Lesson 3-1 Basic Model of Locality

A First Basic Model

To find a locality aware algorithm we need a machine model - will be using a variation on the von Neumann model.

von Neumann Model:

Has a sequential processor that does basic compute operations
Processor connects to a main memory- nearly infinite but really slow
Fast memory - small but very fast, size = Z … measured in number of words

Rules:

1. The processor can only work with data that is in the fast memory, known as the local data rule.

2. When there is a transfer of data between the fast and slow memory, the data is transferred in blocks of size 'L', known as the block transfer rule.
For example: if you want to move [x] words from slow to fast memory, you need to pay to move L-x additional nearby words.

In this model you may need to consider data alignment

Costs:

The model has two costs associated with it:

1. Work, W(n) == the # of computation operations. How many operations will the processor have to perform?
2. Data transfers, Q(n;Z;L) == # of L-sized slow-fast transfers (loads and stores). The number of transfers is dependent upon the size of the cache and block size. This will referred to as 'Q' and be called the I/O Complexity.

Example:

Given an array of size 'n', sum its elements.
The processor needs to do at least n-1 additions, W(n) $\geq$ n-1 additions = $\Omega(n)$
For memory transfers → you need to make at least one pass through the data. This can be considered the lower bound on transfers:

Q(n,Z,L) $\geq$ ceiling(n/L) transfers = $\Omega(n/L)$
(The ceiling takes into account any partial transfer if n/L is not an integer)
Note the equation does NOT depend on Z, the size of the cache - because you are touching each data only once, so the size of the fast memory does not matter.
Reduction does not reuse data -- this is BAD!

Examples of Two-Level Memories:
hard disk & main memory
L1 cache & CPU registers
Tape Storage & Hard disk
Remote Server RAM & local Server RAM
The Internet & your brain

How many transfers are necessary in the worst case, assuming nothing about alignment?
Answer: the ceiling of (n/L) + 1
Here's an example:
Let n = 4 and L=2
Case 1: the array is aligned on an L word boundary. Then transfers = ceiling(n/L) = 2 transfers
Case 2: the array is not aligned on an L word boundary, then an extra transfer is needed

When n >> than L, the +1 can be ignored.

Minimum Transfers to Sort
Given an array of size n, sort it.
Assume a slow/fast memory model.
Recall comparison sorts need to perform n log(n) operations, W(n) = $\Omega(n\ log\ (n))$

What is the number of slow/fast memory transfers? ceiling(n/L) or just n/L
Q(n;Z;L) = $\Omega(ceiling(n/L)$ or $\Omega(n/L)$
n because each element is touched at least once, L because you read the elements from slow memory one block at a time.
This answer would be impressive: Q(n;Z;L) = $\Omega((n/L\ log(n/L)/log(Z/L))$

A matrix-matrix multiply on a machine with a two level memory.
The matrices are all n x n objects.
For a non-Strassen algorithm, work is W(n) = O(n^3)
Question: What is the minimum number of transfers?
Answer: Q(n;Z;L) = Ω (n^2/L)
The n*n counts the number of elements, dividing by L converts it to the number of transfers.

Answer if you are already familiar with the question: Q(n;Z,L) = Ω (n^3/(L $\sqrt{Z}$)

I/O Example Reduction

W(n) = $\theta(n)$ (work)
Q(n;Z,L) = $\Omega(n/L)$ (number of transfers)

Let’s look at an algorithm to see if we can achieve the lower bound:

For a sequential processor without fast memory:

```
s ← 0
for i ← 0 to n-1 do
    s ← s + X[i]
```

When you have a two level memory, you need to think about when to move data from slow to fast memory.

Assume s begins locally, already in the fast memory.
Assume n >> Z (the array is much bigger than the cache).
Assume X is aligned on an ‘L’ word boundary.

Now make slow and fast memory transfers explicit:

```
//assume: n >> Z
//          X aligned on L-word boundary
local s ← 0
for i ← 0 to n-1 by L do
    local L̂ ← min(n, i+L-1)
    local y[0:L̂-1] ← X[i:(i+L̂-1)]
    for di ← 0 to L̂-1 do  s ← s + y[di]
```

Note: for the outer loop, it steps through the array one block (L) at a time.
L^ → is the block of size ‘L’ or smaller? Can often ignore this detail.
y → this is a load from slow to fast memory, it requests at most ‘L’ words (1 block transfer).
Since s and y are local to fast memory, the processor can execute the innermost loop.

