Parallel

Longest Common Subsequence

平行最長共同子字串

R04922067 楊翔雲, Morris

NTU CSIE

Longest Common Subsequence

- 在兩個序列中找到最長共同子序列
- 給定兩個字串 $X = x_1 x_2 \cdots x_n\,, Y = y_1 y_2 \cdots y_m$
- 位置序列 $S = s_1 s_2 \cdots s_r,\ \forall i < r : s_i < s_{i+1}$
- 生成字串 $C(X, S) = c_1 c_2 \cdots c_r$, 其中 $c_i = x_{s_i}$
- 目標找到 $|S|$ 最大，且滿足 $C(X, S_x) = C(Y, S_y)$

Recursive Formula

$$L[i,j] = \begin{cases} L[i-1,j-1]+1 & , if\, x_i = y_j \\ \max\{L[i-1,j], L[i,j-1]\} & , if\, x_i \neq y_j \\ 0 & , otherwise \end{cases}$$

Naive Implementation

- 時間複雜度 $O(nm)$，空間複雜度 $O(nm)$
- 回溯法 (backtracking) 找解

Recursive Formula

$$L[i,j] = \begin{cases} L[i-1,j-1]+1 & , if\, x_i = y_j \\ \max\{L[i-1,j], L[i,j-1]\} & , if\, x_i \neq y_j \\ 0 & , otherwise \end{cases}$$

Advanced Implementation

- Multiple buffering、滾動數組
- 時間複雜度 $O(nm)$，空間複雜度 $O(\min(n,m))$
- 如何找到一組解？Hirschberg's Algorithm

Advanced Algorithm

$O(nm)$ 不是最好的！我們可以做到 $O(\frac{nm}{\log n})$。

How to Do

- Method of Four Russians 四個俄羅斯人算法
- 這只是一種技術 - 藉由預建表達到加速
- 建表空間 $O(n^{1.5})$

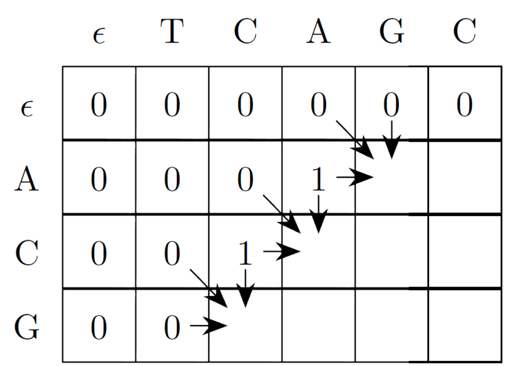
> 儘管能壓縮表格，查表效能好嗎？Mind the Cache！

Data Dependency

根據遞迴公式，所有情況依賴左、上、左上三格

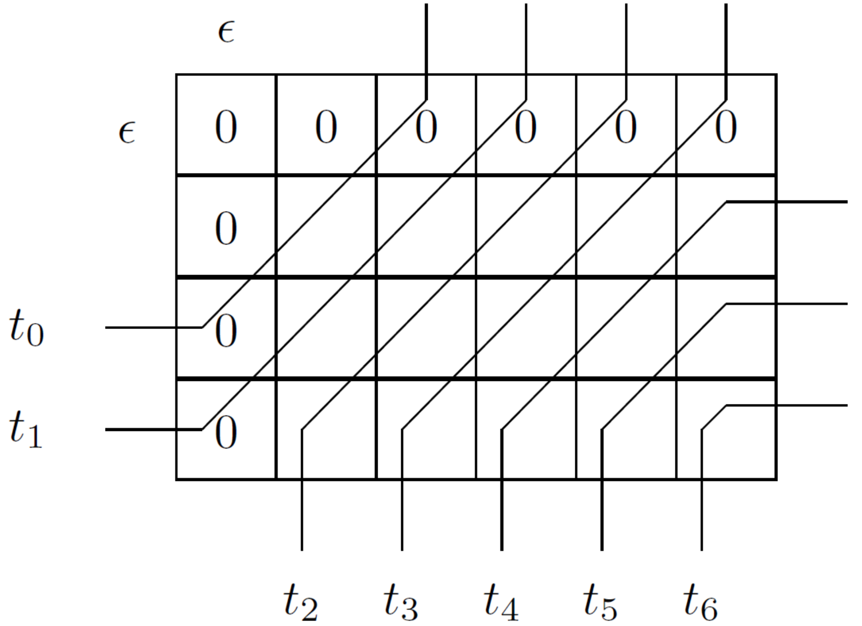
$$L[i,j] = \begin{cases} L[i-1,j-1]+1 & , if\ x_i = y_j \\ \max\{L[i-1,j], L[i,j-1]\} & , if\ x_i \neq y_j \\ 0 & , otherwise \end{cases}$$

