摘要

基因序列比對是生物資訊學中的一項重要工具，而一直以來的研究大多是以參考基因序列作為目標來比對，然而以一條序列已經不足以表達出基因的多樣性，因此以圖狀結構來表達出基因變異的參考基因組圖被提出，也逐漸地有些團隊提出了對於參考基因組圖比對演算法的研究。

在面對基因變異較豐富的區域時，採用啟發式演算法無法保證找出最佳的組合，因此我們以非啟發式的演算法來處理。動態規劃的策略可以使用在這個問題上，我們修改了史密斯-沃特曼演算法使其應用於參考基因組圖上，而這是一項耗費大量時間的演算法。

我們使用了 Intel Arria 10 的現場可程式化邏輯閘陣列來作為硬體加速器的平台，達到 128 倍平行化以加速計算，並透過管線化設計來達成 200MHz 的操作頻率。最終，視參考基因組圖由小到大，我們實現出比軟體加速 6.44 倍至 9.61 倍的加速器，並且在功耗以及記憶體使用量上也都有顯著的優化。

關鍵字：基因序列比對、參考基因組圖、動態規劃、硬體加速器、現場可程式化邏輯閘陣列

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

Abstract

Sequence alignment plays a crucial role as a fundamental tool in bioinformatics. Researchers traditionally focused on aligning sequences to a reference genome sequence. However, a single reference sequence is insufficient to express the diversity of genes. Hence, the concept of a reference genome graph, utilizing a graph structure to represent genetic variations, has been proposed. Currently, several researchers have begun to explore algorithms for sequence-to-graph alignment.

In regions enriched with genetic variation, using heuristic algorithms may not guarantee finding the optimal solutions. Therefore, we apply non-heuristic algorithms to address this issue. Employing dynamic programming methods is a strategic choice for this problem. We adapted the Smith-Waterman algorithm for its application to sequence-to-graph alignment. Nevertheless, a considerable amount of computation time is still required.

We utilize the Intel Arria 10 FPGA as the platform for our hardware accelerator, achieving a 128x parallelization to accelerate the computation. Through the finely-pipelined designs, the circuit operates at 200MHz frequency. Ultimately, we develop a hardware accelerator faster than the software implementation, achieving a speed-up ranging from 6.44x to 9.61x as the reference genome graph scales from small to large. Furthermore, there are significant improvements in power efficiency and memory utilization.

Keywords: sequence alignment, genome graph, hardware accelerator, dynamic programming, FPGA

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

目次

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

圖次

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

表次

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

Chapter 1

緒論

在生物資訊學 (Bioinformatics) 中，基因序列比對 (Sequence Alignment) 是一項關鍵的技術，可以用來識別和理解基因中的遺傳資訊，在基因組學、基因遺傳學、臨床醫學和生物學等領域中，基因序列比對都扮演著重要角色。

基因序列比對的目的是找出一個給定的 DNA(Deoxyribonucleic Acid) 或 RNA(Ribonucleic Acid) 序列與一個已知的參考基因組之間的相似和相異性。通過序列比對，科學家們可以鑑定出重要的遺傳元素，如基因的外顯子 (Exon)，在近期的研究中，也有許多科學家們將基因序列比對用於分析新冠病毒 (SARS-CoV-2) 上，藉此了解病毒的基因组、檢測病毒變種、以及做為醫學用途。基因序列比對是生物資訊學領域中重要且強大的一項工具。

傳統的基因定序技術，由於低通量、高成本與耗時長的特性，不利於大規模的應用。隨著科技的進步，次世代基因定序 (Next-Generation Sequencing，NGS) 迅速的發展，降低了基因定序的成本、減少定序所花的時間，並高度平行化的進行測序，使得科學家們可以擁有更龐大的基因資料進行研究，同時對於基因序列比對的速度和準確性需求也漸增。在現今，Illumina 和 Pacific Biosciences 為兩家次世代基因定序技術公司，分別代表著短片段序列 (Short-Read Sequence) 和長片段序列 (Long-Read Sequence)，在當今的研究中這兩項都是相當重要的技術。

