Seven Simple Experiments

Virtual Memory

Cache Oblivious

Further Subjects
The Experiments

Let $A$ be an array of size $n$

- **permute**: for $j \in [1..n]$ do: $i := \text{random}(0..j)$; swap($A[i], A[j]$);
- **random scan**: $\pi := \text{random permutation}$;
  for $i$ from 0 to $n - 1$ do: $S := S + A[\pi(i)]$;
- **$n$ binary searches** for random positions in sorted array $A$;
- **heapify**
- **heapsort**
- **quicksort** (STL introsort)
- **sequential scan**
Seven Simple Experiments

Virtual Memory

Cache Oblivious

Further Subjects

---

The diagram shows a graph with the x-axis labeled as "log(input size)" and the y-axis labeled as "running time/RAM complexity." The graph plots the performance of various algorithms under different access patterns.

- **Permute**
- **Random access**
- **Binsearch**
- **Heapsort**
- **Heapify**
- **Introsort**
- **Sequential access**

The legend indicates that the graph demonstrates logarithmic growth in running time and RAM complexity for different algorithms.
running time/RAM complexity

log(input size)
Straight line indicates logarithmic growth!
## Two Kinds of Programs

<table>
<thead>
<tr>
<th>extra logarithmic factor</th>
<th>no extra factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>permute</td>
<td>heapify</td>
</tr>
<tr>
<td>random access</td>
<td>introsort</td>
</tr>
<tr>
<td>binsearches</td>
<td>sequential access</td>
</tr>
<tr>
<td>heapsort</td>
<td></td>
</tr>
</tbody>
</table>
## Two Kinds of Programs

<table>
<thead>
<tr>
<th>extra logarithmic factor</th>
<th>no extra factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>permute</td>
<td>heapify</td>
</tr>
<tr>
<td>random access</td>
<td>introsort</td>
</tr>
<tr>
<td>binsearches</td>
<td>sequential access</td>
</tr>
<tr>
<td>heapsort</td>
<td></td>
</tr>
<tr>
<td>a lot of random access</td>
<td>little random access</td>
</tr>
<tr>
<td>low locality</td>
<td>high locality</td>
</tr>
</tbody>
</table>

Memory hierarchy is **NOT** the explanation.

5/20
## Two Kinds of Programs

<table>
<thead>
<tr>
<th>extra logarithmic factor</th>
<th>no extra factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>permute</td>
<td>heapify</td>
</tr>
<tr>
<td>random access</td>
<td>introsort</td>
</tr>
<tr>
<td>binsearches</td>
<td>sequential access</td>
</tr>
<tr>
<td>heapsort</td>
<td></td>
</tr>
</tbody>
</table>

| a lot of random access                       | little random access                                 |
| low locality                                 | high locality                                        |

- Memory hierarchy is **NOT** the explanation.
Virtual Memory

- Every program has its own (virtual) address space.
- OS maps them to a single (real) address space in RAM.

- Maintaining this layer of abstraction comes at a cost.

- VM is not paging! (although paging is a related subject)
Virtual Memory

- Every program has its own (virtual) address space.
- OS maps them to a single (real) address space in RAM.
- Maintaining this layer of abstraction comes at a cost.
- VM is not paging! (although paging is a related subject)
Virtual Memory

- Every program has its own (virtual) address space.
- OS maps them to a single (real) address space in RAM.
- Maintaining this layer of abstraction comes at a cost.
- VM is not paging! (although paging is a related subject)
The Process of Translation

- Offset has constant length, remaining bits belong to the index.
- Index is looked up in a binary prefix tree of logarithmic height.
The VAT Model
(Virtual Address Translation)

- Computations are performed by a RAM machine.
- All used addresses are automatically translated.
- Translation is logically transparent for a program.
- Translation is performed on an EM\(^1\) machine with translation cache (TC). Misses of the TC constitute the cost.

\(^1\)External Memory Model
VAT or no VAT

The Cheap Cases

- Sequential Access
  - Translation path rarely changes.
  - When changes, doesn’t change much.
  - With LRU strategy TC misses are rare, and their cost insignificant.