	ϵ	T	C	A	G	C
ϵ	0	0	0	0	0	0
A	0	0	0	1	1	1
C	0	0	1	1	1	2
G	0	0	1	1	2	2



How to Parallel LCS ?

Wavefront Method 波前法：
平行處理波上的資料，如下圖 t_i 表示第 i 次運行。
當矩陣為 $n \times m$ 時，運行 $n + m - 1$ 次。



Efficiency

> 同一個演算法，效能天差地遠

提高資料局部性

Enhanced Data Locality

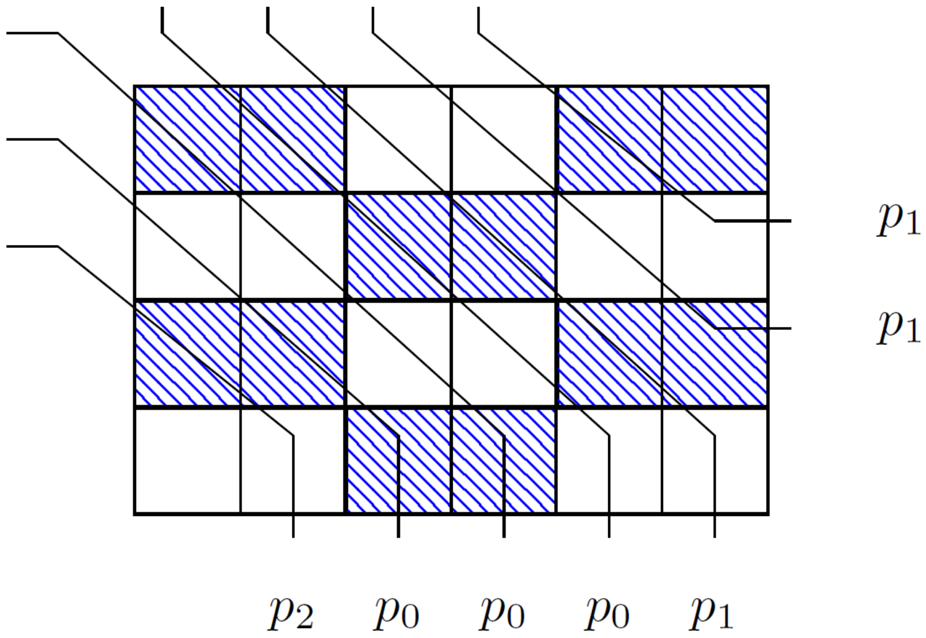
減少通信成本

Reduce Communication Costs

Efficiency - Problem

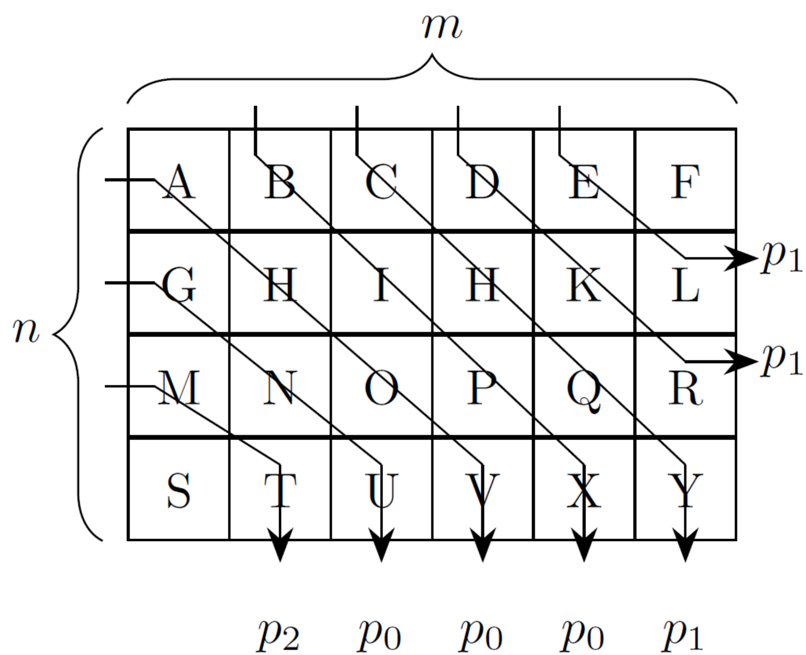
假設每一 1×2 的區塊為一個 page size

造成不同的處理器會使用同一個 page $\Rightarrow$ false sharing