在傳統的基因序列比對主要以動態規劃 (Dynamic Programming) 作為基礎，以局部比對的史密斯-沃特曼 (Smith-Waterman Algorithm)[1] 和全域比對的尼德曼-翁施演算法 (Needleman-Wunsch Algorithm)[2] 最常見，許多序列比對的演算法都以此為核心來延伸出不同的應用，並且也有許多啟發式 (Heuristic) 演算法被提出，

doi:10.6342/NTU202304510

以非保證最佳解的犧牲換取較快的運算速度。

隨著技術的不斷發展，科學家們逐漸意識到使用單一線性的參考基因組，已經不足以表示出個體和族群的基因多樣性，並且也無法表示出基因的變異。因此，參考基因組圖 (Genome Graph) 的概念被提出，這是將參考基因組以更靈活、更豐富的數據結構來表示，這可以代表著不同個體、不同族群、甚至不同物種間的遺傳差異。參考基因組圖將 DNA 序列的變異資訊以圖狀結構的形式來呈現，更好地把真實世界中的多樣性也納入參考基因組內。

對於參考基因組序列的比對已經非常發展得非常成熟，而對於參考基因組圖的比對則是相對較少，近期也有許多團隊提出了啟發式的演算法以快速的計算出結果。然而，因為參考基因組圖的特性，可以納入許多基因變異，因此可以用來做在基因變異較多區域上的研究，例如 BRCA1(Breast Cancer Type 1)、LRC(Leukocyte Receptor Complex) 等區域，面對這些區域，通常會需要一個準確的比對結果，而啟發式演算法並非能保證找出最佳解，因此準確算法在基因變異較多區域的應用上仍是有需求。

在硬體加速器中，可以藉著平行化 (Parallelization)、管線化 (Pipeline) 來增加計算處理的能力，並且節省運算所消耗的能量，因此很適合作為基因比對上的加速器。本論文將運用硬體加速器的技巧，處理基因變異較多區域的參考基因組圖比對。

1.1 論文貢獻

本論文提出一個對參考基因組圖的非啟發式準確比對加速平台設計，並且可以平行化處理多筆不同長度的短片段或長片段序列，並以現場可程式化邏輯閘陣列 (Field Programmable Gate Array) 實現該架構。我們使用兩個階段的方式找出正確序列方向及其起終點，並運用到邏輯閘陣列上的動態隨機存取記憶體 (Dynamic Random-Access Memory) 以便處理更大的參考基因組圖並且可以重複運用，同時以較低的功耗與較快的運算速度來完成計算。

本論文的主要貢獻如下：

1. 設計對參考基因組圖的非啟發式準確比對加速平台，輸出一張參考基因組圖

doi:10.6342/NTU202304510

和多條檢索序列後，平行化處理並得到比對結果，並實現於現場可程式化邏輯閘陣列上。

2. 檢索序列可為短片段或長片段序列，其中短片段序列可處理長度為 256bp 以下，至多處理 8,388,608 條，而長片段序列可處理長度為 32,768bp 以下，至多可處理 65,536 條，並且長度可為非定值。

3. 運用邏輯閘陣列上的動態隨機存取記憶體，以儲存更大的參考基因組圖並可重複利用，至多包含 12,582,912 個鹼基的參考基因組圖。

4. 設計應用於對參考基因組圖的史密斯-沃特曼演算法，並以硬體加速的技巧來實現。

1.2 論文架構

本章節概略描述生物資訊學上的背景知識與本論文的主題，並列出論文的主要貢獻。接著在章節2中會介紹本論文的背景知識及相關研究，章節3中將介紹所使用的演算法以及相對應的資料前處理，章節4將介紹我們的硬體架構、記憶體安排、及各個模組設計，並於章節5中進行實驗結果的分析，最終在章節6作為本論文的結論以及未來展望。

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

Chapter 2

背景知識與相關研究

我們本次設計的硬體可以平行化處理多個檢索序列 (Query Sequence) 對參考基因組圖 (Reference Genome Graph) 的基因序列比對 (Sequence-to-Graph Alignment)，參考基因序列的目標資料為基因變異豐富的區域，而檢索序列則是可以處理短片段序列 (Short-Read) 或是長片段序列 (Long-Read)，在本章節中包含了基因序列比對演算法的介紹與現有的相關研究。

2.1 基因序列

DNA(Deoxyribonucleic Acid) 是一種由多個核苷酸 (Nucleotide) 構成的雙股螺旋狀結構，每個核苷酸包含含氮鹼基 (Nitrogenous Base)、戊醣 (Pentose)、和磷酸基團 (Phosphate Group)。DNA 上的含氮鹼基有四種，分別為腺嘌呤（Adenine，簡稱 A）、胸腺嘧啶（Thymine，簡稱 T）、胞嘧啶（Cytosine，簡稱 C）和鳥嘌呤（Guanine，簡稱 G）。嘌呤與嘧啶之間會形成氫鍵 (Hydrogen Bond)，其中 A 只與 T 相連、C 只與 G 相連，相連的兩條形成了雙股螺旋狀。DNA 內有著生物體的遺傳資訊，對於科學和醫學用途，解析和研究 DNA 序列是非常重要的一環。