Work = W(n) = Θ(n)

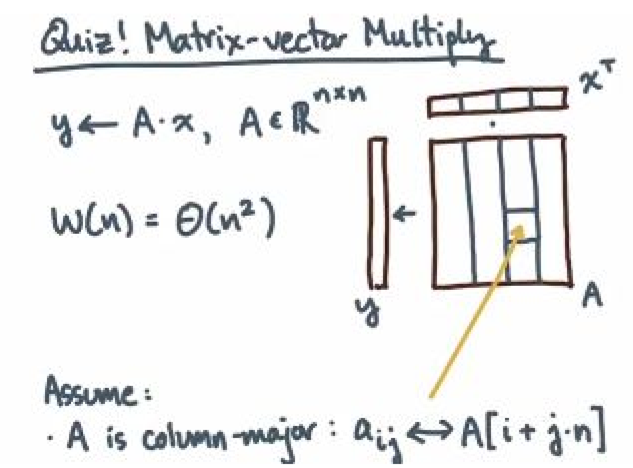
Transfers = Q(n;Z,L) = Θ(ceiling of (n/L))

Compare to the lower bounds:
Lower bounds: Work = W(n) = Θ(n), Q(n;Z,L) = $\Omega(n/L)$

Observation:
Caches are very fast, but they are not sufficient to guarantee high performance.

Matrix-Vector Multiply
Multiply a dense n x n matrix,'A', by a vector, 'y'.

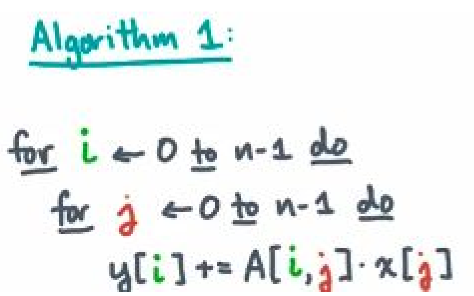
Work = W(n) = Θ (n^2)
The array is stored in memory in 'column major order'. The matrix is stored column-wise, one column follows the previous column in memory.



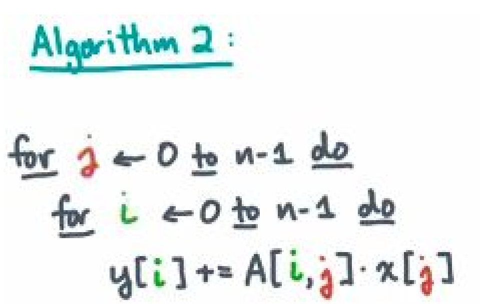
The element in memory can be found using the following rule:

a_{ij} < -- > A[i + (j-n)]

Consider two algorithms to compute the product:



In this algorithm the outer loop iterates over rows, inner loop over columns.



In this algorithm the outer loop iterates over columns, the inner over rows.

In the basic RAM model, these algorithms are identical.

Question: Which algorithm in the two level model does fewer transfers?
Assumptions:

- The fast memory can hold two vectors: $Z = 2n + O(L)$
- L/n -- L divides n
- all arrays and matrices are aligned on L word boundaries.
- can ignore floors and ceilings
- can assume the algorithm preloads x and y, and stores y at the end

These assumptions imply the number of transfers is at least:
$Q(n;Z,L) = 3n/L + ???$

So really … how many additional transfers does loading the matrix require.

Answer: Algorithm 2 requires fewer transfers.
Consider algorithm 1, it iterates over rows. So loading an element will load a blocks worth of column elements. (The array is stored by columns). Then the next element in the row will need to be loaded. This will cause a new column of elements to be loaded.
This will lead to $Q(n;Z,L) = 3n/L + n^2$

In algorithm 2, the block transfer matches the storage format.
This will lead to $Q(n;Z,L) = 3n/L + n^2/L$

In the sequential model these two algorithms are identical, but in the two level model they are different.

If you have a fully associative cache, will it help algorithm 1 to be as fast as algorithm 2?

Algorithmic Design Goals
What are the goals? What makes an algorithm good?

Goal 1: Work optimality
The two level algorithm should do the same work as the best asymptotic algorithm.

$w(n) = \Theta(W_*(n))$ W_* is the work of the best asymptotic algorithm

Goal 2: Has High computational intensity
This is the ratio of work to words transferred.

$$\text{Maximize: } I(n;Z,L) \equiv \frac{W(n)}{L \cdot Q(n;Z,L)}$$

Intensity is operations/word, it measures the data reuse of the algorithm. It is good to have high intensity, as long as the work is optimized.
Should remind you of work and span.

Which is Better?
Given two algorithms, which is better?

