I/O Lower Bounds for Sorting and Matrix Problems

Jeff Vitter

Duke University

Department of Computer Science

Center for Geometric & Biological Computing

http://www.cs.duke.edu/CGBC/

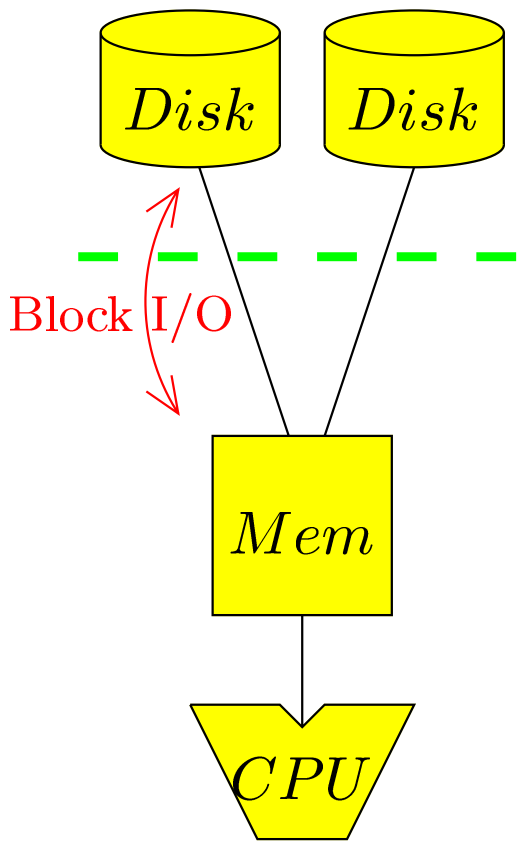
EEF Summer School—July 2002

Outline

- ⋆ Fundamental Techniques for batched problems.
  - Merge sort, distribution sort.
- ⋆ Techniques for solving batched geometric problems.
  - Distribution sweeping, batched filtering, randomized incremental construction.
  - Red-blue orthogonal rectangle intersection, convex hull, range search, nearest neighbors.
  - Empirical results (via TPIE programming environment).
- $\Longrightarrow$ Fundamental lower bounds.
  - Sorting, permuting, FFT, matrix transposition, bundle sort.
  - Dynamic memory allocation
  - Hierarchical memory.
- ⋆ Parallel disks.
  - Load balancing among disks is key issue.
  - Duality: reading (prefetching) $\longleftrightarrow$ writing, merging $\longleftrightarrow$ distribution

Review of Parallel Disk Model

[Aggarwal & Vitter 88], [Vitter & Shriver 90, 94], ...



N = problem data size.

M = size of internal memory.

B = size of disk block.

D = number of independent disks.

P = number of CPUs.

Q = number of queries.

Z = problem output size.

Notational convenience (in units of blocks):

$$n = \frac{N}{B}, \quad m = \frac{M}{B}, \quad q = \frac{Q}{B}, \quad z = \frac{Z}{B}.$$

Fundamental I/O Bounds (with $D = 1$ disk)

- Batched problems [AV88], [VS90], [VS94]:
  - Scanning (touch problem): $\Theta\left(\frac{N}{B}\right) = \Theta(n)$
  - Sorting:
    $$\Theta\left(\frac{N}{B}\frac{\log \frac{N}{B}}{\log \frac{M}{B}}\right) = \Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = \Theta\left(n \log_m n\right)$$
  - Permuting: $\Theta\left(\min\left\{N,\ n\log_m n\right\}\right)$
- For other problems [CGGTVV95], [AKL95], ...
  - Graph problems $\asymp$ Permutation
  - Computational Geometry $\asymp$ Sorting
- Online problems:
  - Searching and Querying: $\Theta\left(\log_B N + \frac{Z}{B}\right) = \Theta(\log_B N + z)$
- What if there are D parallel disks ???

