Cache Oblivious Algorithms and Data Structures
*Theory and Practice*

Hitesh Ballani
Department of Computer Science
Cornell University

CS 612
31\textsuperscript{st} March
Multi-level memory hierarchies are omnipresent
Memory speed $\propto (\text{Distance from processor})^{-1}$
Good locality is important for achieving high performance
Modern hardware is not uniform - many different parameters

In homework 1, we used X-RAY to measure

- CPU speed
- Instruction Latency/Throughput
- Number of registers
- Special Instructions (eg. fma)
- Cache Stride/Associativity/Capacity/Line-Size/Hit-Latency

Current programs

- ignore the parameters - poor performance
- determine the parameters

its a jungle out there
Modern hardware is not uniform - many different parameters

In homework 1, we used X-RAY to measure
- CPU speed
- Instruction Latency/Throughput
- Number of registers
- Special Instructions (e.g., fma)
- Cache Stride/Associativity/Capacity/Line-Size/Hit-Latency

Current programs
- ignore the parameters - poor performance
- determine the parameters
  - by hand, e.g., hand-optimized BLAS libraries
  - automatically, e.g., ATLAS generated BLAS libraries
Modern hardware is not uniform - many different parameters

In homework 1, we used X-RAY to measure
- CPU speed
- Instruction Latency/Throughput
- Number of registers
- Special Instructions (eg. fma)
- Cache Stride/Associativity/Capacity/Line-Size/Hit-Latency

Current programs
- ignore the parameters - poor performance
- determine the parameters
  - by hand, eg. hand-optimized BLAS libraries
  - automatically, eg. ATLAS generated BLAS libraries
Is it possible to abstract away this complexity?

- A model that
  - could capture the essence of the hierarchy
  - without knowing its specifics
- Algorithms that are efficient on all hierarchies simultaneously
- and this holy grail is what *Cache Oblivious Algorithms* aim to attain
Is it possible to abstract away this complexity?

- A model that
  - could capture the essence of the hierarchy
  - without knowing its specifics
- Algorithms that are efficient on all hierarchies simultaneously
- and this holy grail is what Cache Oblivious Algorithms aim to attain
Is it possible to abstract away this complexity?

- A model that
  - could capture the essence of the hierarchy
  - without knowing its specifics

- Algorithms that are efficient on all hierarchies simultaneously

- and this holy grail is what *Cache Oblivious Algorithms* aim to attain
The model we use to analyze algorithms in CS 681

- All basic operations take up constant time

Complexity is the number of operations executed

- Limited practical use
  - Does not take into account the differences of speeds of random access to memory

Hierarchical Memory Models

- account for multi-level hierarchies
- for eg, Aggarwal et. al., ’87
- too complicated for practical use
The model we use to analyze algorithms in CS 681
- All basic operations take up constant time
- Complexity is the number of operations executed
- Limited practical use
  - Does not take into account the differences of speeds of random access to memory

Hierarchical Memory Models
- account for multi-level hierarchies
- for eg, Aggarwal et. al., ’87
- too complicated for practical use
The model we use to analyze algorithms in CS 681
- All basic operations take up constant time
- **Complexity is the number of operations executed**
- Limited practical use
  - Does not take into account the differences of speeds of random access to memory
- **Hierarchical Memory Models**
  - account for multi-level hierarchies
  - for eg, *Aggarwal et. al., ’87*
  - too complicated for practical use
External Memory Model (I/O Model)

*a two parameter model*

- 2 storage levels
  - Cache
  - Memory

- Complexity - number of transfers between cache and memory

**Limitations**

- Parameters B and M must be known
- Does not handle multiple memory levels

*Aggarwal and Vitter, ’88*
External Memory Model (I/O Model)

*a two parameter model*

- 2 storage levels
  - Cache
  - Memory
- Complexity - number of transfers between cache and memory

Limitations
- Parameters B and M must be known
- Does not handle multiple memory levels
External Memory Model (I/O Model)
a two parameter model