次世代基因定序技術（Next-Generation Sequencing，簡稱 NGS）是一項革命性的生物學工具，它使我們能夠快速、精確地解讀 DNA 序列。NGS 技術在過去幾十年中飛速發展，已經在生物或醫學相關的許多領域中帶來重大突破。透過 NGS 技術，科學家們能夠快速獲得生物體的完整基因組序列，這對於解釋遺傳變異、研究疾病機制、發展藥物和定制個體化醫療方案是非常重要的一環。

doi:10.6342/NTU202304510

當前，Illumina 和 Pacific Biosciences（PacBio）是兩家次世代基因定序技術公司，它們的技術和產品已在科學研究、臨床醫學和生物學領域產生了重大影響。

Illumina 以其高通量、低錯誤率和低成本的短片段 (Short-Read)DNA 定序技術而聞名。Illumina 的定序儀器如 Illumina HiSeq 和 Illumina NovaSeq 等，能夠將序列打散成碎片，並大量複製合成，最終進行平行定序，用儀器讀取定序資料。這種方法使科學家們能夠以極高的解析度快速獲得大規模的 DNA 序列數據。

PacBio 則提供一種完全不同的 DNA 定序方法。PacBio 的技術基於單一分子即時（Single-Molecule Real-Time，SMRT）定序，它使用連續的 DNA 多次複製和觀察單個分子。這種方法不同於 Illumina 的短片段定序技術，PacBio 能夠處理長片段 DNA 序列 (Long-Read)，克服了在重複區域和結構變化區域的挑戰。PacBio 定序儀器如 PacBio Sequel 和 Sequel II 能提供長片段序列，使其在基因結構變異和重複序列的研究中具有明顯的優勢。

2.2 基因序列比對

2.2.1 四種序列比對的計分

以檢索序列對參考基因組序列 (Reference Sequence) 的比對 (Sequence-to-Sequence Alignment) 為例，當進行序列比對時，會有一個已知的參考基因組序列作為參考，檢索序列將與參考基因組序列進行比對。對於每個檢索序列上的鹼基而言，可能產生四種結果：

1. 匹配 (Match): 檢索序列的鹼基和參考基因組序列的鹼基相同。
2. 不匹配 (Mismatch): 檢索序列的鹼基和參考基因組序列的鹼基相異。
3. 插入間隙 (Insertion): 檢索序列中多出了參考基因組序列中沒有的鹼基。
4. 缺失間隙 (Deletion): 檢索序列中缺少了參考基因組序列中沒有的鹼基。

四種結果的示意圖可參考圖2.1。使用者可以根據需求設定每種結果的得分和扣分，這些得分和扣分的設定將影響比對過程中的分數計算。

doi:10.6342/NTU202304510

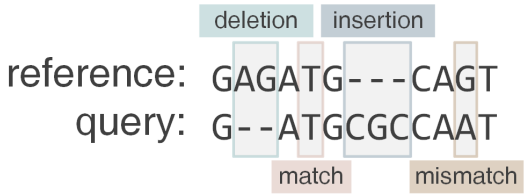

Fig. 2.1: 四種基因序列比對的結果

2.2.2 序列比對的方式

序列比對的方式可分為三種，全域比對 (Global Alignment)、半全域比對 (Semi-Global Alignment) 和局部比對 (Local Alignment)。以下是對這些比對方式的解釋：

1. 全域比對 (Global Alignment): 全域比對是一種將整個檢索序列與整個參考基因組序列進行比對的方法。它通常用於尋找兩個序列之間的總體相似性，即使這些序列的某些部分可能不完全匹配。最常使用的全域比對演算法之一是尼德曼-翁施演算法 (Needleman-Wunsch Algorithm)[2]，可以用以研究物種的演化或整個基因組的比較。

2. 半全域比對 (Semi-Global Alignment): 半全域比對是介於全域比對和局部比對之間的一種方法。它在不強制比對參考基因組序列的兩端的情況下尋找相似性。半全域比對通常用於找到參考基因組序列中的一個子序列與檢索序列的最佳匹配，而無需關心參考基因組序列的其餘部分。