Fundamental I/O Bounds (with $D = 1$ disk)

- Batched problems [AV88], [VS90], [VS94]:
  - Scanning (touch problem): $\Theta\left(\dfrac{N}{B}\right) = \Theta(n)$
  - Sorting:

$$\Theta\left(\frac{N}{B}\frac{\log \frac{N}{B}}{\log \frac{M}{B}}\right) = \Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = \Theta\left(n \log_m n\right)$$

  - Permuting: $\Theta\left(\min\left\{N,\ n\log_m n\right\}\right)$
- For other problems [CGGTVV95], [AKL95], ...
  - Graph problems $\asymp$ Permutation
  - Computational Geometry $\asymp$ Sorting
- Online problems:
  - Searching and Querying: $\Theta\left(\log_B N + \frac{Z}{B}\right) = \Theta(\log_B N + z)$
- D parallel disks: *Saves factor of D for batched problems, Replace B by DB in online problems (disk striping).*

I/O Lower Bound for Permuting

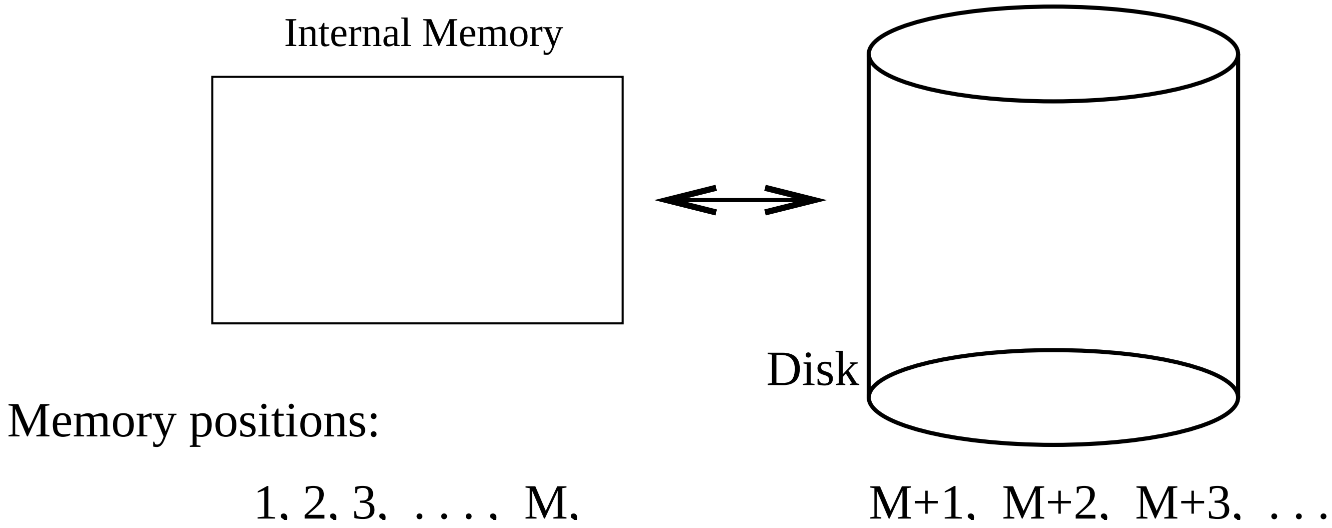
Permuting problem: Given N distinct items from $\{1, 2, \ldots, N\}$, rearrange the N items into sorted order.

- ☆ We will show the lower bound that permuting requires $\Omega(\min\{N,\ n \log_m n\})$ I/Os.
- ☆ Typically the min term is $n \log_m n$.
- ☆ Permuting is a special case of sorting.
- ☆ I/O lower bound also applies to sorting. *It is based only upon routing considerations, since the order is already known.*
- ☆ For the pathological case when $N < n \log_m n$, we can show that sorting requires $\Omega(n \log_m n)$ I/Os in comparison model.
- ☆ In the RAM model, permutation takes only $O(N)$ time. But in I/O model, it (and most interesting problems) require sorting complexity (except for pathological case)!

I/O Lower Bound for Permuting

Goal: See how many I/O steps T are needed so that any of the $N!$ permutations of the N items can be realized.

We say that a permutation is realizable if it appears in extended memory in the required order.



Tactic: Determine how much the tth I/O step can increase the number of possible realizable permutations.

I/O Lower Bound for Permuting

Assumption: the N items to permute are indivisible.

realizable permutations after tth read I/O

$$= \begin{cases} \binom{M}{B} \times (\text{\# realizable permutations after } (t-1)\text{st I/O}) & \text{if block was previously accessed} \\ B! \times \binom{M}{B} \times (\text{\# realizable permutations after } (t-1)\text{st I/O}) & \text{if this is first access to block} \end{cases}$$

There are N/B blocks initially unaccessed.

choices for block accessed in tth I/O $= \left(\frac{N}{B} + t\right) \leq N(1+\log N)$.