- 2 storage levels
  - Cache
  - Memory
- Complexity - number of transfers between cache and memory

Limitations
- Parameters $B$ and $M$ must be known
- Does not handle multiple memory levels

Aggarwal and Vitter, ’88
Cache Oblivious Model (Ideal Cache Model)  

*no parameters*

**Key Insight**

Design algorithms without knowing B and M

- **Design**
  - Know the existence of the hierarchy
  - Not the parameters

**Advantages**

- Simple and Portable
- Automatically Tuned for hierarchy*
- Efficiency in the asymptotic sense*
Cache Oblivious Model (Ideal Cache Model)

no parameters!

Key Insight

Design algorithms without knowing B and M

- Design
  - Know the existence of the hierarchy
  - Not the parameters

Advantages

- Simple and Portable
- Automatically Tuned for hierarchy*
- Efficiency in the asymptotic sense*
**Cache Oblivious Model (Ideal Cache Model)**

*no parameters!*

---

**Key Insight**

Design algorithms without knowing $B$ and $M$

---

**Design**

- Know the existence of the hierarchy
- Not the parameters

---

**Advantages**

- Simple and Portable
- Automatically Tuned for hierarchy*
- Efficiency in the asymptotic sense*
Assumptions made by the CO model
Frigo et. al.’99

**Optimal Cache Replacement**
LRU can be used instead, with no *asymptotic* loss in performance

Frigo et. al.,’99

**Full Associativity**
Can be simulated in ordinary memory with *constant slowdown*

Sleator and Tarjan,’85
Frigo et. al.,’99
Assumptions made by the CO model
Frigo et. al.’99

**Optimal Cache Replacement**

LRU can be used instead, with no *asymptotic* loss in performance

*Sleator and Tarjan,’85*

**Full Associativity**

Can be simulated in ordinary memory with *constant slowdown*

*Frigo et. al.,’99*
Assumptions made by the CO model
Frigo et. al.’99

Optimal Cache Replacement
LRU can be used instead, with no *asymptotic* loss in performance

Full Associativity
Can be simulated in ordinary memory with *constant slowdown*

Random thought
Do these constant factors hurt performance in practice?

Frigo et. al.,'99

Sleator and Tarjan,’85

Hitesh Ballani

Cache Oblivious Algorithms and Data Structures
So where is this obliviousness used?

In CS612...

- we looked at cache models
- learnt how to transform programs to improve performance

Cache Obliviousness is a tool to build

- Asymptotically efficient algorithms and data structures
- "Programs = Algorithms + Data Structures" - Niklaus Wirth
- Asymptotically efficient programs
So where is this obliviousness used?

In CS612 . . .
- we looked at cache models
- learnt how to transform programs to improve performance

Cache Obliviousness is a tool to build
- Asymptotically efficient algorithms and data structures
- "Programs = Algorithms + Data Structures" - Niklaus Wirth
- Asymptotically efficient programs
CO Algorithms are hot property
a cottage industry in itself

Their simplicity holds a lot of promise
- Proposed by Frigo et. al. in 1999
- More than 30 papers already

Existing Algorithms
- Numerical Algorithms: Matrix Mult./Transpose, FFT...
- Searching: Van Emde Boas Layout, B-Trees ...
- Sorting: Funnel Sort, Distribution Sort ...
- Data Structures: Priority Queues, Ordered File Maintenance ...
- Other areas include "application-level" problems in computational geometry, graph algorithms, etc.
Main Tool: Divide and Conquer

- Divide the problem recursively
- Solve the trivial problem directly

What's the relation to CO algorithms?
- Trivial problem fits in the cache $\Rightarrow$ good performance
- Results applicable to multi-level hierarchy

Think of CO algorithms as a "catch" phrase
- Divide and Conquer algorithms are CO, for eg. quicksort, mergesort, median selection etc.
- May not achieve optimal performance
Main Tool: Divide and Conquer

