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).
  ➞ 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) ↔ writing, merging ↔ distribution
\[ \frac{B}{Z} = z, \quad \frac{B}{\theta} = b, \quad \frac{B}{W} = m, \quad \frac{B}{N} = n \]

Notational convenience (in units of blocks):

- Problem output size: \( Z \)
- Number of queries: \( \theta \)
- Number of CPUs: \( p \)
- Number of independent disks: \( d \)
- Size of disk block: \( B \)
- Size of internal memory: \( W \)
- Problem data size: \( N \)

[Aggarwal & Vitter 88, Vitter & Shriver 90, 94]
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:
    \[
    \Theta \left( \frac{N \log \frac{N}{B}}{B \log \frac{M}{B}} \right) = \Theta \left( \frac{N}{B} \log_{M/B} \frac{N}{B} \right) = \Theta(n \log_m n)
    \]
  - Permuting: $\Theta \left( \min \{ N, \ n \log_m n \} \right)$

- For other problems [CGGTVVV95], [AKL95], …
  - Graph problems $\sim$ Permutation
  - Computational Geometry $\sim$ 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): $\Theta \left( \frac{N}{B} \right) = \Theta(n)$
  - Sorting:
    $$\Theta \left( \frac{N \log \frac{N}{B}}{B \log \frac{M}{B}} \right) = \Theta \left( \frac{N}{B} \log_{M/B} \frac{N}{B} \right) = \Theta (n \log_m n)$$
  - Permuting: $\Theta (\min \{N, \ n \log_m n\})$

☆ For other problems [CGGTVV95], [AKL95], …
  - Graph problems $\preceq$ Permutation
  - Computational Geometry $\preceq$ 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.

Memory positions:

1, 2, 3, \ldots, M, \quad \text{and} \quad M+1, M+2, M+3, \ldots

Tactic: Determine how much the $t$th I/O step can increase the number of possible realizable permutations.
\[(N + 1) \log N \geq \left( t + \frac{B}{N} \right) = 0 / I \]

There are \( N \) blocks initially unaccessed.

If this is first access to block,

\[
\binom{O / I}{(O / I - t) \text{ readable permutations after } \#} \times \binom{B}{W} \times iB
\]

If block was previously accessed

\[
\binom{O / I}{(O / I - t) \text{ readable permutations after } \#} \times \binom{B}{W}
\]

\( O / I \) read I/O that Realizable After #

Assumption: The \( N \) items to permute are indivisible.
\[
\left\{ u^\log N \, v^\text{min} \right\} \cup \left\{ \frac{(B/W)^\log B}{(B/N)^\log N} \, v^\text{min} \right\} \cup = \mathcal{L}
\]

\[
\frac{B \cdot N}{N} = \mathcal{L}
\]

\[
B \cdot N - N \cdot \frac{B}{W} \cdot N = \left( N \cdot \frac{B}{W} \cdot N \right) \cdot \mathcal{L} + \left( B \cdot \log B \right) \frac{B}{N}
\]

\[
N \log N = \left( N \cdot \log B \right) \cdot \mathcal{L} + \left( B \cdot \log B \right) \frac{B}{N}
\]

\[
(iN \log i)^\mathcal{U} = \left( (iN \log i) + \left( B \right)^\log B \right) \cdot \mathcal{L} + iB \log B \frac{B}{N}
\]

Taking logs and applying Stirling’s approximation:

\[
iN < \left( (N \log N + 1)N \left( \frac{B}{W} \right) \right)^\frac{B}{N}(iB)
\]
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

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

☆ 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 $i$th 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 $N(1 + \log N)$. 

Jeff Vitter
More Refined Analysis to Get Leading Coefficient

\[
\star \quad \implies (N(1 + \log N))^{R+W} \prod_{1 \leq i \leq R} \binom{M}{b_i} \geq \frac{N!}{(B!)^{N/B}}
\]

\star Let \( \tilde{b} \) be the average value of \( b_i \).
\star By convexity argument, LHS is maximized by setting each \( b_i := \tilde{b} \).
\star \( W \geq \frac{1}{B} (\sum_{1 \leq i \leq R} b_i) = \frac{1}{B} (R\tilde{b}) \implies R \leq (R + W)/(1 + \tilde{b}/B) \).
\star \( (N(1 + \log N))^{R+W} \left( \frac{M}{\tilde{b}} \right)^{(R+W)/(1+\tilde{b}/B)} \geq \frac{N!}{(B!)^{N/B}} \).
\star Maximize LHS by setting \( \tilde{b} = B \), so we get

\[
(N(1 + \log N))^{R+W} \left( \frac{M}{B} \right)^{(R+W)/2} \geq \frac{N!}{(B!)^{N/B}}.
\]

which gives desired lower bound on the total number \( R + W \) of I/Os.
Define togetherness function as

\[
\begin{align*}
(\bar{y}_i)f \quad & \quad \bigcup_i \quad = (\bar{y}) M C \\
(\bar{y}_i)x f \quad & \quad \bigcup_i \quad = (\bar{y}) x C
\end{align*}
\]

Let \( y_i \) = number of steps in internal memory that should be in \( i \) th block.

Let \( x_i \) = number of steps in \( i \) th block that should be in \( i \) th block.

\[
\begin{cases}
0 = x \text{ if } 0 \\
0 < x \text{ if } \log x
\end{cases}
\]

We define \( (x)f \) as when \( B \) is \( I/O \) input.