Number T of required I/Os for Permuting

$$(B!)^{N/B}\left(\binom{M}{B}N(1+\log N)\right)^{T} \geq N!$$

Taking logs and applying Stirling's approximation:

$$\begin{aligned}
\frac{N}{B}\log B! + T\left(\log\binom{M}{B} + \log N\right) &= \Omega(\log N!) \\
\frac{N}{B}(B\log B) + T\left(B\log\frac{M}{B} + \log N\right) &= N\log N \\
T\left(B\log\frac{M}{B} + \log N\right) &= N\log N - N\log B \\
&= N\log\frac{N}{B} \\
T &= \Omega\left(\min\left\{N,\ \frac{N}{B}\frac{\log(N/B)}{\log(M/B)}\right\}\right) \\
&= \Omega\left(\min\left\{N,\ n\log_m n\right\}\right)
\end{aligned}$$

More Refined Analysis to Get Leading Coefficient

Assuming that M/B is an increasing function,

I/Os required to sort or permute n items is at least

$$\frac{2N}{D}\,\frac{\log n}{B\log m + 2\log N} \sim \begin{cases} \dfrac{2n}{D}\log_m n & \text{if } B\log m = \omega(\log N); \\ \dfrac{N}{D} & \text{if } B\log m = o(\log N). \end{cases}$$

- ☆ WLOG, we can assume that each I/O is *simple*: at any time there is only one copy of each item—on disk or in memory. *No copying!*
- ☆ We need to do enough write I/Os to keep up with read I/Os.
- ☆ The problem is that read I/Os may have fewer than B items.
- ☆ Let b_i = # items read in ith read I/O.
- ☆ Let R = # read I/Os, and W = # write I/Os.
- ☆ $W \geq \frac{1}{B}\left(\sum_{1\leq i\leq R} b_i\right)$.
- ☆ Each read I/O boosts # realizable permutations by a factor of $N(1+\log N)\binom{M}{b_i}$.
- ☆ Each write I/O boosts # realizable permutations by a factor of $N(1+\log N)$.

More Refined Analysis to Get Leading Coefficient

- $\Longrightarrow \big(N(1+\log N)\big)^{R+W} \displaystyle\prod_{1\le i\le R} \binom{M}{b_i} \ge \frac{N!}{(B!)^{N/B}}$
- Let $\widetilde{b}$ be the average value of b_i.
- By convexity argument, LHS is maximized by setting each $b_i := \widetilde{b}$.
- $W \ge \frac{1}{B}\left(\sum_{1\le i\le R} b_i\right) = \frac{1}{B}(R\widetilde{b}) \Longrightarrow R \le (R+W)/(1+\widetilde{b}/B)$.
- $\big(N(1+\log N)\big)^{R+W} \dbinom{M}{\widetilde{b}}^{(R+W)/(1+\widetilde{b}/B)} \ge \dfrac{N!}{(B!)^{N/B}}$.
- Maximize LHS by setting $\widetilde{b} = B$, so we get

$$\big(N(1+\log N)\big)^{R+W} \binom{M}{B}^{(R+W)/2} \ge \frac{N!}{(B!)^{N/B}}.$$

which gives desired lower bound on the total number $R+W$ of I/Os.

Converting from Row-Major to Column-Major Order

Theorem 3.3 The number of I/Os required to transpose a $p \times q$ matrix stored in row-major order is

$$\Theta\left(\frac{n \log \min\{M, \min\{p, q\}, n\}}{\log m}\right).$$

NOTE: Transposition is a special case of permutation. It can be done in $O(n)$ I/Os when $B^2 \leq M$.