Divide and Conquer
- Divide the problem recursively
- Solve the trivial problem directly

What’s the relation to CO algorithms?
- Trivial problem fits in the cache $\Rightarrow$ good performance
- Results applicable to multi-level hierarchy

Think of CO algorithms as a "catch" phrase
- Divide and Conquer algorithms are CO, for eg. quicksort, mergesort, median selection etc.
- May not achieve optimal performance
Matrix Transposition: the naive approach

- Transpose \( m \times n \) matrix, \( B = A^T \)

\[
\text{for } i = 1 \text{ to } m \\
\text{for } j = 1 \text{ to } n \\
B(j,i) = A(i,j)
\]

Column Access!!

Cache Complexity (# of cache misses)

\[
Q(m,n) = O(mn)
\]
Matrix Transposition: Divide and Conquer

Transpose $m \times n$ matrix, $B = A^T$

Case 1: $m = n = 1$

Case 2: $n \geq m$

Case 3: $m > n$

Cache Complexity (# of cache misses)

$Q(m,n) = O(1 + mn/B), B = \text{Cache Line Size}$
### Algorithms along the same lines

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>$O(1 + n/B(1 + \log_M n))$</td>
</tr>
<tr>
<td>Frigo et. al.’99</td>
<td>Vitter et. al.’94</td>
</tr>
<tr>
<td>MMM</td>
<td>$O(n + n^2/B + n^3/(B\sqrt{M}))$</td>
</tr>
<tr>
<td>Frigo et. al.’99</td>
<td>Hong et. al.’81</td>
</tr>
<tr>
<td>Strassen’s MMM</td>
<td>$O(n + n^2/B + n^{\log_7/(B\sqrt{M})})$</td>
</tr>
<tr>
<td>Frigo et. al.’99</td>
<td>Strassen’69</td>
</tr>
<tr>
<td>Median and Selection</td>
<td>$O(1 + n/B)$</td>
</tr>
<tr>
<td>Demaine’02</td>
<td>Blum et. al.’73</td>
</tr>
<tr>
<td>Jacobi’s method (2D)</td>
<td>$O((mn)^2/(B\sqrt{M}))$</td>
</tr>
<tr>
<td>Chung et. al.’04</td>
<td>??</td>
</tr>
</tbody>
</table>
Performance in Practice

out of Cinderella’s world

- Cache Oblivious Algorithms
  - Good cache performance
  - Poor Execution Time
  - Slower than not so naive algorithms

Mainly due to function call overhead

- Function calls in Matrix Transposition (worst case)
  - \( \log(mn) \) nodes in the recursion tree
  - \( (2mn - 1) \) function calls
Cache Oblivious Algorithms
- Good cache performance
- Poor Execution Time
- Slower than not so naive algorithms

Mainly due to function call overhead
- Function calls in Matrix Transposition (worst case)
  - log(mn) nodes in the recursion tree
  - (2mn - 1) function calls
Performance in practice

- from Kumar’03
- Notebook, Windows 2K, 512MB RAM, PIII 1GHz, g++ -O3
Need for tuning!!

- Recursion call overhead
- Stop recursion early
  
  
  "... the code is still subject to some tuning, e.g., where to trim the base case of a recursion ..."

Demaine, '02

Adaptive Cache Oblivious Algorithms

- Use accurate timing function
- Self-tuning for a good recursion depth
Need for tuning!!

- Recursion call overhead
- Stop recursion early
  - "... the code is still subject to some tuning, e.g., where to trim the base case of a recursion ..."

Adaptive Cache Oblivious Algorithms

- Use accurate timing function
- Self-tuning for a good recursion depth
Need for tuning!!

- from Chung’04
- Recursion overhead is significant
- Adaptivity is important
Static Search Tree (Binary Search)

Bender et. al.’00

Static Search Tree

- Fundamental tool in many data structures
- A perfectly balanced binary search tree
- Static: no insertions and deletions

How do we search with few cache misses?