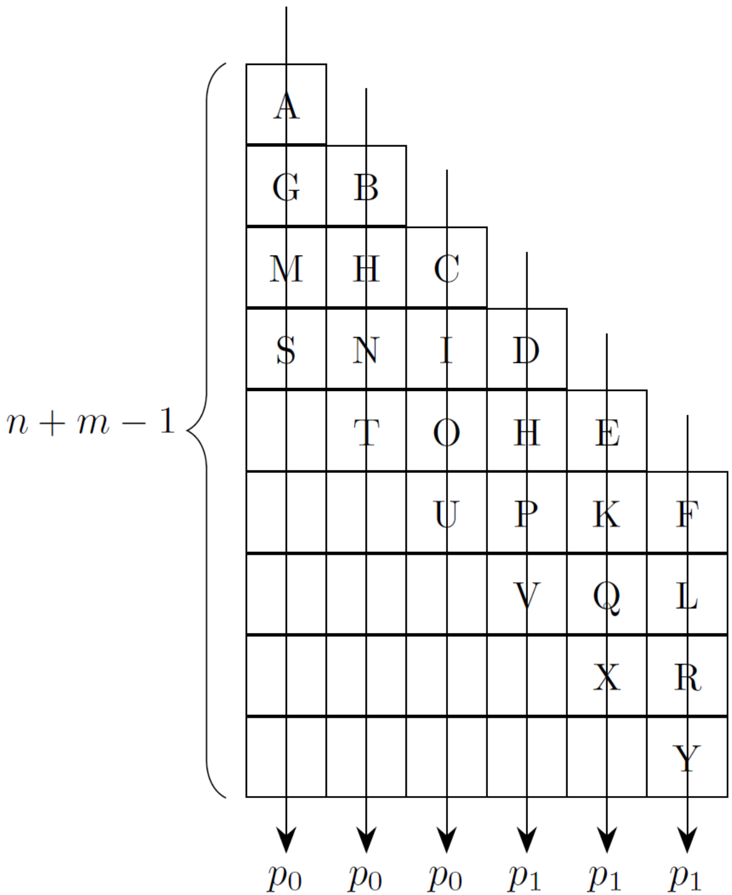


Efficiency - Data Layout

換個排列就會好了些 ... 但仍需要 $n + m - 1$ 次



Efficiency - Data Layout



Take a Break

假設我們有 10 個處理器

$10\times$ faster

運行長度 $n = m = 10^6$，單一處理器約為 4096 s

現在約為 512 s

Do it Better

有沒有可能少於 $n+m-1$ 次完成？可以，我們改公式吧！

$C = \Sigma = \texttt{ACGT}$：字元集

$P[i,j]$：對字元 c_i，位置 j 之前（包含）出現 c_i 的位置。

$$P[i,j] = \begin{cases} 0 & , if\, j = 0 \\ j & , if\, y_j = c_i \\ P[i,j-1] & , o.w. \end{cases}$$

