CSE 638: Advanced Algorithms

Lectures 16 & 17
(Analyzing I/O and Cache Performance)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2013
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 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 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]
## 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)) (\frac{N}{B})</td>
<td>(O\left(\frac{N \log_m N}{B} \right)) (\frac{N}{B})</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 ([\text{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 ([\text{Am}]) ((\text{Insert, Weak Delete, Delete-Min}))</td>
<td>(O\left(\frac{1}{B} \log_m N \right)) (\frac{N}{B})</td>
<td>(O\left(\frac{1}{B} \log_m N \right)) (\frac{N}{B})</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(V + \frac{E}{B}\right) \cdot \log_2 \frac{V}{B})</td>
<td>(O\left(V + \frac{E}{B}\right) \cdot \log_2 \frac{V}{B})</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>

| \(N\) = \#elements, \(V\) = \#vertices, \(E\) = \#edges, \(\text{Am}\) = 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. for \( j \leftarrow 1 \) to \( n \) do
3. for \( k \leftarrow 1 \) 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 $i \leftarrow 1$ to $n$ do
2. for $j \leftarrow 1$ to $n$ do
3. for $k \leftarrow 1$ 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 \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

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

1. for i ← 1 to n / m do
2. for j ← 1 to n / m do
3. for k ← 1 to n / m do
4. 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 $X$, $Y$, and $Z$ just fit into the cache.

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

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

The \( I/O \)-complexity of \( \text{Block-MM}(X, Y, Z, n) \)

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

( 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)
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( $(X_{i_1k_1i_2k_2})_{i_1j_1k_2j_2}$, $(Y_{k_1j_1k_2j_2})_{i_1j_1i_2j_2}$, $(Z_{i_1j_1i_2j_2})_{i_1j_1i_2j_2}$, 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. \hspace{1em} for \( j_1 \leftarrow 1 \) to \( n / s \) do
3. \hspace{2em} for \( k_1 \leftarrow 1 \) to \( n / s \) do
4. \hspace{3em} for \( i_2 \leftarrow 1 \) to \( s / t \) do
5. \hspace{4em} for \( j_2 \leftarrow 1 \) to \( s / t \) do
6. \hspace{5em} 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

\[
\begin{align*}
Z &= \begin{pmatrix}
Z_{11} & Z_{12} \\
Z_{21} & Z_{22}
\end{pmatrix} \\
X &= \begin{pmatrix}
X_{11} & X_{12} \\
X_{21} & X_{22}
\end{pmatrix} \\
Y &= \begin{pmatrix}
Y_{11} & Y_{12} \\
Y_{21} & Y_{22}
\end{pmatrix}
\end{align*}
\]

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

\[
\begin{array}{c|c}
Z_{11} & Z_{12} \\
\hline
Z_{21} & Z_{22}
\end{array}
\quad = \quad
\begin{array}{c|c}
X_{11} Y_{11} + X_{12} Y_{21} & X_{11} Y_{12} + X_{12} Y_{22} \\
\hline
X_{21} Y_{11} + X_{22} Y_{21} & X_{21} Y_{12} + X_{22} Y_{22}
\end{array}
\]

\textit{Rec-MM}( Z, X, Y )

1. \textit{if} \ Z \equiv 1 \times 1 \text{ matrix} \ \textit{then} \ Z \leftarrow Z + X \cdot Y
2. \textit{else}
3. \textit{Rec-MM}( Z_{11}, X_{11}, Y_{11} ), \ \textit{Rec-MM}( Z_{11}, X_{12}, Y_{21} )
4. \textit{Rec-MM}( Z_{12}, X_{12}, Y_{12} ), \ \textit{Rec-MM}( Z_{12}, X_{12}, Y_{22} )
5. \textit{Rec-MM}( Z_{21}, X_{21}, Y_{11} ), \ \textit{Rec-MM}( Z_{21}, X_{22}, Y_{21} )
6. \textit{Rec-MM}( Z_{22}, X_{21}, Y_{12} ), \ \textit{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 (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), \(Q(n) = \begin{cases} 
0 \left( n + \frac{n^2}{B} \right), & \text{if } n^2 \leq \alpha M \\
8Q \left( \frac{n}{2} \right) + O(1), & \text{otherwise}
\end{cases}\)

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

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

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

\[ Z_{1111} \hspace{1cm} Z_{1112} \hspace{1cm} Z_{1211} \hspace{1cm} Z_{1212} \\
Z_{1121} \hspace{1cm} Z_{1122} \hspace{1cm} Z_{1221} \hspace{1cm} Z_{1222} \\
Z_{2111} \hspace{1cm} Z_{2112} \hspace{1cm} Z_{2211} \hspace{1cm} Z_{2212} \\
Z_{2121} \hspace{1cm} Z_{2122} \hspace{1cm} Z_{2221} \hspace{1cm} Z_{2222} \\
\]
Recursive Matrix Multiplication with Z-Morton Layout

Source: wikipedia
Recursive Matrix Multiplication with Z-Morton Layout

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(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>000</td>
<td>001</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>y:</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00000</td>
<td>00001</td>
<td>00100</td>
<td>00101</td>
<td>01000</td>
<td>01001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00010</td>
<td>00011</td>
<td>00110</td>
<td>00111</td>
<td>01010</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00100</td>
<td>00101</td>
<td>00110</td>
<td>00111</td>
<td>01100</td>
<td>01101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01000</td>
<td>01001</td>
<td>01100</td>
<td>01101</td>
<td>01110</td>
<td>01111</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>10001</td>
<td>10010</td>
<td>10011</td>
<td>11000</td>
<td>11001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10010</td>
<td>10011</td>
<td>10100</td>
<td>10101</td>
<td>11010</td>
<td>11011</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10100</td>
<td>10101</td>
<td>10110</td>
<td>10111</td>
<td>11100</td>
<td>11101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10110</td>
<td>10111</td>
<td>11010</td>
<td>11011</td>
<td>11110</td>
<td>11111</td>
<td></td>
</tr>
</tbody>
</table>