3. 局部比對 (Local Alignment): 局部比對是用於尋找兩個序列之間的局部相似性高區域的方法。這種方法忽略序列的兩端，僅關注在兩個序列中找到的最佳匹配區域。最常見的局部比對演算法是史密斯-沃特曼演算法 (Smith-Waterman Algorithm)[1]，可以找出兩個序列中的任何片段的最佳匹配，用來識別特定基因、蛋白質區域。

這三種比對方式在生物資訊學中都有廣泛的用途，而本次實作的目標為多個檢索序列對一個參考基因組圖的比對，檢索序列是相對短的檢體，因此屬於一種局部比對。關於這三種比對方式可以參考圖2.2

doi:10.6342/NTU202304510

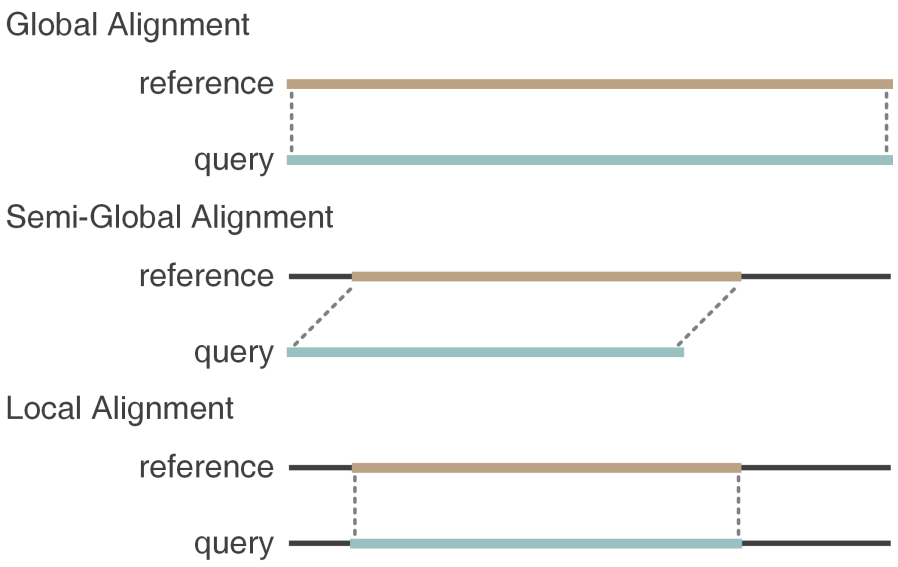

Fig. 2.2: 全域比對、半全域比對和局部比對的比較示意圖

2.2.3 史密斯-沃特曼演算法

在尋找檢索序列與參考基因組之間的局部比對時，可以用動態規劃的演算法來思考。為了解決這個問題，我們採用了史密斯-沃特曼演算法。在這個說明中，我們將比對的分數以 S 表示，參考基因組被放置在表格的橫軸、以編號 j 表示，而檢索序列則被放置在縱軸、以編號 i 表示。

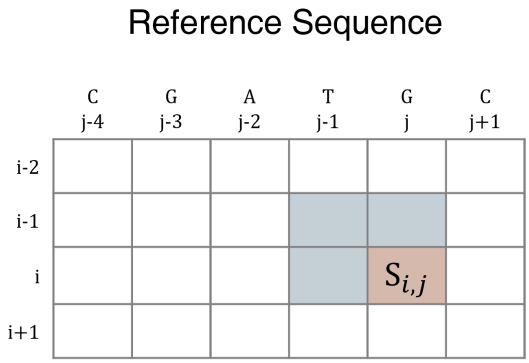

Fig. 2.3: 史密斯-沃特曼演算法的動態規劃表格示意圖

匹配或不匹配的分數是由左上方的格子提供，即 $S_{i-1,j-1}$ 再加上匹配得分或不匹配罰分；缺失間隙的分數則是由正左方的格子提供，即 $S_{i,j-1}$ 再加上缺失間隙的罰分；插入間隙的分數則是由正上方的格子提供，即 $S_{i-1,j}$ 再加上插入間隙的罰分。當前格子的分數由這些值中的最大值得到，若這些值都為非正數，表示無論從哪個方向提供的分數都不如從當前格子開始比對，當前格子分數則設為 0。

doi:10.6342/NTU202304510

此外，在邊界條件方面，當 i 為 0 或 j 為 0 時，分數被設為 0，以確保演算法的正確運作，這樣的動態規劃方法使得我們能夠有效地完成整個比對圖表的分數計算。