We define $f(x) = \begin{cases} x \log x & \text{if } x > 0; \\ 0 & \text{if } x = 0. \end{cases}$

Let $x_{i,k}$ = number of steps in kth block that should be in ith block.
Let y_i = number of steps in internal memory that should be in ith block.
Define togetherness function as

$$C_k(t) = \sum_{1 \leq i \leq n} f(x_{i,k}) \qquad C_M(t) = \sum_{1 \leq i \leq n} f(y_i)$$

Potential

$$\text{Potential}(t) = C_M(t) + \sum_{k \geq 1} C_k(t)$$

$$\text{Potential}(T) = N \log B$$

$$\text{Potential}(0) = \begin{cases} 0 & \text{if } B < \min\{p,q\}; \\ N \log \frac{B}{\min\{p,q\}} & \text{if } \min\{p,q\} \leq B \leq \max\{p,q\}; \\ N \log \frac{B^2}{N} & \text{if } \max\{p,q\} < B. \end{cases}$$

We can show that:

$$\begin{aligned} \nabla\text{Potential}(t) &= C_M(t) - C_M(t-1) - C_k(t-1) \\ &= O(B \log m) \end{aligned}$$

$$\Longrightarrow \text{Lower bound} = \Omega\left(\frac{\text{Potential}(T) - \text{Potential}(0)}{B \log m}\right).$$

Bundle Sorting [MSV]

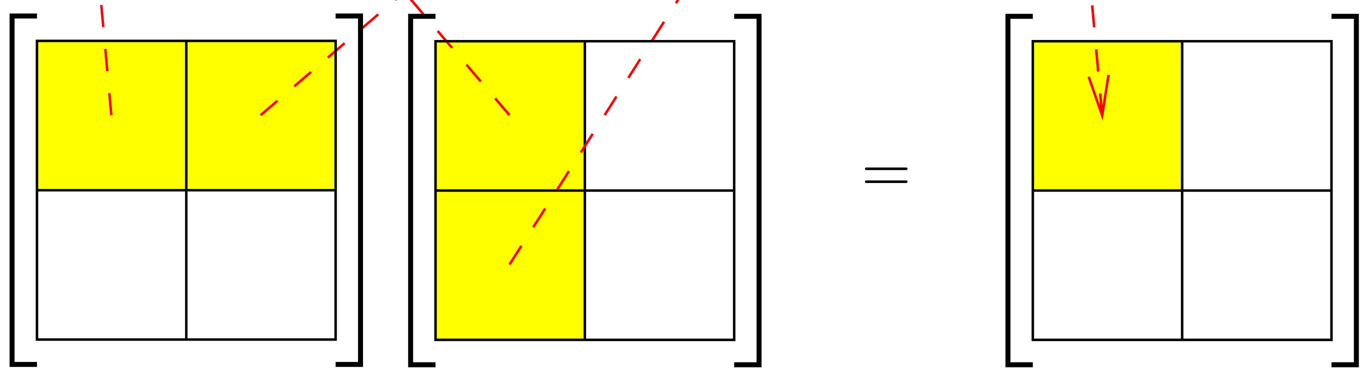
Combination of permutation approach and matrix transposition approach gives us a lower bound on the problem of *bundle sorting*, in which there are only K distinct key values (but secondary info of each record is different):

$$\#\ \text{I/Os} = \Theta\left(n \log_m \frac{K}{B}\right).$$

This work also noticed that sorting can be done in-place, at expense of having blocks not be contigous in each run or bucket.

Recursive Matrix Multiplication

$$A_{1,1} \; B_{1,1} \; + \; A_{1,2} \; B_{2,1} \; = \; C_{1,1}$$

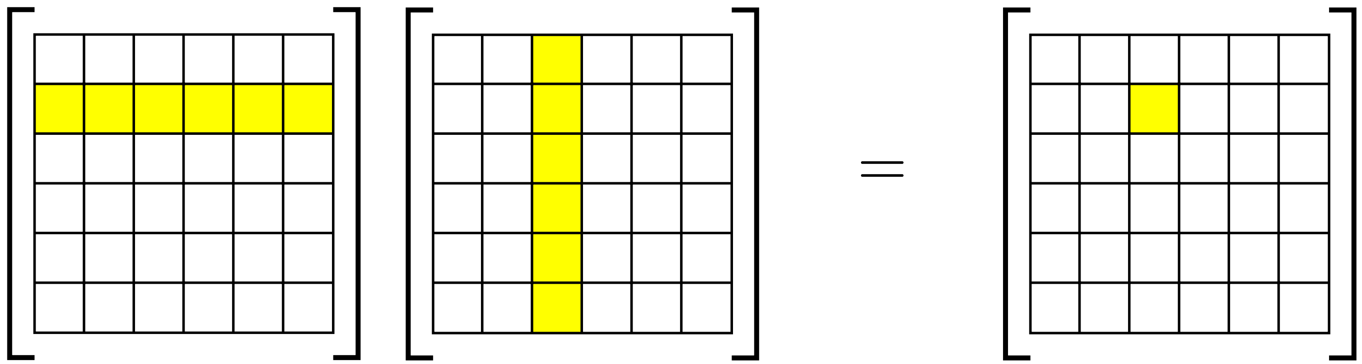


☆ I/O complexity for $K \times K$ matrices:

$$T(K) = 8T\left(\frac{K}{2}\right) + 6\frac{K^2}{B} \quad (1)$$

$$= 9\sqrt{3}\frac{K^3}{B\sqrt{M}}. \quad (2)$$

Iterative Matrix Multiplication



- ☆ Rather than do partitioning at each level of recursion, do the partitioning all at once, up front.
- ☆ Preprocess by reblocking row-major $K \times K$ input matrices into blocks of size $\sqrt{M/3} \times \sqrt{M/3}$.
- ☆ Do matrix multiplication on blocks.
- ☆ Reblock output into row-major order.

I/O Complexity for Iterative Matrix Multiplication

I/O complexity for multiplying two $K \times K$ matrices:

$$\begin{aligned} T(K) &= \left(\frac{K}{\sqrt{M/3}}\right)^3 \frac{M}{B} + 6\frac{K^2}{B}\left(1 + \log_{m/2}\frac{\sqrt{3}K}{\sqrt{M}}\right) && (3) \\ &\approx 3\sqrt{3}\frac{K^3}{B\sqrt{M}} && (4) \end{aligned}$$

3 times faster when the reblocking is done all up front!

The Need for Memory-Adaptive EM Algorithms.

☆ Traditional EM algorithms assume *fixed memory* allocation.

☆ Problem:

- OS/DBMS can *dynamically* change memory allocation.
- EM applications exhibit *thrashing*.

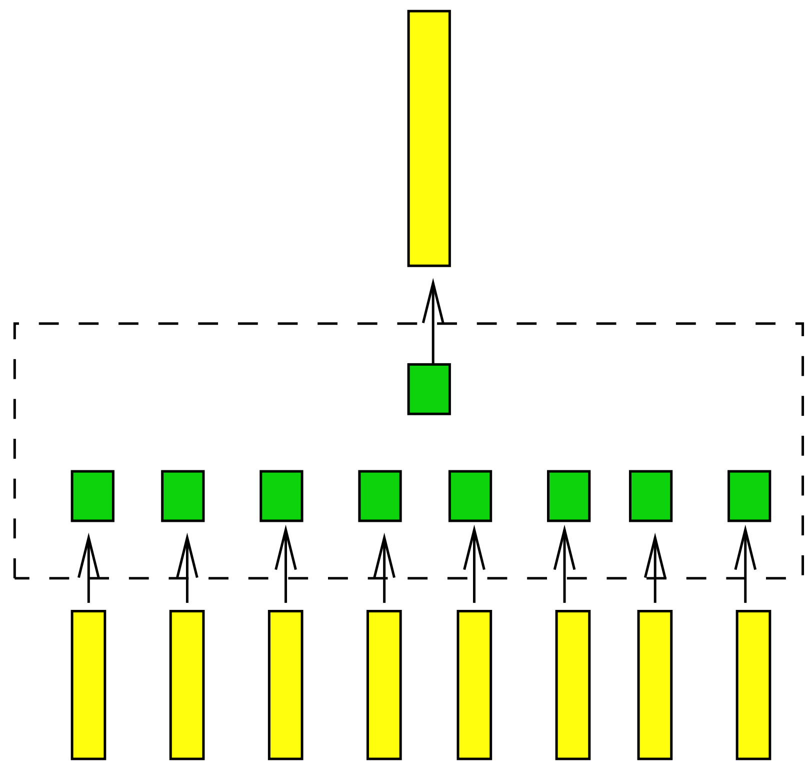
☆ Solution:

EM algorithms that *adapt online* to memory fluctuations.