- Quicksort
  - Runs partition again and again, which is essentially a sequential scan
VAT or no VAT

The Expensive Cases

- Random Access
  - Consecutive accesses have substantially different translation paths (a.a.s.).
  - Almost every access has TC misses on a constant fraction of the path.
  - Each access has logarithmic cost.
VAT or no VAT

One More Cheap Case

- Heapify
  - Consecutive accesses have substantially different translation paths.
  - Careful analysis shows that with LRU strategy TC misses are rather rare.
Locality

- The classic model that makes use of data locality is the Cache Oblivious model.
- Cache Oblivious motivation is fundamentally different from VAT.
- We have a partial result.
- As long as “tall cache assumption” \( M = \Omega(B^2) \) is not required, Cache Oblivious algorithms are VAT efficient.
  - Observed by Naila Rahman in 2003
  - We explain why.
Locality

- The classic model that makes use of data locality is the Cache Oblivious model.
- Cache Oblivious motivation is fundamentally different from VAT.
- We have a partial result.
- As long as “tall cache assumption” ($M = \Omega(B^2)$) is not required, Cache Oblivious algorithms are VAT efficient.
  - Observed by Naila Rahman in 2003
  - We explain why.
Locality

- The classic model that makes use of data locality is the Cache Oblivious model.
- Cache Oblivious motivation is fundamentally different from VAT.
- We have a partial result.
- As long as “tall cache assumption” \( M = \Omega(B^2) \) is not required, Cache Oblivious algorithms are VAT efficient.
  - Observed by Naila Rahman in 2003
  - We explain why.
Locality

- The classic model that makes use of data locality is the Cache Oblivious model.
- Cache Oblivious motivation is fundamentally different from VAT.
- We have a partial result.
- As long as “tall cache assumption” \((M = \Omega(B^2))\) is not required, Cache Oblivious algorithms are VAT efficient.
  - Observed by Naila Rahman in 2003
  - We explain why.
Transformation

Shown for completeness, explanation in the paper.

Theorem

Let a cache oblivious algorithm have a running time $C(M; B; n)$, where $M$ is size of the cache, and $B$ is size of a block (as used in the cache oblivious approach). It causes at most

$$\sum_{i=0}^{d} C(a2^i P; 2^i P; n)$$

VAT misses, where $a := \lfloor W/d \rfloor$, while using optimal replacement strategy.

- $W$ — size of TC.
- $d$ — height of the translation tree
- $P$ — length of offset
## Examples of Transformations

<table>
<thead>
<tr>
<th>Problem</th>
<th>IO complexity</th>
<th>VAT complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential scan</td>
<td>( \frac{n}{B} )</td>
<td>( \frac{n}{P} )</td>
</tr>
<tr>
<td>quicksort</td>
<td>( (\frac{n}{B}) \log(\frac{n}{B}) )</td>
<td>( (\frac{n}{P}) \log(\frac{n}{P}) )</td>
</tr>
<tr>
<td>matrix multiplication(^2)</td>
<td>( \frac{n^3}{(M^{1/2}B)} )</td>
<td>( \frac{n^3}{(a^{1/2}P^{3/2})} )</td>
</tr>
<tr>
<td>search in a vEB(^3) tree</td>
<td>( \log_B n )</td>
<td>( \log_K n \ln \log_P n )</td>
</tr>
</tbody>
</table>

- \( P = 2 \) length of offset
- \( a = \text{size of TC/height of the translation tree} \)
- \( K = \text{arity of the translation tree} \)

\(^2\)in recursive layout, no Strassen
\(^3\)a tree with van Emde Boas layout
When should one use VAT?

- For huge data that fit in RAM.
Comments

- translation occurs twice in case of the virtual machines.
- similar mechanism drives disk accesses.
- translation cost can be reduced by enlarging the offset ... but it does not happen in practice.
Objections

- The translation tree has limited height (=4).
  - Usage of the levels grows continuously, not discrete.
  - There are reasons to use longer addresses.

- Why won’t they use hashing?
  - It is not online in RAM.
  - Hardware support is hard.
  - It is used in TLB$^4$.

---

$^4$associative memory is the hardware interpretation of hashing
Objections

- The translation tree has limited height (=4).
  - Usage of the levels grows continuously, not discrete.
  - There are reasons to use longer addresses.

- Why won’t they use hashing?
  - It is not online in RAM.
  - Hardware support is hard.
  - It is used in TLB\(^4\).

\(^4\)associative memory is the hardware interpretation of hashing
Summary

- Locality is important.
- Locality is essential for big data.
- VAT cost is an efficient measure of non-locality.
Thank You!