關於史密斯-沃特曼演算法可參考公式2.1與演算法1，其中 $\Delta_{i,j}$ 代表該格子的匹配得分或不匹配罰分，Δ_{del} 代表缺失間隙罰分，Δ_{ins} 代表插入間隙罰分。

$$S_{i,j} = \max \begin{cases} 0 \\ \Delta_{i,j} \\ S_{i-1,j-1} + \Delta_{i,j} \\ S_{i-1,j} + \Delta_{ins} \\ S_{i,j-1} + \Delta_{del} \end{cases} \tag{2.1}$$

Algorithm 1: Sequence-to-Sequence Smith-Waterman Algorithm

Input: query sequence $Q[1...m]$ and reference sequence $R[1...n]$
Output: best alignment score S_{max}

Initialization: $S[0,...] \leftarrow 0, S[...,0] \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    $\Delta_{i,j} \leftarrow (Q[i] = R[j])\ ?\ \Delta_{match} : \Delta_{mismatch}$
    $S[i,j] \leftarrow max(0,\ \Delta_{i,j})$
    $S[i,j] \leftarrow max(S[i,j],\ S[i-1,j] + \Delta_{ins})$ // insertion
    $S[i,j] \leftarrow max(S[i,j],\ S[i-1,j-1] + \Delta_{i,j})$ // match or mismatch
    $S[i,j] \leftarrow max(S[i,j],\ S[i,j-1] + \Delta_{del})$ // deletion
    $S_{max} \leftarrow max(S_{max},\ S[i,j])$
  end
end
return S_{max}

2.2.4 其他序列比對演算法

在序列對序列比對的領域中，已發展得較為成熟，有許多相關的演算法已被提出，而當中有許多使用啟發式 (Heuristic) 演算法來實作，以提高效率和降低計算複雜度。Blast[3] 將檢索序列分為小片段，稱為 K-mers，然後搜索這些 K-mers 在參考基因組序列中的匹配。這種方法能夠快速找到相似性區域，並有效降低計

doi:10.6342/NTU202304510

算成本。Minamap2[4] 採用了種子-連結-比對 (Seed-Chain-Align) 的演算法，首先尋找匹配的種子序列，然後連結這些種子以形成候選比對區域，最後進行精確的比對，這種策略使 Minimap2 能夠高效地找到相似性區域，特別是在長序列的情況下。

這些啟發式演算法已經廣泛應用於序列對序列的比對，並且在基因組學和生物資訊學中發揮了重要作用。此外，針對這些演算法加速的開發也成為當前的熱門研究領域，有許多團隊提出了使用硬體加速器來使這些演算法可以更快的計算出結果 [5]，以進一步提高比對的效率和速度，這些工具和技術的發展使研究人員能夠更有效地處理大規模的基因序列數據，並從中獲得有價值的資訊。

2.3 參考基因組圖

傳統的參考基因組是以單一線性序列的方式儲存的，然而這種線性的表示法只能表現到單一個體的 DNA，而且也因為只有一條參考序列表達，可能會造成等位基因偏差 (Allele Bias)，因此在近期的參考基因組中，例如 GRCh38，就已經加入了替代基因座 (Alternate Locus，簡稱 ALT) 的概念，而現今已經累積了多個個體和族群中的龐大基因組數據，用線性序列已不足以表達參考基因組。

因此，參考基因組圖的概念被提出 [6]。為了解決單一參考基因組序列的侷限性，參考基因組圖將基因變異加入參考基因組序列，形成圖狀的結構，更能表達出跨個體或是跨族群之間的基因變異。在進行序列比對時，研究人員可以使用參考基因組圖來替代傳統的線性參考基因組，特別適用於那些基因變異較為複雜的區域。

我們可以利用軟體 vg toolkit[7][8] 來生成參考基因組圖，要先準備參考基因組的序列作為基底，其檔案形式可以為 FASTA 或 FASTQ，同時把基因變異也一併輸入，其檔案形式可以為 vcf(或是 vcf.gz)，而在使用基因變異時也會需要其索引 (VCF index)，若原本基因變異資訊中未包含索引，可以使用 SAMtools[9] 來產生索引。最終，我們會得到副檔名為 vg 的參考基因組圖，此為一個二進位檔，可以藉由同一套軟體 vg toolkit 來將其轉化成可讀的文字檔 txt，或者是使用同樣由 vgteam 所提供的套件 libvgio[10] 來讀取參考基因組圖中的資訊。

