Cache Oblivious Model

Lecturer: Nodari Sitchianva
Scribe: Romain Gratia

12/12/2012

1 Introduction

In this model the parameters M and B from the External Memory model are unknown. It means that performances of a CO algorithm are hardware-independent. Nonetheless the cache replacement policy has to be assumed. During this course, the Ideal-Cache Model will be used :

- Program with only one memory
- Analysis like I/O model, M and N arbitrary
- Supposed optimal off-line cache replacement strategy for M and B
- Supposed fully-associative cache :
  - Direct mapped cache.
  - Explicit memory management.
  - Dictionary (2-universal hash functions) of cache lines in memory
  - Expected O(1) access time of cache line in memory

These assumptions can be considered as unrealistic, but this is justified by this theorem :

Theorem. *Let $Q_{M,B}(N)$ be the I/O complexity of algorithm A with optimal replacement policy. Then on Least Recent Used cache-policy, A will have I/O complexity $\leq 2Q_{2M,B}(N) = O(Q_{2M,B}(N))$.*

This theorem shows that LRU is an optimal replacement cache policy since if the size of the cache is doubled then the number of cache misses will be at most doubled.
We also assume tall cache assumption : $M = \Omega(B^2)$. Otherwise there will not be optimal cache oblivious algorithms.

2 Searching trees

We are considering a binary tree with N nodes :

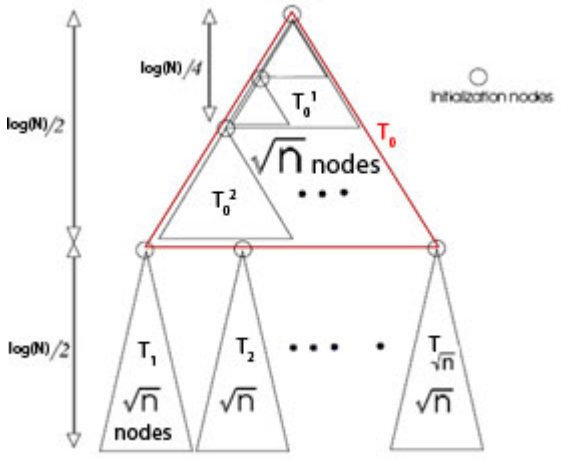

FIGURE 1 – Binary tree and his subtrees

The tree is subdivised in subtrees so that his lay out in memory each memory block contains the biggest tree which can fit in :

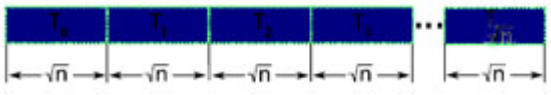

FIGURE 2 – Binary tree laid out in memory

Here I/O corresponds to the number of blocks we are going to access. To access a leaf we are going to access : O($\frac{logN}{logB}$) = O(log_B N) subtrees of size B each.

More formal demonstration :

$$Q(N) = \begin{cases} Q(\sqrt{N}) + Q(\sqrt{N}) + O(1) = O(log_B N)(N > \alpha B) \\ O(1)(N < \alpha B) \end{cases}$$

3 Sorting : Distribution sort

3.1 Algorithm

1. Divide array A[1..N] into $\sqrt{N}$ subarrays & recursively sort each one.
2. – Pick $\sqrt{N-1}$ pivots i.e. $\sqrt{N}$ buckets
   – Distribute each subarrrays A_i into these buckets $B_1 \ldots B_{\sqrt{N}}$
3. Recursively sort each buckets

The recursion stops when only one element.

$$Q(N) = \begin{cases} 2\sqrt{N} \times Q(\sqrt{N}) + O(\frac{N}{B}) = O(\frac{N}{B} log_M N)(N > M) \\ O(\frac{N}{B})(N \leq M) \end{cases}$$

Since we assumed tall cache assumption : Q(N) = O($\frac{N}{B} log_($ $\frac{M}{B}$)($\frac{N}{B}$))

3.2 Analyze in cache-oblivious

First we assume that the pivots are given. First attempt :