這一步可以平行嗎？可以

- $O(|C|m/p + m)$：每個字元獨立
- $O(|C|m/p + \log m)$：**倍增找解**

Doubling Algorithm

倍增算法，找到最靠近該位置的最近匹配位置

$$P[i,j,0] = \begin{cases} 0 & , if\, y_j \neq c_i \\ j & , if\, y_j = c_i \end{cases}$$

$$P[i,j,k] = \max\{P[i,j,k-1], P[i,j-2^{k-1},k-1]\}$$

- Serial algorithm $O(|C| m \log m)$
- Parallel algorithm $O(|C| m \log m / p + \log m)$

空間只需要 $O(m)$，我們只需要最後那一排。

Doubling Algorithm - Practice

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
$\mathtt{C}_0$	0	0	2	0	0	5
$\mathtt{C}_1$	0	0	2	2	0	5
$\mathtt{C}_2$	0	0	2	2	2	5
$\mathtt{C}_3$	0	0	2	2	2	5

Practice - $P[i,j]$

$$X = \texttt{ACG}, Y = \texttt{TCAGC}$$

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
$\texttt{A}$	0	0	0	3	3	3
$\texttt{C}$	0	0	2	2	2	5
$\texttt{G}$	0	0	0	0	4	4
$\texttt{T}$	0	1	1	1	1	1

Recursive Formula

每次迭代 $c = x_i$

$$L[i,j] = \begin{cases} 0 & , if\, i = 0 \; or\, j = 0 \\ L[i-1,j] & , if\, P[c,j] = 0 \end{cases}$$

$$L[i,j] = \max\{L[i-1,j], L[i-1, P[c,j]-1]+1\}$$

與先前一樣，同時拔除尾端找子問題的最佳解

保證匹配 x_i 和 $y_{P[c,j]}$，而非 x_i 和 y_j 是否匹配

- $L[i,j]$ 需 $O(nm/p+n)$
  $P[i,j]$ 需 $O(|C|m/p+\log m)$
- 最後得到 $O(n+\log m)$

Recursive Formula - Compare

> 保證匹配 x_i 和 $y_{P[c,j]}$，而非 x_i 和 y_j 是否匹配？

$\text{LCS}(\texttt{TCAG}, \texttt{AC})$

- 一般做法，在下述兩種情況找最佳解
    - $\text{LCS}(\texttt{TCAG}, \texttt{A})$
    - $\text{LCS}(\texttt{TCA}, \texttt{AC})$
- 現在可以這麼找
    - $\text{LCS}(\texttt{TCAG}, \texttt{A})$
    - $\text{LCS}(\texttt{T}, \texttt{A}) + \texttt{C}$

Practice - $L[i, j]$

$X = \texttt{ACG}, Y = \texttt{TCAGC}$

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
A	0	0	0	3	3	3
C	0	0	2	2	2	5
G	0	0	0	0	4	4
T	0	1	1	1	1	1

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
ϵ	0	0	0	0	0	0
A	0	0	0	1	1	1
C						
G						

Take a Break

降低 critial path 長度，屬於 **常數優化**

$2\times$ faster

現在約為 256 s

Level Parallelism

- Bit-level parallelism
  位元同時計算，如位元壓縮之類的算法設計
- Instruction-level parallelism
  編譯器、硬體，可藉由 instruction scheduling 增加平行度
- Memory-level parallelism
  現在位置
- Task parallelism
  將問題一分為多，分別處理好再合併

Back to Bit-level

- 之前所講的都是在一次運算只對一個數據 $L[i,j]$ 操作
- 若一次運算對 w 個數據操作，其 w 為硬體的平行度
    - 如 $L[i,j],\ L[i,j+1],\ \cdots,\ L[i,j+w-1]$
    - 當代 64-bit 架構，單一處理器能快上 60 个 倍

How to Do ?

