The Cost of Address Translation

Tomasz Jurkiewicz   Kurt Mehlhorn   Pat Nicholson

Max Planck Institute for Informatics

full version of paper by TJ and KM available at arXiv
preliminary version presented at ALENEX 2013
full version to appear in Journal of Experimental Algorithms
follow-up paper by TJ, KM, and PN available at arXiv

Outline

- The Role of Models of Computation in Algorithmics
- Seven Simple Experiments
- Virtual Memory and Address Translation: The VAT Model
- Analysis of Some Algorithms
- Cache Replacement Strategies
- EM-Algorithms in the VAT Model
- Conclusions and Remarks

Properties of Good Models

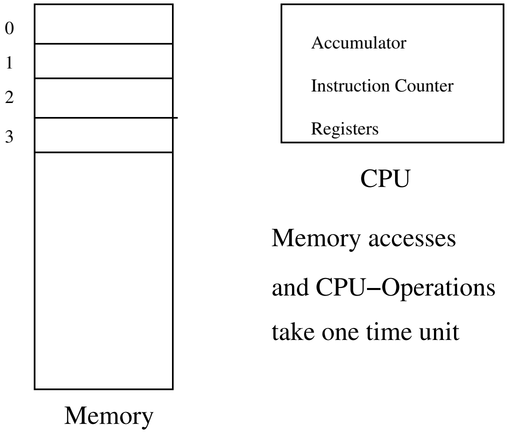
Simplicity: It will not be used otherwise.

Predictive Power: Otherwise, analysis of algs has no practical meaning

Popular Models:

- Random Access Machine (RAM) Model
- External Memory (EM) Model

The RAM Model



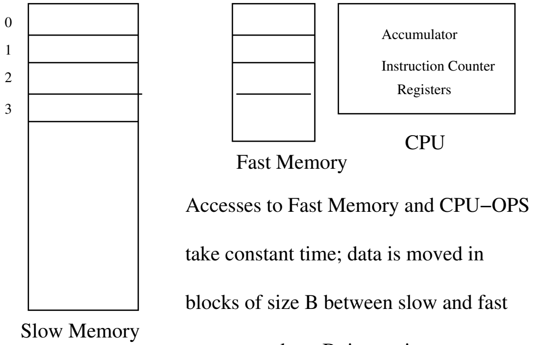
```
sum = 0; i = 0;
while (i < n)
sum = sum + A[i];
i = i + 1;
```

Scanning an array of length n takes

$$3n + O(1)$$

time units.

The External Memory (EM) Model



Accesses to Fast Memory and CPU−OPS

take constant time; data is moved in

blocks of size B between slow and fast

memory; takes cB time units.

c is a small constant, about 4.

```
sum = 0; i = 0;
while (i < n)
sum = sum + A[i];
i = i + 1;
```

Scanning an array of length n takes

$$(3 + c)n + O(1)$$

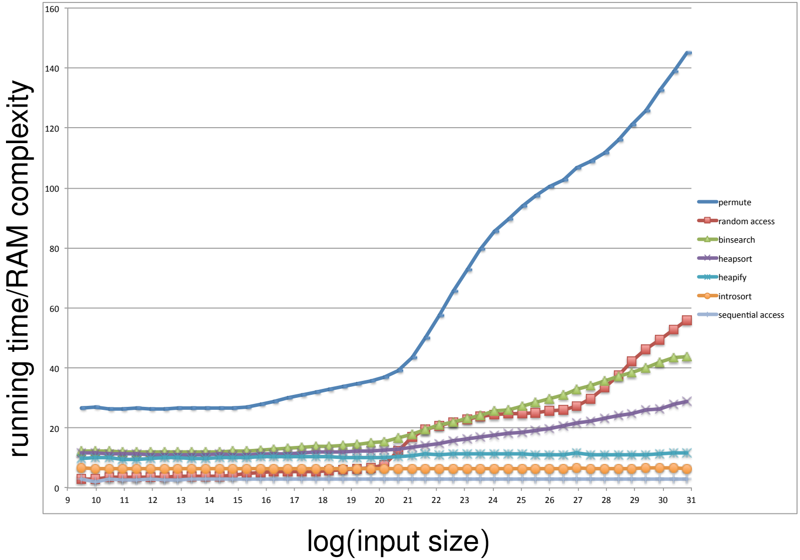
time units since we move n/B blocks of data for a cost of $(n/B) \cdot cB = cn$.