Dist(A1 ;. . . ;A$\sqrt{N}$;B1 ;. . .,B$\sqrt{N}$) For i=1 to $\sqrt{N}$ For j=1 to $\sqrt{N}$ Dist(A_i, B_j)
I/O complexity :
- Read : O($\frac{N}{B}$) I/O's
- Write : O($\sqrt{N}\sqrt{N}$) = O(N) I/O's

Instead of we use a recursive algorithm :

$$Dist(A_1; \ldots; A_{\sqrt{N}}; B_1; \ldots; B_{\sqrt{N}})$$

if N=1 : copy appropriate elements of A1 to B1 (*)

$$Dist(A_1; \ldots; A_{\frac{\sqrt{N}}{2}}; B_1; \ldots; B\frac{\sqrt{N}}{2})$$

$$Dist(A_1; \ldots; A_{\frac{\sqrt{N}}{2}}; B_{\frac{\sqrt{N}}{2}+1}; \ldots; B_{\sqrt{N}})$$

$$Dist(A_{\frac{\sqrt{N}}{2}+1}; \ldots; A_{\sqrt{N}}; B_1; \ldots; B\frac{\sqrt{N}}{2})$$

$$Dist(A_{\frac{\sqrt{N}}{2}+1}; \ldots; A_{\sqrt{N}}; B_{\frac{\sqrt{N}}{2}+1}; \ldots; B_{\sqrt{N}})$$

Analysis :
- Time complexity
  - Of jumps : T(m)=4T(m)+O(1)=O(m^2) (distribution over m buckets)
  - Copying (once an element is copied it's done) : O(1) per element.

  ⇒ Total : T(m,k)= O(m^2+k) (k : elements), in our example :
  T($\sqrt{N}$,N) = O(N)

- I/O complexity
  - Number of jumps (with tall cache assumption) :

$$Q(m) = \begin{cases} 4Q(m/2) + O(1) = O(\frac{m^2}{B})(m > \frac{M}{B}) \\ O(m)(m \leq \frac{M}{B}) \end{cases}$$

  - Number of copy : O($\frac{1}{B}$) I/O's per elements : just scan and copy, jumps are counted aside

$\Rightarrow$ Total I/O's : Q(m,k)=O($\frac{m^2}{B} + \frac{k}{B}$), in our example : Q($\sqrt{N}, N$)=O($\frac{N}{B}$)

But here we assumed that the pivots were given. Now we are going to take their choices into account. Picking pivots is in Copy(Ai,Bj) :

- Copy appropriate elements from Ai to Bj if |Bj|>$2\sqrt{N}$
  - Pick the median : x= median(B_j) (O($\frac{m}{B}$))
  - Partition items of B_j around x (O($\frac{m}{B}$))
  - Set x as a new pivot increasing the number of buckets by 1

Each bucket contains $\geqslant \sqrt{N}$ & $\leqslant 2\sqrt{N}$ elements. Now we have to take into account the I/O complexity.
Number of bucket splits is O($\sqrt{N}$) each split takes O($\sqrt{\frac{N}{B}}$) I/Os or O($\sqrt{N}$) time.
$\Rightarrow$ Total I/Os spent splitting the buckets : O($\frac{N}{B}$) I/Os or O(N) time.

4 Matrix multiplication

How to process the multiplication of two matrices $M_{nxn}M_{nxm} = M_{nxm}$
One way is to divide it in four submatrices :

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \times \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$

$\Rightarrow C_1 = A_1B_1 + A_2B_3$

If we store A in line and B in column : O($\frac{N^3}{B}$) I/Os, but how to store C ?

One solution is to lay out each matrix in a table by using the Morton Z-Order and then use the same principles than in the Distribution sort algorithm :

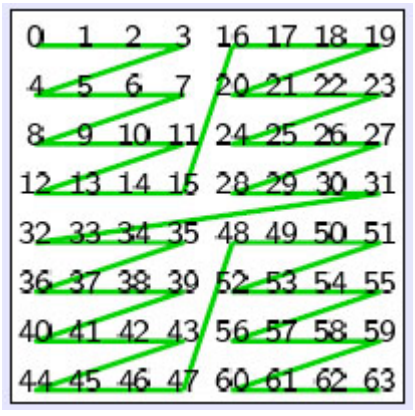

FIGURE 3 – Morton Z-Order example