生成的參考基因組圖由三種資訊所構成：節點 (Vertex)、邊 (Edge)、路徑

doi:10.6342/NTU202304510

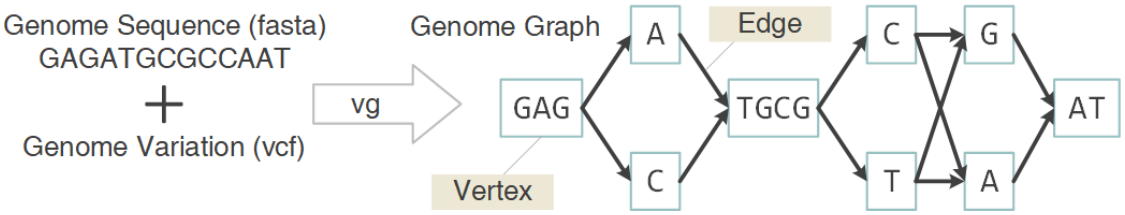

Fig. 2.4: 以 vg 合成基因組圖示意圖

(Path)，該圖為有向圖 (Directed Graph)。在此結構中，每一節點代表一段長度大於零的基因序列及其編號，在 vg 中以 S 來標示；每一條邊則由兩個節點的編號和其連接的方向構成，在 vg 中以 L 來標示，代表著兩個節點之間的連接關係；至於路徑，其主要功能在於標示出參考基因組所包含的節點及其連接方向，用以從基因組圖重建出基因序列，因此在本次設計中不會用到路徑資訊，而是著重於節點和邊的結構性質。

此種圖像結構格式不僅擴展了參考基因組的適用範圍，也有助於更全面地理解基因變異，為基因組學和生物資訊學的研究提供了更強大的工具。參考基因組圖的引入使得基因定序和基因變異分析變得更加準確和全面，推動生物、醫學研究更進一步的發展。

2.4 檢索序列對參考基因組圖的序列比對

有了能保存更多資訊的基因組圖，因而衍生出了將參考基因組換成參考基因組圖的序列比對，有許多研究針對此需求而發展了相關的軟體，例如 vgteam 開發的 vg toolkit 便有這項功能，而後有基於位元-向量演算法 (Bit-Vector) 和種子-延伸演算法 (Seed-and-Extend) 而能更快計算的 GraphAligner[11]，種子-延伸演算法當中的計算種子算法 (Seeding)，雖可以降低計算所需的時間，但其本質為一個啟發式 (Heuristic) 的演算法，在面對基因變異較多的區域時容易降低精確性，並且在長片段序列時使用延伸算法 (Extending) 容易產生延伸錯誤。因此面對基因變異較多的區域時，使用精確無啟發式的演算法會是較佳的選擇。

doi:10.6342/NTU202304510

針對本文所面向的主題，也就是短片段或長片段檢索序列，以及參考基因組圖為基因變異較多的區域時，而需要精確算法的情形，有一相近的研究 PaSGAL[12] 也被提出，並且使用了軟體執行緒上的 SIMD(Single Instruction Multiple Data) 平行化加速，這也是本文的主要比較對象。

doi:10.6342/NTU202304510

Chapter 3

演算法

在這個章節中，將描述我們實作的硬體加速器中所使用的演算法，以及相對應的資料前處理。

3.1 資料前處理

由 vg 所讀取的參考基因組圖中，節點和邊的資訊被分開儲存 (在本設計中路徑資訊並未被使用)。此外，這些節點的編號並未按照特定順序排列，也並非保證鄰近的節點其編號會相差較小。換言之，對於給定的節點，其來源邊的節點編號可能在其前方或後方。這兩個特性對於建立動態規劃表格而言並不利，因此在本次設計中我們對參考基因組圖進行了以下的資料前處理，這些前處理是對參考基因組圖的一次性操作，即替換掉檢索序列時可以使用同一張已進行前處理過的基因組圖。