- Optimal bounds
  - Comparisons: $O(\log N)$
  - Memory Transfers: $O(\log_B N)$
- A perfectly balanced binary tree
  - Comparisons: $O(\log N)$
- How to minimize the cache misses?
Static Search Tree (Binary Search)

_Bender et. al.’00_

**Static Search Tree**
- Fundamental tool in many data structures
- A perfectly balanced binary search tree
- Static: no insertions and deletions

**How do we search with few cache misses?**
- Optimal bounds
  - Comparisons: $O(\log N)$
  - Memory Transfers: $O(\log_B N)$
- A perfectly balanced binary tree
  - Comparisons: $O(\log N)$
- How to minimize the cache misses?
How to minimize the cache misses?

Prokop’99

Choosing the memory layout

- **Layout**: Mapping of nodes of a tree to memory cells
- Different kinds of layouts
  - In-order
  - Breadth-first
  - Depth-first
  - van Emde Boas

van Emde Boas Layout: Main Idea

Store recursive sub-trees in contiguous memory
How to minimize the cache misses?
Prokop’99

Choosing the memory layout

- **Layout**: Mapping of nodes of a tree to memory cells
- Different kinds of layouts
  - In-order
  - Breadth-first
  - Depth-first
  - van Emde Boas

van Emde Boas Layout: Main Idea

Store recursive sub-trees in contiguous memory
van Emde Boas layout

- Split the tree at the middle level of edges
  - One top recursive subtree
  - $\sim \sqrt{N}$ bottom recursive subtrees: size $\sim \sqrt{N}$
- Recursively layout the top and the bottom subtrees
Tree Height = 4

Actual layout of Tree in memory:
1, 2, 3, 4, 8, 9, 5, 10, 11, 6, 12, 13, 7, 14, 15
How does this help us?

**Search complexity**

- Recursive subtrees of size at most $B \Rightarrow$ two contiguous blocks
- Two cache misses for each such subtree
- # of cache misses when searching down $\log n$ levels:
  \[
  \frac{(2 \log n)}{\log B} = 2 \log_B n
  \]

**Is this Divide and Conquer?**

- The layout is a kind of divide and conquer
- The algorithm is the usual tree-search algorithm
How does this help us?

Search complexity

- Recursive subtrees of size at most $B \Rightarrow$ two contiguous blocks
- Two cache misses for each such subtree
- $\#$ of cache misses when searching down $\log n$ levels:
  \[
  \frac{2 \log n}{\log B} = 2 \log_B n
  \]

Is this Divide and Conquer?

- The layout is a kind of divide and conquer
- The algorithm is the usual tree-search algorithm
Performance in practice

- from Kumar’03
- Linux/Itanium/2GB/g++ -O3/48 byte nodes
Performance in practice

- Cache Oblivious Search Trees via Binary Trees of Small Height, Brodal et. al.'02
- Linux, Pentium III 1GHz, 256KB cache, 1GB RAM, 4 byte nodes
Another dose of reality!

Take Home Message

One needs to be careful when putting theory into practice
Another dose of reality!

Take Home Message
One needs to be careful when putting theory into practice
### Some other data structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Funnels</td>
<td>Prokop’99</td>
</tr>
<tr>
<td>Dynamic Search Tree</td>
<td>Bender et. al.’00</td>
</tr>
<tr>
<td>Packed Memory Structure</td>
<td>Bender et. al.’00</td>
</tr>
<tr>
<td>Priority Queue</td>
<td>Arge et al.’02</td>
</tr>
</tbody>
</table>
Cache Oblivious Algorithms and Data Structures

- Abstract away the hardware parameters
  - Can handle varying cache specifics and multi-level memory hierarchies while attaining asymptotic efficiency
- A lot of CO algorithms have been developed lately
  - most are generalizations of previous external memory algorithms
  - main techniques: Divide and Conquer, Recursive Layout
- Their innate simplicity holds a lot of promise!!
- A number of issues not addressed by the theoretic model are critical for performance in practical settings