CS7960 L6 : I/O-Cache Oblivious + Parallel

Disk <--\textit{I/O}--> RAM <--\textit{I/O}--> CPU
N = size of problem
B = block size
M = size of memory
T = size of output
I/O = block move between disk + memory

Sorting N items:
\[ \Theta((N/B) \log_{M/B} (N/B)) \ll N \log_2 N \]

-------------------------

Cache-Oblivious Algorithms

[Frigo, Leiserson, Prokop, Ramachandran '99]

- design algorithms with good I/O efficiency without knowledge of M, B
- sometimes don't know M,B
- portable. Same code to different systems
- holds for all levels of hierarchy simultaneously
- does not work as well in practice.

Modeling assumptions
* Ideal Cache : cache always flushes the block that will be used furthest in future
  - LRU performs within constant factor
* Full Associativity: any block can go anywhere in cache (not always true - maybe 8 places)
  - can be gotten around using hashing, in expectation, with constant overhead
* Tall Cache: $M > B^2$ (usually $M > B^{1+a}$ for $a > 0$ constant ok).

-----

Scanning:
\[ [\frac{N}{B} + 1] \text{ I/0s} \]
- store elements in consecutive blocks of memory.
  \[ \ldots | XXX [X | XXXX | XXXX | XXXX | XX] XX | \ldots \]
  - Extra 1 because may not hit boundary exactly.

Array reversal?
\[ [\frac{N}{B} + 1] \text{ I/0s} \text{ (two scans from opposite ends)} \]

-----

Divide and Conquer:
Divide into subproblems until size is $< M$ (and $\Theta(M)$) or $< B$ (and $\Theta(B)$)

Median Finding:
(A) Split $D$ into $N/5$ sets of size 5 (adjacent)
(B) Find median of each set $\rightarrow M$
(C) Recursively compute median of $M$ $\rightarrow m$
(D) Split $D$ into $L$ ($l \in L < m$) and $R$ ($r \in R \geq m$)
(E) Recur on $L$ or $R$.

A: free
B: 2 scans | first on $D$, second records median to $M$
C: recursive call of size $N/5$
D: 3 scans | first on $D$, second and third records $L$ and $R$
E: recursive call of size $N(7/10)$

$T(N) = O(N/B + 1) + T(N/5) + T(7N/10) = O(N/B + 1)$

------
Binary Search:
\[ \Theta(\log N - \log B) \]
- recall if we know $M, B$ then $\Theta(\log N/\log B) = \Theta(\log_B N)$

Merge Sort:
\[ O((N/B) \log_2 (N/B)) \]
- recall if we know $M, B$ then $\Theta((N/B) \log_{\{M/B\}} (N/B))$

- same can be achieved with variation of
Quick Sort == Distribution Sort
or with "Funnel Sort" -- similar to merge sort but split $N^{1/3}$ pieces and merge $N^{1/3}$ way with a "funnel"

---------------------------
Parallel External Memory

P1 - [M]  | |  [ D ]
P2 - [M]  |I|  [ I ]
P3 - [M]  |/|  [ S ]
...       |O|  [ K ]
Pp - [M]  | |  [   ]

- P CPUs.
- each CPU has private cache of size M
- block of size B
- P block transfers == 1 I/O (one for each CPU)
- Block level CREW

------
Scanning
scan_P(N) = $O(N/PB + \log P)$ parallel I/Os
if $P \leq N/(B \log N)$ --> scan_P(N) = $O(N/BP)$

------
Sorting
sort_P(N) = O((N/PB) log_{M/B} (N/B))
parallel I/Os
  if P <= N/B^2

--------------------------
Parallel Disk Model (PDM) for External Memory

    | | - d1
    |I| - d2
P - [M] - |/| - d3
    |O| ....
    | | - dD

M << N, 1 <= DB <= M/2 (often M^2)

Assume transfers are synchronous, although faster otherwise.

[Vitter + Schriver '94]

sometimes ...
  p1 - [M1] - | | - d1
  p2 - [M2] - |I| - d2
  p3 - [M3] - |/| - d3
    ... |O| ....
  pP - [MP] - | | - dD

Scanning:  \( \Theta(N/DB) \)
Sorting : $\Theta((N/DB) \log_{\{M/B\}} (N/B))$
Search : $\Theta(\log_{\{DB\}} N)$

Striping :

... | 111 | 222 | 333 | 444 | 555 | 666 | 777 | 888 | 999 | ...

-->  
D1  ... | 111 | 444 | 777 | ...  
D2  ... | 222 | 555 | 888 | ...  
D3  ... | 333 | 666 | 999 | ...  

Usually extending regular EM algorithms to striped discs is sufficient  
- a few new ideas needed...

How to stripe a single-disk queue?

--------------------------

TPIE : Templated Portable I/O Environment  
(formerly, Transparent Parallel I/O Environment)  
http://www.madalgo.au.dk/Trac-tpie

What do you think?  
- How useful is it?  
- How would you change/extend the model?