3.1.1 拓撲排序

在我們的方法中，在整張圖被載入進硬體加速器之前，我們首先在主機端進行拓撲排序 (Topological Sort) 操作，利用拓撲排序的特性來重新編排每個節點的編號。換言之，針對排序後的圖，每個節點的編號皆保證小於其來源邊的節點編號。這樣的處理方式使得每個節點僅需查找比自身編號更小的節點，因此在建立動態規劃表格的過程中，無須考慮向後方查找格子的情況，只需要向前方查找。

doi:10.6342/NTU202304510

而拓樸排序也使整個圖狀結構變為一個有向無環圖 (Directed Acyclic Graph，簡稱 DAG)。

而拓樸排序時，我們設計了一個使用簡單的佇列 (Queue) 和廣度優先搜尋 (Breadth-First Search) 來完成拓樸排序的算法，一般而言廣度優先搜尋是無法達成拓樸排序的，但由於參考基因組圖是由一個長鏈的主要序列再向外延伸出許多分支，是個較簡單的圖狀結構，因此採用廣度優先搜尋便能達成拓樸排序，並且能夠使相鄰節點的編號也是相近的。最終，拓樸排序可以在 $O(V+E)$ 的時間複雜度下完成計算，其中 V 為節點個數、E 為邊個數，此時間複雜度相較於硬體中的史密斯-沃特曼演算法而言是較低的，在後面的實驗結果章節5.3.2也會將此前處理的花費時間列出。

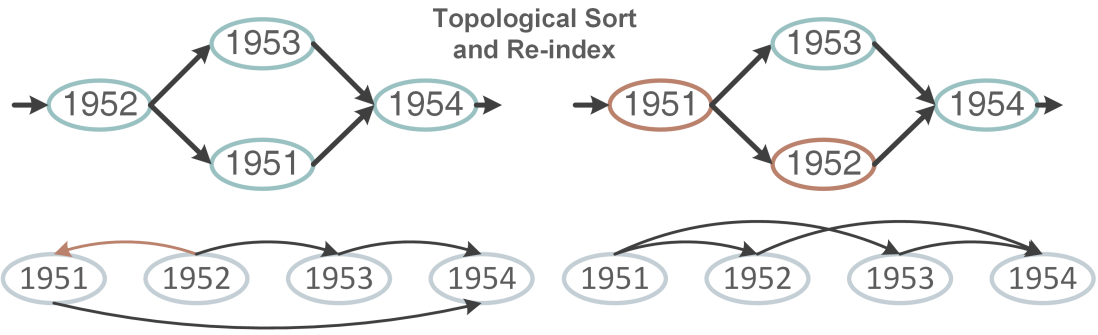

Fig. 3.1: 拓撲排序前後示意圖。原先的基因組圖 (左圖) 可能會有來源邊的節點編號倒序的情況，造成需要向前後方來查找；經過排序與重新後 (右圖)，讓所有節點都可以只向前方查找。

3.1.2 合併節點和邊

資料傳輸進到硬體加速器中的方式是一筆一筆固定 128 位元的資料傳入，因此我們對 vg 的儲存方式進行了結構上的改造。在原先 vg 的結構中，節點和邊被分開儲存，因此我們改變了資料的儲存形式，我們重新設計了資料的儲存形式，將節點和邊的資訊整合在一起，以便傳入硬體加速器。

在我們的儲存方式中，對於每個節點的鹼基，我們使用 2 位元來儲存其鹼基資訊 (A、T、C、G)，同時使用 8 位元來標示出邊的資訊。在邊的資訊中，每個位元對應到自身的前方 8 個鹼基是否有形成一條邊，舉例來說，0000_0011 在我們的表示法中就代表該節點的前方第一個和第二個鹼基分別與該節點形成一條邊。這

doi:10.6342/NTU202304510

種編碼前處理是參考自 PaSGAL[12] 中的參數 B_{width}，經過統計了實驗資料中三種大小不一的高變異區域參考基因組圖，未涵蓋到相對應邊的發生率低於 0.218%。由於後續兩階段演算法的特性，不易對找出正確的起終點會有影響，因此選定 8 位元來儲存邊的資訊，對於發生率與採用位元數的統計也會在實驗章節5.3.2來討論。對於轉化後的表示法可以參考圖3.2。

因此，我們選擇將鹼基和邊的資訊合併儲存，使每個鹼基需要 10 位元的空間。這樣的編碼方式使我們能夠在每 128 位元的資料中儲存 12 個鹼基，並將其傳送至硬體加速器進行高效率的運算，這種編碼方式為接下來的動態規劃演算法帶來幫助。

