \frac{n^2}{B}\right), & \text{if } n^2 \leq \alpha M \\
8Q \left(\frac{n}{2}\right) + O(1), & \text{otherwise}
\end{cases} \)

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

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></th>
<th>x:</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>y:</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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.

$$h = \Theta(\log_2 n)$$
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

h

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
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 \( k = \Theta(\sqrt{n}) \).
van Emde Boas Layout

A recursive subdivision of a binary search tree.

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

- $p = \text{number of } \triangle\text{'s visited by a search path}$

- 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

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)
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

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

1. if p < r then
2. \( q \leftarrow \lceil (p + r) / 2 \rceil \)
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?
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 \( (\frac{M}{B} - 1) \)-Way Merge Sort

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

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

Using \( k = \frac{M}{B} - 1 \), we get:

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

$k \geq 2$ sorted input sequences

$k$ linking buffers (each of size $2k^2$)

$k$-merging buffers

one merged output sequence

Memory layout of a $k$-merger:

\[
\begin{array}{ccccccc}
R & L_1 & B_1 & L_2 & B_2 & L_{\sqrt{k}} & B_{\sqrt{k}}
\end{array}
\]
**k-Merger (k-Funnel)**

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 $M = \Omega(B^2)$
\(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)
$k$-Merger ($k$-Funnel)

$k \geq 2$ sorted input sequences

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

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

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

one merged output sequence

Let $s_i$ be the $i$-th input queue. Then

$$\sum_{i=1}^{\sqrt{k}} s_i = O(k^3).$$

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) is

$$\sum_{i=1}^{\sqrt{k}} O\left(1 + \frac{r_i}{B}\right) = O\left(k + \frac{k^3}{B}\right).$$

Cache-complexity:

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

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

\[ k \geq 2 \text{ sorted input sequences} \]

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

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

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

\[ \frac{k}{\sqrt{k}} \text{ - merger output sequence} \]

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

| R | L₁ | B₁ | L₂ | B₂ | \( L_{\frac{k}{\sqrt{k}}} \) | \( B_{\frac{k}{\sqrt{k}}} \) |

**Cache-complexity:**

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

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

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

- \#cache-misses for accessing the input queues = \( O \left(1 + \frac{k^3}{B}\right) \)
- \#cache-misses for writing the output queue = \( O \left(\frac{k^3}{B}\right) \)
- \#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)

$k \geq 2$ sorted input sequences

$k \geq \alpha \sqrt{M}$: $Q'(k) = \left(2k^{\frac{3}{2}} + 2\sqrt{k}\right)Q'(\sqrt{k}) + \Theta(k^2)$

- Each call to $R$ outputs $k^{\frac{3}{2}}$ items. So, #times merger $R$ is called $= \frac{k^3}{k^{\frac{3}{2}}} = k^{\frac{3}{2}}$
- Each call to an $L_i$ puts $k^{\frac{3}{2}}$ 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 $\leq k^{\frac{3}{2}} + 2\sqrt{k}$
- 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 $= k^{\frac{3}{2}} \times O\left(\sqrt{k}\right) = O(k^2)$

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>

Cache-complexity:

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

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

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

Cache-complexity:

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

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

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