The Art of the Algorithm

Bit-level - Introdution

因 $L[i, j]$ 與其相鄰格子差值至多為 1

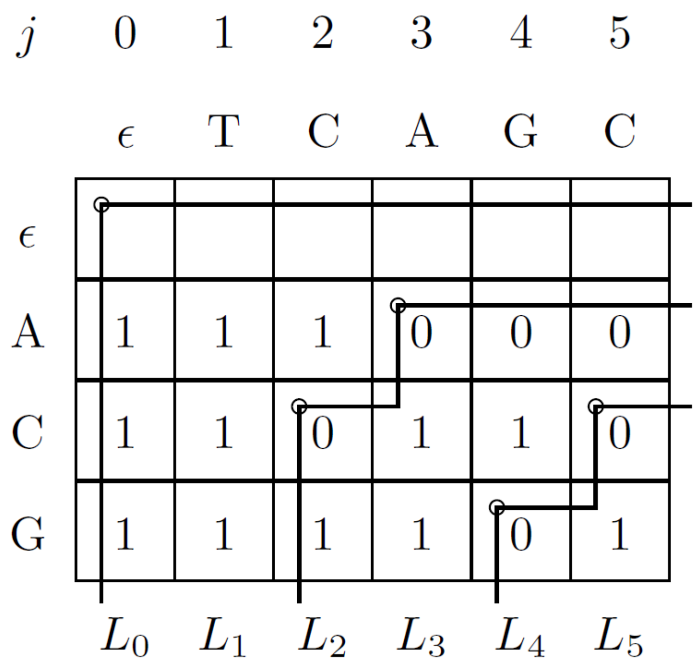
轉換成 k-dominant matches boudaries，找出有多少條

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
ϵ	0	0	0	0	0	0
A	0	0	0	1	1	1
C	0	0	1	1	1	2
G	0	0	1	1	2	2

Bit-level - CIPR Algorithm

分成 $m+1$ 個階段 $L_0, L_1, L_2, \cdots, L_m$

以 0 表示可以傳遞 L_j 至 L_{j+1}，$L_2 =$ 101、$L_3 =$ 110 ...



Bit-level - CIPR Algorithm

運算 L_j 至 L_{j+1}，Crochemore 給出下列公式

$$L_0 = 2^n - 1$$

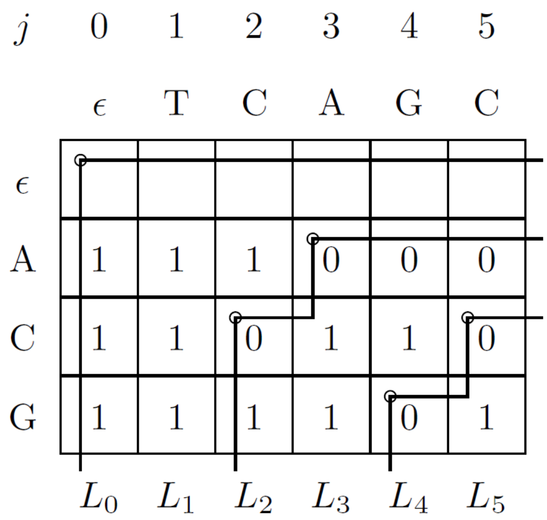
$$L_j = (L_{j-1} + (L_{j-1}\ \texttt{AND}\ M[y_j]))\ \texttt{OR}\ (L_{j-1}\ \texttt{AND}\ M'[y_j])$$

- $M[y_j]$
  字元 y_j 匹配字串 X 組成的位向量，0 為未匹配、1 為匹配
- $L_{j-1}\ \texttt{AND}\ M[y_j]$
  為下一階段可行的 k-dominant 位置，以 1 表示可行解
- $L_{j-1} + (L_{j-1}\ \texttt{AND}\ M[y_j])$
  利用加法進位的性質，將連續可行解移除掉
- $\texttt{OR}\ (L_{j-1}\texttt{AND}M'[y_j])$
  進位造成錯誤，用其修正之