NOTE: Transposition is a special case of permutation. It can be done in \( \Theta \) time.

\[
\left( \log_m \left\{ \log \min \left\{ u, \{b, d\} \min \left\{ M \right\} \right\} ight\} \theta \right)
\]

stored in row-major order is

\[ b \times d \times \theta \]

**Theorem 3.3** The number of \( I/O \) required to transpost a matrix
\[
\begin{align*}
\mathcal{U} &= \text{Lower bound} \iff \\
\left( \frac{\mathcal{B} \log m}{\text{Potential}(T) - \text{Potential}(0)} \right) &= (m \log \mathcal{B})O = \\
(1 - t)^Y \mathcal{C} - (1 - t)^W \mathcal{C} - (t)^W \mathcal{C} &= (t) \Delta \text{Potential} \\
\text{We can show that:} \\
\mathcal{B} > \{b, d\} \max \quad \text{if } \max \{b, d\} \max > \mathcal{B} > \{b, d\} \min \quad \text{if } \min \left\{ \begin{array}{c}
\mathcal{B} > \{b, d\} \max \\
\mathcal{B} < \{b, d\} \min
\end{array} \right. \\
\{b, d\} \min > \mathcal{B} \quad \text{if } \min
\end{align*}
\]

\[
\mathcal{B} \log N = (T) \text{Potential} \\
\sum_{1 \leq t} (t)^Y \mathcal{C} + (t)^W \mathcal{C} = (t) \text{Potential}
\]
expense of having blocks not be contiguous in each run or bucket.
This work also noticed that sorting can be done in-place, at

\[
\left( \frac{B}{K} \log u \right) \Theta = \text{OS/I #}
\]

B (each record is different):

Each approach gives us a lower bound on the problem of bundle sorting,

Combination of permutation approach and matrix transposition

Bundle Sorting [MSV]
Recursive Matrix Multiplication

\[
\begin{bmatrix}
A_{1,1} & B_{1,1} \\
\end{bmatrix} + \begin{bmatrix}
A_{1,2} & B_{2,1} \\
\end{bmatrix} = \begin{bmatrix}
C_{1,1} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
A_{1,1} & B_{1,1} \\
\end{bmatrix} + \begin{bmatrix}
A_{1,2} & B_{2,1} \\
\end{bmatrix} = \begin{bmatrix}
C_{1,1} \\
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\]

☆ I/O complexity for \( K \times K \) matrices:

\[
T(K) = 8T \left( \frac{K}{2} \right) + 6 \frac{K^2}{B}
\]

\[
= 9\sqrt{3} \frac{K^3}{B\sqrt{M}}. \quad (2)
\]
Iterative Matrix Multiplication

\[
\begin{bmatrix}
\text{Block 1} \\
\text{Block 2}
\end{bmatrix}
\begin{bmatrix}
\text{Block 3} \\
\text{Block 4}
\end{bmatrix} =
\begin{bmatrix}
\text{Block 5} \\
\text{Block 6}
\end{bmatrix}
\]

☆ 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.
3 times faster when the reblocking is done all up front

\[
\left( \frac{W^\lambda}{X^\lambda} \log_{2} \frac{I}{W^\lambda} + 1 \right) \frac{M}{2X^\epsilon} + \frac{M}{W^\lambda} \left( \frac{3/2}{X^\lambda} \right) = (X)^L
\]

I/O Complexity for multiplying two $X \times X$ matrices:
The Need for Memory-Adaptive EM Algorithms.

☆ Traditional EM algorithms assume \textit{fixed memory} allocation.

☆ Problem:
  • OS/DBMS can \textit{dynamically} change memory allocation.
  • EM applications exhibit \textit{thrashing}.

☆ Solution:
  EM algorithms that \textit{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.
Merging 8 runs using 9 internal memory blocks.

Why Traditional EM Algorithms Trash?
Why Traditional EM Algorithms Thrash.

- 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 $m$. 

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

\( i \)th phase:

Internal memory

\[ (m_i = \frac{M_i}{B} \text{ blocks}) \]

Use comparison model:

\[
\text{# possible outcomes to comparisons per I/O} = \begin{cases} \quad B! \times \binom{M_i}{B} & \text{reading unread block.} \\ \quad \binom{M_i}{B} & \text{reading dirty block.} \end{cases}
\]

\[
(B!)^{N/B} \prod_i \left(\frac{M_i}{B}\right)^{2m_i} \geq N! \implies \sum_i 2m_i \log m_i = \Omega(n \log n).
\]

Jeff Vitter
Resource Consumption of Sorting

Sorting algorithm completes in $\ell$ phases

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

★ 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 $i$ $\implies$ 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

$$ RC_{\text{run-formation}} = O(#I/Os \times \text{Max cost of each I/O}) $$

$$ = O(n \log m_{\text{max}}) $$

☆ 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

\[
\text{RC}_{\text{sort}} = O \left( \text{RC}_{\text{run-formation}} + \frac{\log n_0}{\log R} \text{RC}_{\text{pass}} \right) \\
= O \left( n \log m_{\text{max}} + \frac{\log n_0}{\log R} \text{RC}_{\text{pass}} \right)
\]

For dynamic optimality,

\begin{itemize}
  \item \( \text{RC}_{\text{pass}} = O(n \log R) \).
  \item \( R = \Omega(m_{\text{max}}^c) \).
\end{itemize}
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, .....

Jeff Vitter
...  

String processing, molecular databases.

- Handling many disks, large merge orders, many partition

- See http://www.cs.duke.edu/TPIE

Conclusions and Open Problems