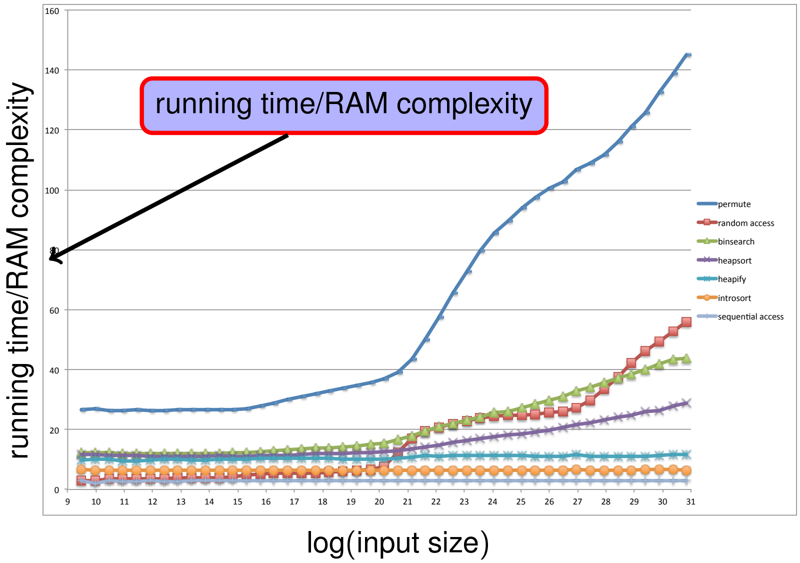
max planck institut informatik

The Experiments

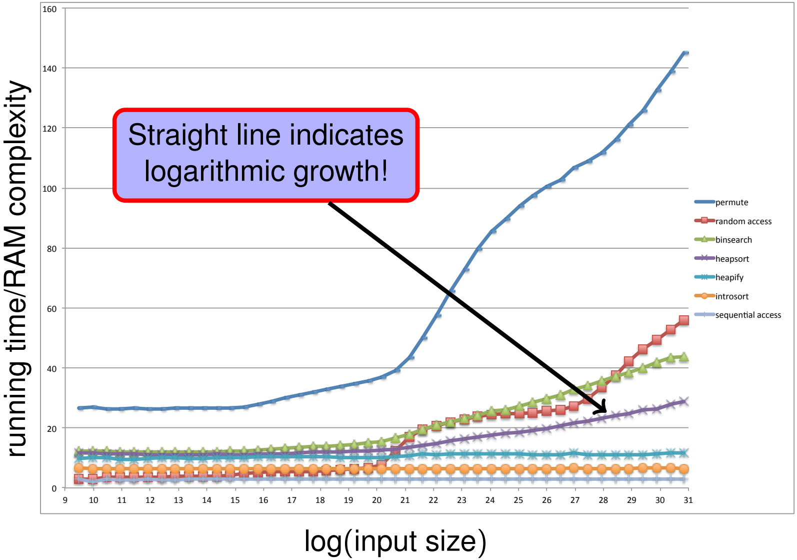
Let A be an array of size n

- permute: for $j \in [1..n-1]$ do: $i :=$ random$(0..j)$; swap$(A[i], A[j])$;
- random scan: $\pi :=$ 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

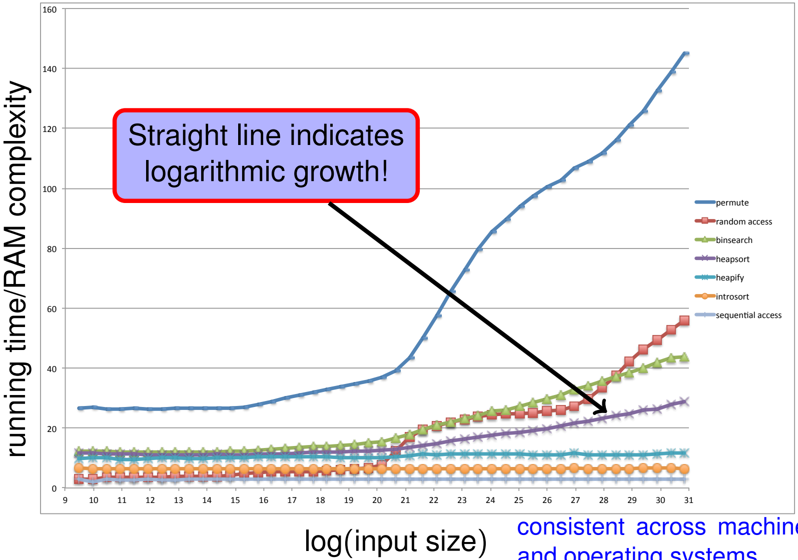

running time/RAM complexity
log(input size)
160
140
120
100
80
60
40
20
0
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
permute
random access
binsearch
heapsort
heapify
introsort
sequential access