Practice

$M[\texttt{A}] = \texttt{001}$、$M[\texttt{C}] = \texttt{010}$、$M[\texttt{G}] = \texttt{100}$、$M[\texttt{T}] = \texttt{000}$

- $L_1 = \texttt{111}$
- L_1 $\texttt{AND}$ $M[\texttt{C}] = \texttt{010}$
  支配點的可行解
  出現 1 的位置可成為新的支配點
- L_1 $\texttt{AND}$ $M'[\texttt{C}] = \texttt{101}$
  修正遮罩
  出現 1 的位置不可成為支配點
- $\texttt{111} + \texttt{010} = \underline{\texttt{1}}\texttt{001}$
  進位來移動支配線
  因 **溢位** 多紀錄一條
- $L_2 = \texttt{001}$ $\texttt{OR}$ $\texttt{101} = \texttt{101}$



Bit-level + Memory-level

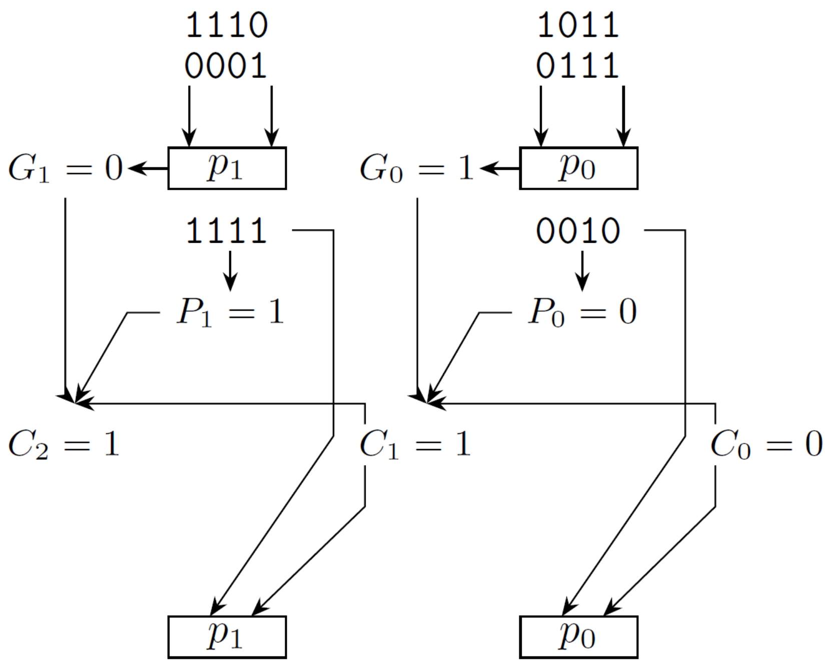
平行加法器於多處理器平台上，需要複習以下幾種加法器

- Ripple-carry Adder
- Carry-lookahead Adder
- Carry-select Adder
- More ...

Carry Adder

利用 $C_{i+1} = G_i + P_i\ C_i$ 、切數個區塊加法，$O(n/p + p)$

$\texttt{1110 1011} + \texttt{0001 0111} = \underline{\texttt{1}}\,\texttt{0000 0010}$



Take a Break

支援 64-bit 操作的計算機，增添 64 倍的效能改善

$128\times$ faster

現在約為 4 s

已經很完美了嗎？No

Back to Task

當平行遇到了瓶頸

一次能平行處理的資料數不夠多，如何拓展更高的平行度？

先決條件： 是否可在合併 Task 階段使用高效率的平行

Task-level - Introduction

將 X 拆成前半 X_{front} 和後半 X_{back}

$$\begin{aligned}\mathrm{LCS}(X, Y) = \mathrm{merge}(&\mathrm{LCS}(X_{\text{front}}, Y),\\ &\mathrm{LCS}(\mathrm{reverse}(X_{\text{back}}), \mathrm{reverse}(Y)))\end{aligned}$$