☆ All prior work has been exclusively empirical:

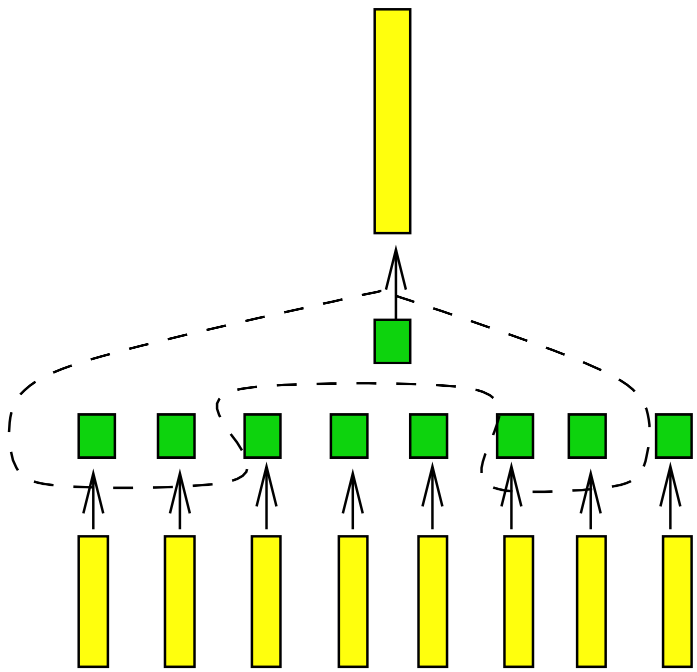
- Memory-Adaptive Hash Join (Zeller& Gray, Pang et al.)
- Pang et al., 1995: Non-optimal memory-adaptive sort.
- Zhang and Larson, 1997: Memory-adaptive sort, works only for very restricted kinds of fluctuations.

Why Traditional EM Algorithms Thrash.



Merging 8 runs using 9 internal memory blocks

Why Traditional EM Algorithms Thrash.



Merging 8 runs using 5 internal memory blocks:
Leading blocks of 4 runs are out of memory

- ⋆ If m drops to less than 8 but merge-order remains 8, worst case cost is one I/O per element output by merge.
- ⋆ Solution: Reorganize computation; ie, change merge-order in response to change in m.

Dynamic Memory Environment

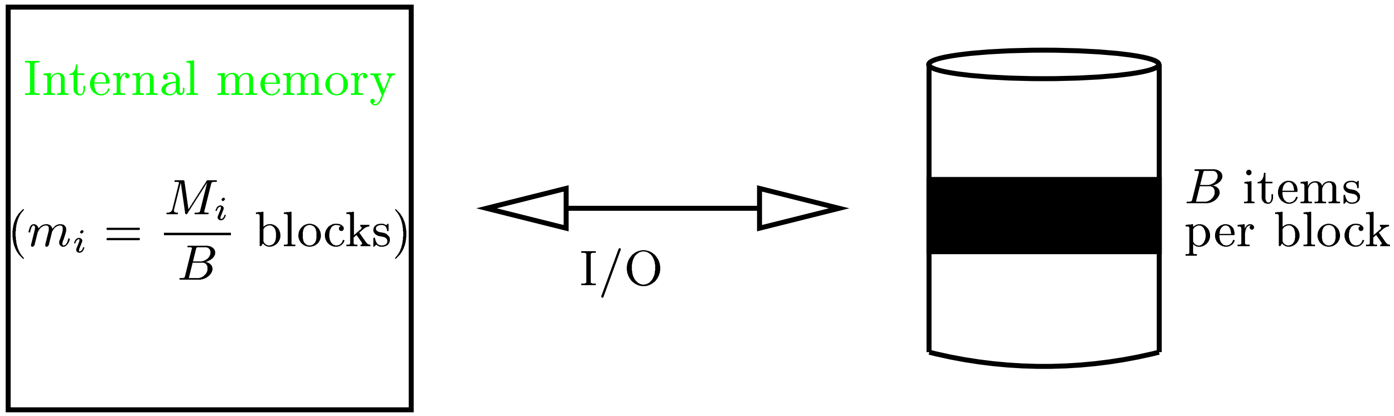
- ☆ EM algorithm is allocated m memory blocks by the OS/DBMS for an unspecified amount of time.
- ☆ When OS/DBMS wants to change the allocation of m, it first allows EM algorithm to carry out m I/Os (“Reaction time”). Then it changes m.
- ☆ We use a simplified “constant factor approximation” of this model.