running time/RAM complexity
160
140
120
100
80
60
40
20
0
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
permute
random access
binsearch
heapsort
heapify
introsort
sequential access
running time/RAM complexity
log(input size)



Straight line indicates logarithmic growth!
running time/RAM complexity
log(input size)
160
140
120
100
80
60
40
20
0
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
permute
random access
binsearch
heapsort
heapify
introsort
sequential access



consistent across machines and operating systems

Two Kinds of Programs

extra logarithmic factor	no extra factor
permute	heapify
random access	quicksort
binsearches	sequential access
heapsort	

Two Kinds of Programs

extra logarithmic factor	no extra factor
permute	heapify
random access	quicksort
binsearches	sequential access
heapsort	
lots of random access	little random access
low locality	high locality

Two Kinds of Programs

extra logarithmic factor	no extra factor
permute random access binsearches heapsort	heapify quicksort sequential access
lots of random access low locality	little random access high locality

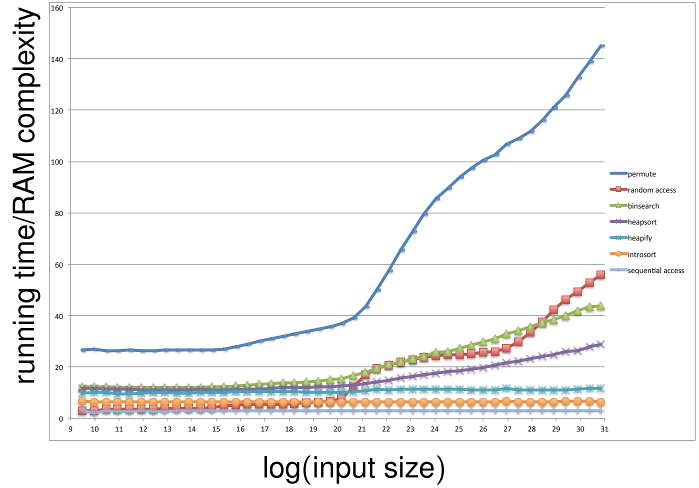
Isn't it just the memory hierarchy?

NO

Memory Hierarchy is not the Explanation

- in the EM-model, linear scan with step size larger than block size has the same cost as random scan.
- but measurements show that the running time of the former program is linear and that the second program suffers under a logarithmic slowdown.
- argument is due to Jirki Kataijanen; our argument was more complex

What is the Explanation?



Virtual Memory

- Our programs do not run on a naked machine, but under an OS
- Every program has its own (virtual) address space $0, \ldots$
- 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

- Our programs do not run on a naked machine, but under an OS
- Every program has its own (virtual) address space $0, \ldots$
- 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

- Our programs do not run on a naked machine, but under an OS
- Every program has its own (virtual) address space $0, \ldots$
- 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)

Address Translation

Virtual address (say 64 bit) consists of an offset (12 bits) and chunks l_1 to l_d of fixed size (9 bits each).

Virtual address = 16 Bits l_1 l_2 l_3 l_4 Offset

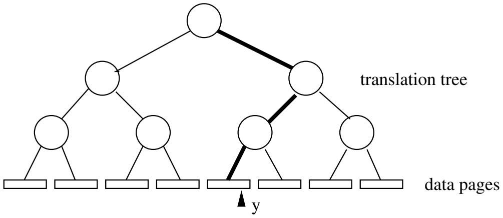
Translation from virtual to physical address

= tree walk in the translation tree.

Translation tree is a tree of depth d ($d = 4$ in 64-bit machines). One starts in the root, goes to child l_1, then to child l_2 of child, ..., then to child l_4 of .... .

Now one is in a page containing data. The offset selects the byte in the data page.