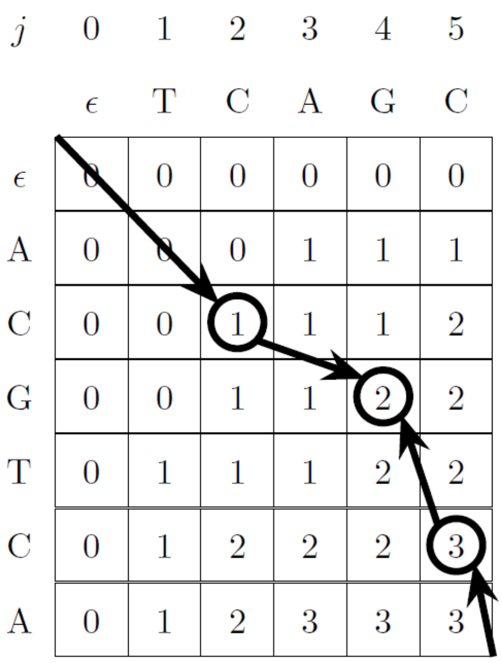
Task-level - Merge Task

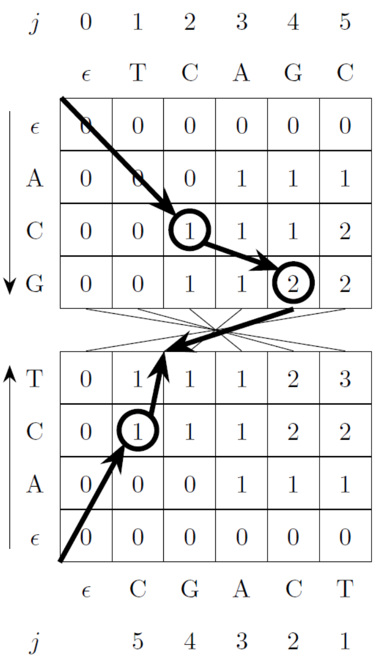
j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
ϵ	0	0	0	0	0	0
A	0	0	0	1	1	1
C	0	0	1	1	1	2
G	0	0	1	1	2	2
T	0	1	1	1	2	2
C	0	1	2	2	2	3
A	0	1	2	3	3	3

j	0	1	2	3	4	5
	ϵ	T	C	A	G	C
ϵ	0	0	0	0	0	0
A	0	0	0	1	1	1
C	0	0	1	1	1	2
G	0	0	1	1	2	2

T	0	1	1	1	2	3
C	0	1	1	1	2	2
A	0	0	0	1	1	1
ϵ	0	0	0	0	0	0
	ϵ	C	G	A	C	T
j		5	4	3	2	1

Task-level - Why





Take a Break

現在，我們能拆成至少 2 個階段

$$2\times \text{ faster}$$

現在約為 2 s

> 這已經是最快了嗎？還沒呢

Instruction-level parallelism

先不談指令平行

由於 shared memory 的關係，編譯器大多不做優化

你可以做到

- Copy optimization
- Improve hardware prediction
- Memory coalesce
- More ... 你可以修 **高等編譯器**

Instruction-level SIMD

- MMX/SSE/AVX
- Vectorization vs. Parallelization

Submit Your Best Algorithm

如果是妳的話，能給我更多啟發對吧？

- 批改娘 10110. Longest Common Subsequence
- 批改娘 10111. Longest Common Subsequence II
- 批改娘 20012. Bit Vector Adder
- Coming Soon

Reference

- Jiaoyun Yang, Yun Xu, Yi Shang, "An Efficient Parallel Algorithm for Longest Common Subsequence Problem on GPUs", 2010
- Maxime Crochemore, Costas S. Iliopoulos, Yoan J. Pinzón, James F. Reid, "A fast and practical bit-vector algorithm for the Longest Common Subsequence problem", 2001
- Maxime Crochemore, Costas S. Iliopoulos, Yoan J. Pinzón, "Speeding-up Hirschberg and Hunt-Szymanski LCS Algorithms", 2001