Simple Model for Memory-Adaptive EM Algorithms

- ☆ EM algorithm $\mathcal{A}$ is allocated memory in an allocation sequence $\sigma = m_1, m_2, m_3, \ldots$ of *allocation phases.*
- ☆ OS/DBMS determines σ in an online adversarial manner.
- ☆ ith phase: Algorithm owns m_i blocks of memory for $2m_i$ I/Os.
- ☆ EM algorithm must adapt to allocation sequence.
- ☆ Suppose that $\mathcal{A}$ solves problem $\mathcal{P}$ during σ.
- ☆ $\mathcal{A}$ is dynamically optimal for $\mathcal{P}$ iff
  - No other algorithm $\mathcal{A}'$ can solve problem $\mathcal{P}$ more than a constant number of times during σ.

Dynamic Memory Lower Bound for Sorting

ith phase:



Use comparison model:

$$\begin{array}{c}\#\text{ possible outcomes}\\ \text{to comparisons per I/O}\end{array} = \begin{cases} B! \times \dbinom{M_i}{B} & \text{reading unread block.} \\ \dbinom{M_i}{B} & \text{reading dirty block.} \end{cases}$$

$$(B!)^{N/B} \prod_i \binom{M_i}{B}^{2m_i} \geq N! \quad \Longrightarrow \quad \sum_i 2m_i \log m_i = \Omega(n \log n).$$

Resource Consumption of Sorting

Sorting algorithm completes in ℓ phases

$$\Longrightarrow \quad \sum_{i=1}^{\ell} 2m_i \log m_i = \Omega(n \log n).$$

- ☆ Resource Consumption of an I/O in phase i is

$$\log m_i$$

- ☆ Algorithm is *dynamically optimal* iff

Total Resource Consumption (RC) $= O(n \log n)$.

A Framework for Memory-Adaptive Mergesort

☆ Run Formation

- Phase $i \quad \Longrightarrow \quad$ Generate a run of length m_i blocks.
- Number of runs in $\mathcal{Q}$ is $n_0 \leq n$. (Very often, $n_0 \ll n$.)
- Total Resource Consumption

$$\begin{aligned} \mathrm{RC}_{\mathrm{run\_formation}} &= O(\#\mathrm{I/Os} \times \text{Max cost of each I/O}) \\ &= O(n \log m_{\max}) \end{aligned}$$

☆ Merging Stage

- Memory-adaptive merging routine $\mathcal{M}$.
- Repeat: Merge R runs from $\mathcal{Q}$, append output run to $\mathcal{Q}$.

Resource Consumption Requirement for Merging

$$\begin{aligned}
\mathrm{RC}_{\mathrm{sort}} &= O\left(\mathrm{RC}_{\mathrm{run_formation}} + \frac{\log n_0}{\log R}\mathrm{RC}_{\mathrm{pass}}\right) \\
&= O\left(n \log m_{\max} + \frac{\log n_0}{\log R}\mathrm{RC}_{\mathrm{pass}}\right)
\end{aligned}$$

For dynamic optimality,

- ☆ $\mathrm{RC}_{\mathrm{pass}} = O(n \log R)$.
- ☆ $R = \Omega(m_{\max}^c)$.

Aspects

- ☆ Various external memory data structures and techniques are required for the scheme to work efficiently.
- ☆ Lower Bounds for problems related to sorting and matrix multiplication (and related problems).
- ☆ Sorting algorithm was used to get dynamically optimal algorithms for permuting, permutation networks, FFT.
- ☆ Dynamically Optimal memory-adaptive version of a buffer tree.
- ☆ Techniques applicable via sorting and buffer trees to many other applications.
- ☆ Dynamically optimal matrix multiplication algorithm.

Conclusions and Open Problems

☆ *Répertoire of useful paradigms (distribution, merging, distribution sweeping, persistence, parallel simulation, B-trees, external interval tree, external priority search tree) for important problems.*

- Worst-case optimality requires overhead.
- Simpler versions are practical!
- Building blocks for external data structures

☆ Lots of interesting open problems!

- Lower bounds without indivisibility assumption.
- [Adler] showed that removing the indivisibility assumption for an artificial problems related to transposition can lead to faster algorithms.
- New models: hierarchical memory, oblivious caching, dynamic memory allocation, MEMS, optical storage, ...

Conclusions and Open Problems

- TPIE, see http://www.cs.duke.edu/TPIE/
- Handling many disks, large merge orders, many partition elements, large fanouts. (Don't use square root trick.)
- String processing, molecular databases.
- ...