Address Translation

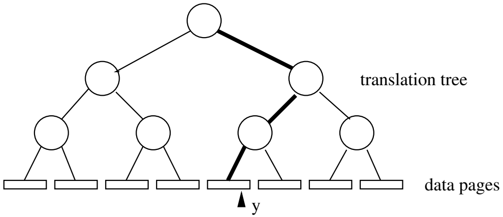


Translation tree has fan-out K and depth d; here $K = 2$ and $d = 3$. Translation path for the virtual index 100 is shown. The offset y selects a cell in the physical page with virtual index 100.

Nodes and data pages are stored in memory. Only nodes and data pages in fast memory can be accessed directly. Otherwise, fetch from slow memory. Each such fetch is a cache fault.

EM-model only counts cache faults for data pages; VAT-model counts also cache faults for nodes.

Address Translation

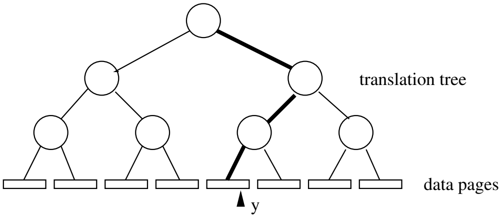


Translation tree has fan-out K and depth d; here $K = 2$ and $d = 3$. Translation path for the virtual index 100 is shown. The offset y selects a cell in the physical page with virtual index 100.

Nodes and data pages are stored in memory. Only nodes and data pages in fast memory can be accessed directly. Otherwise, fetch from slow memory. Each such fetch is a cache fault.

EM-model only counts cache faults for data pages; VAT-model counts also cache faults for nodes.

Address Translation



Translation tree has fan-out K and depth d; here $K = 2$ and $d = 3$. Translation path for the virtual index 100 is shown. The offset y selects a cell in the physical page with virtual index 100.

Nodes and data pages are stored in memory. Only nodes and data pages in fast memory can be accessed directly. Otherwise, fetch from slow memory. Each such fetch is a cache fault.

EM-model only counts cache faults for data pages; VAT-model counts also cache faults for nodes.

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 a machine with translation cache (TC). Misses of the TC constitute the cost.
- Translation cache has a fixed size, say W nodes.
- EM-model counts only cache misses at the leaf level of the translation tree.

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 a machine with translation cache (TC). Misses of the TC constitute the cost.
- Translation cache has a fixed size, say W nodes.
- EM-model counts only cache misses at the leaf level of the translation tree.

We analysed the algorithms from the experimental study and several others in the model. Proved (almost) matching upper and lower bounds. Bounds are in good agreement with the measurements.

Analysis of Algorithms: Cheap Cases

- Sequential Access
  - Translation path rarely changes.
  - When it changes, it doesn't change much.
  - With LRU strategy TC misses are rare, and their cost insignificant.
- Quicksort: partition is essentially a sequential scan
- Heapify
  - consecutive accesses may have substantially different translation paths.
  - careful analysis shows that with LRU strategy TC misses are rather rare.
  - $\log n$ parallel linear scans, one starting at n and one starting at $n/2$, one starting at $n/4$, on starting at $n/8$, ..., and all going backwards.

Analysis of Algorithms: 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.
- Heapsort

A Technical Remark: Normal Form for TC Content

- for lower bounds it is conventient to assume a normal form content of the translation cache, namely
- TC-cache always contains an initial segment of the tree, i.e., if a node is cached, its parent is also cached.
- for OPT and LRU this can be enforced at no cost (no additional cache misses) by increasing cache by d nodes
- $d =$ depth of translation tree
- normal form simplifies lower bound arguments

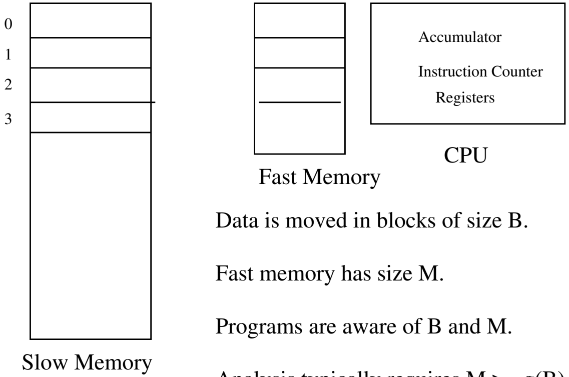
Implications?