Algorithm 1:

$$W_1(n) = \Theta(n)$$

$$Q_1(n;Z,L) = \Theta\left(\frac{n}{L}\right)$$

Algorithm 2:

$$W_2(n) = \Theta(n \log n)$$

$$Q_2(n;Z,L) = \Theta\left(\frac{n}{L \log Z}\right)$$

Answer: Neither, there is insufficient information.
Recall the goals: low work and high intensity.
Algorithm 1 does less work, but the intensity is a constant.
Algorithm 2 the intensity grows.

$$I_1 = \frac{W_1}{LQ_1} = \Theta(1)$$

$$I_2 = \frac{W_2}{LQ_2} = \Theta(\log n \cdot \log Z)$$

Intensity, Balance, and Time
The relationship between work, transfers, and execution time.
τ = [time]/[operations]

Time to compute = $T_{comp} = \tau W$

α = amortized time to move data between slow and fast memory = [time]/[word]

The time to execute Q transfers = $T_{mem} = \alpha LQ$

The minimum time to execute the program = $T \geq \max(T_{comp}, T_{mem})$ … assumes perfect overlap

The execution time relative to the ideal running time:

$$T_{comp} = \tau W$$
$$T_{mem} = \alpha L Q$$
$$T \geq \max(T_{comp}, T_{mem})$$
$$= \underset{\substack{\uparrow \\ \text{ideal computation time}}}{\tau W} \max\left(1, \frac{\alpha/\tau}{W/(LQ)}\right)$$

It is ideal because it assumes data movement is free.

Must pay penalty for moving the data.
This is: ‘machine balance’/Intensity

B = machine balance is: [ops]/[word] (this is machine dependent)

[ops]/[word] → how many operations can be executed in the time it takes to move a word of data

$$T_{comp} = \tau W$$
$$T_{mem} = \alpha L Q$$
$$T \geq \max(T_{comp}, T_{mem})$$
$$= \tau W \cdot \max(1, B/I)$$

The time as a function of Balance and Intensity

The maximum time is: $T \leq \tau W(1 + B/I)$ (no overlap)

Normalize Performance:

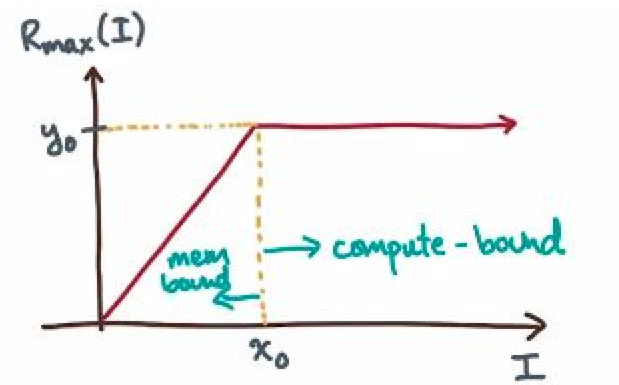
Normalized performance:

$$R \equiv \frac{\tau W_*}{T} \leq \frac{W_*}{W} \cdot \min\left(1, \frac{I}{B}\right)$$

Roofline Plots

To visualize the relationships between R, I , B look at a roofline plot.

Assume W_* and W are fixed, but I can vary.



Plot of this is a linearly increasing slope, an inflection point, and a plateau.

The value of the plateau and the location of the inflection.

What are the values of x_0, y_0?

x_0 = B the critical point is reached as soon as I == B. So when designing an algorithm, try for an intensity 1 => B.

$y_0 = W_*/W$ (it is the maximum possible value), if you design an algorithm that is not work optimal you will pay a penalty.

Intensity of Conventional Matrix Multiply

Consider a Matrix-Matrix Multiply (non von Strassen)

Execute this algorithm on a two level memory machine.

Assume:

- Transfer size == 1 word (L = 1 word)
- Z = 2n + O(1)

<u>Question:</u> What is the intensity of the algorithm?

$I(n;Z) = \Theta(1)$

Note:
$W(n) = \Theta(n^3)$
$Q(n;Z) = n^2$(for elements in A) + 2 n^2(for elements in C) + n^3(for elements in B)

The reads of 'B' dominate the overall transfer cost.
$Q(n;Z) = n^3$

I(n;Z) = ratio of W and Q = 1

Can you do better? Yes

There are n^3 transfers, and n^2 data. There might be an 'n' re-use of data available.

Intensity of Conventional Matrix Multiply Part 2

Divide the matrices into b x b blocks.

```
for i ← 0 to n-1 by b do
  for j ← 0 to n-1 by b do
    let Ĉ ≡ b × b block at C[i,j]
    for k ← 0 to n-1 by b do
      let Â ≡ b × b block at A[i,k]
      let B̂ ≡ b × b block at B[k,j]
      Ĉ ← Ĉ + Â · B̂
    C[i,j] block ← Ĉ
```

The reads and writes of blocks are slow/fast memory transfers.

Count the block transfers and determine the intensity of the algorithm.

Assume:

Assume: $L = 1$, $b \mid n$, $n \mid Z$, $Z = 3b^2 + O(1)$

Answer: $I(n;Z) = \Theta(b)$ or $\Theta\sqrt{Z}$

$W(n) = n^3$

$Q(n;Z) = \Theta(n^3/b)$

Blocking is better than individual element reading.