The Cost of Address Translation
ALENEX13, New Orleans, USA, January 7, 2013

Tomasz Jurkiewicz      Kurt Mehlhorn

Max Planck Institute for Informatics
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: $i := \text{random}(0..j)$; $\text{swap}(A[i], A[j])$;
- **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**
The diagram shows the running time/RAM complexity as a function of the logarithm of the input size. The x-axis represents the log of the input size, and the y-axis shows the running time/RAM complexity. The graph compares different algorithms, including permute, random access, binsearch, heapsort, heapify, introsort, and sequential access. The permute algorithm shows a notable logarithmic growth pattern, indicating that its complexity increases in a logarithmic manner with respect to the input size.
Seven Simple Experiments

Virtual Memory

Cache Oblivious

Further Subjects

**running time/RAM complexity**

log\((\text{input size})\)

---

The diagram shows the running time/RAM complexity for various algorithms as a function of the logarithm of the input size. The straight line indicates logarithmic growth.
Straight line indicates logarithmic growth!
### Two Kinds of Programs

| extra logarithmic factor | no extra factor |
|--------------------------|-----------------|
| permute                  | heapify         |
| random access            | introsort       |
| binsearches              | sequential access |
| heapsort                 |                 |
Two Kinds of Programs

| extra logarithmic factor | no extra factor |
|--------------------------|-----------------|
| permute                  | heapify         |
| random access            | introsort       |
| binsearches              | sequential access |
| heapsort                 |                 |
| a lot of random access   | little random access |
| low locality             | high locality   |
## Two Kinds of Programs

| extra logarithmic factor | no extra factor |
|--------------------------|-----------------|
| permute                  | heapify         |
| random access            | introsort       |
| binsearches              | sequential access |
| heapsort                 |                 |
| a lot of random access   | little random access |
| low locality             | high locality   |
| **Memory hierarchy is NOT the explanation.** |
Seven Simple Experiments

Virtual Memory

Cache Oblivious

Further Subjects

---

Running time/RAM complexity vs. log(input size) for various algorithms:
- permute
- random access
- binsearch
- heapsort
- heapify
- introsort
- sequential access

---

max planck institut
informatik

6/20
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 0...
- 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

\[
\sum_{i=0}^{d} C(a2^i P; 2^i P; n) \text{ VAT misses},
\]

where \( a := \left\lfloor W/d \right\rfloor \), while using optimal replacement strategy.

- \( W \) — size of TC.
- \( d \) — height of the translation tree
- \( P \) — \( 2^{\text{length of offset}} \)
### Examples of Transformations

| Problem                          | IO complexity       | VAT complexity       |
|---------------------------------|---------------------|----------------------|
| sequential scan                 | $n/B$               | $n/P$                |
| quicksort                       | $(n/B) \log(n/B)$   | $(n/P) \log(n/P)$   |
| matrix multiplication\(^2\)     | $n^3/(M^{1/2}B)$    | $n^3/(a^{1/2}P^{3/2})$ |
| search in a vEB\(^3\) tree      | $\log_B n$         | $\log_K n \ln \log_P n$ |

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

\(^2\)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!