- Penalty for random accesses is even higher than EM-model predicts; avoid under all cirumstances.
- There is an extensive literature on good algorithms for the EM-model. Are these algorithms any good for the VAT-model or do we have to redo the theory?
  - Some are, e.g., linear scan and merge sort and quicksort.
  - The fact that nobody noticed the VAT-penalty before us suggests that the first alternative holds.
  - Tomasz and I could not prove a general positive theorem.
  - Together with Pat Nicholson, we recently obtained a general positive result.

Implications?

- Penalty for random accesses is even higher than EM-model predicts; avoid under all cirumstances.
- There is an extensive literature on good algorithms for the EM-model. Are these algorithms any good for the VAT-model or do we have to redo the theory?
  - Some are, e.g., linear scan and merge sort and quicksort.
  - The fact that nobody noticed the VAT-penalty before us suggests that the first alternative holds.
  - Tomasz and I could not prove a general positive theorem.
  - Together with Pat Nicholson, we recently obtained a general positive result.

EM Algs: A Typical Result



Data is moved in blocks of size B.

Fast memory has size M.

Programs are aware of B and M.

Analysis typically requires M >= g(B)

Funnel sort (Frigo et al)

On an EM-machine with cache size $\widetilde{M}$ and block size $\widetilde{B}$, it sorts n items with

$$O\left(\frac{n}{\widetilde{B}}\left\lceil\frac{\log n/\widetilde{M}}{\log \widetilde{M}/\widetilde{B}}\right\rceil\right)$$

cache faults provided that $\widetilde{M} \geq \widetilde{B}^2$.

and its Transfer to VAT

Funnel sort (Frigo et al): On an EM-machine with cache size $\widetilde{M}$ and block size $\widetilde{B}$, it sorts n items with $O\left(\frac{n}{\widetilde{B}}\left\lceil\frac{\log n/\widetilde{M}}{\log \widetilde{M}/\widetilde{B}}\right\rceil\right)$ cache faults provided that $\widetilde{M} \geq \widetilde{B}^2$.

Theorem

On a VAT-machine with cache size M and block size B, Funnel sort sorts n items with at most

$$O\left(\frac{4n}{B}\left\lceil\frac{\log 4n/M}{\log M/(4dB)}\right\rceil\right)$$

cache faults. This assumes $(B\log_K(2n/B))^2 \leq M/4$.

Note $M/(4dB) \geq (M/B)^{1/2}$ for realistic values of M, B, K, and n.

Theorem

Consider an EM-algorithm with $C(\widetilde{M}, \widetilde{B}, n)$ cache faults provided that $\widetilde{M} \geq g(\widetilde{B})$. Let $d = \log_K(n/B)$. The program can be made to run on a VAT-machine with cache size M and page size B with at most $4dC(M/4, dB, n)$ cache faults provided that $M \geq 4g(dB)$.

We execute EM-algorithm with $\widetilde{M} = M/4$ and $\widetilde{B} = dB$ on a VAT-machine with block size B and cache size M. Use $M/4$ cells for data and $3M/4$ for nodes. Since

$$\widetilde{M} = M/4 \geq g(dB) = g(\widetilde{B}),$$

the number of EM-cache faults is at most $C(\widetilde{M}, \widetilde{B}, n)$.
Whenever, the EM-alg moves a block, the VAT-machine moves the corresponding d data blocks and all internal nodes of the translation paths to these d data pages. Number of internal nodes is bounded by $2d + \sum_{i \geq 1} d/K^i \leq 2d + d/(K-1) \leq 3d$.
Thus a translation cache of size $3M/4$ suffices. For every cache fault of the EM-model, the VAT-machine incurs $4d$ cache faults.

Comments and Objections

- translation occurs twice in case of the virtual machines.
- translation cost can be reduced by enlarging the offset, operating systems offer this choice (somewhat inconveniently), however it is rarely used.
- The translation tree has limited height (= 4) and hence . . .
  - true, but we need models that allow us to understand measurements
  - the VAT-model explains the measurements (to a large extent)
  - true, but $\log_{M/B} n/B$ is EM-model is never more than 4

Summary

- Locality is crucial for big data. VAT-model makes this even more evident than EM-model.
- Very non-local programs suffer a logarithmic penalty in VAT-model; this factor shows in the measurements.
- Transfer theorem shows that good EM-algorithms are usually also good in VAT-model
  - No need to redo the theory.
  - Maybe explains, why nobody discovered the effect before.
- suggest to spend 60 minutes on the effect of virtual memory in algorithm engineering class

Thank You!