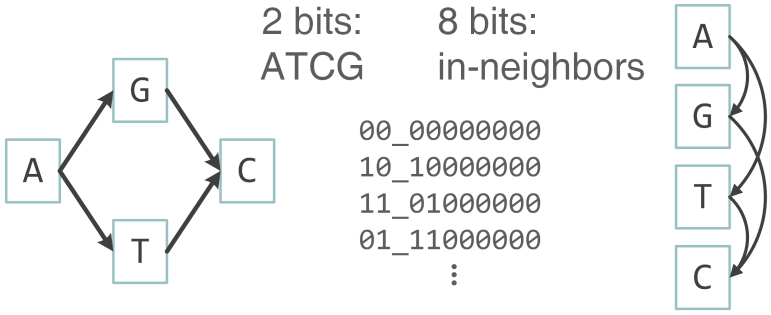

Fig. 3.2: 合併節點和邊後的 10bits 儲存方式。

3.2 應用於基因組圖之史密斯-沃特曼演算法

應用於基因組圖之史密斯-沃特曼演算法的概念參考自非啟發式的比對軟體 PaSGAL[12]，在此小節中將會深入描述，並整理出對應之虛擬碼 (Pseudo Code)，以供更好的理解執行的演算法。將檢索序列與參考基因組序列比對拓展為檢索序列與參考基因組圖的序列比對，經過資料前處理後，我們同樣可以用類似史密斯-沃特曼演算法的方式來計算出分數。在這個說明中，我們將比對的分數以 S 表示，參考基因組被放置在表格的橫軸、以編號 j 表示，而檢索序列則被放置在縱軸、以編號 i 表示。其簡單示意圖可參考圖3.3。

在修改後的史密斯-沃特曼演算法中，匹配或不匹配的分數不再只是由左上方的格子提供，而是正上方列所有來源邊的格子提供，再加上匹配得分或不匹配罰分；同理，缺失間隙的分數也不再只是由正左方的格子提供，而是當前列所有來

doi:10.6342/NTU202304510

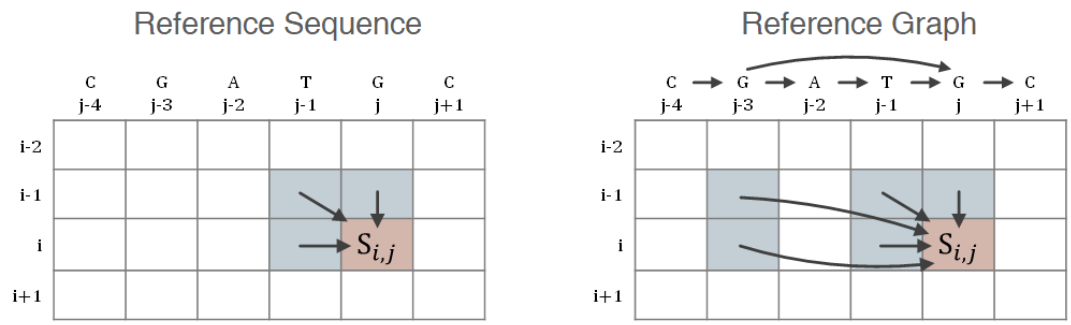

Fig. 3.3: 應用於基因組圖之史密斯-沃特曼演算法示意圖。左圖為原版的史密斯-沃特曼演算法；右圖為應用於基因組圖上的史密斯-沃特曼演算法

源邊的格子提供，再加上缺失間隙的罰分；插入間隙則保持不變，分數同樣是由正上方的格子提供，即 $S_{i-1,j}$ 再加上插入間隙的罰分。

$$S_{i,j} = \max \begin{cases} 0 \\ \Delta_{i,j} \\ S_{i-1,k} + \Delta_{i,j}, \quad \forall k : (k,j) \in E \\ S_{i-1,j} + \Delta_{ins} \\ S_{i,k} + \Delta_{del}, \quad \forall k : (k,j) \in E \end{cases} \tag{3.1}$$

假設檢索序列的長度為 m，而參考基因組圖的所有鹼基個數為 n，總共的邊數為 $|E|$，則應用於基因組圖之史密斯-沃特曼演算法的時間複雜度為 $O(m \times (n + |E|))$，而計算動態規劃需要填滿整個分數表格，因此所需要儲存空間的空間複雜度為 $O(m \times n)$。關於修改後的史密斯-沃特曼演算法可以參考公式3.1和演算法2，給定參考基因組圖 $G(V, E)$ 與檢索序列 Q，其中 $V[j][l]$ 表示在編號 j 上節點中的第 l 個鹼基，而 $E(j)$ 則表示節點 j 的所有來源節點；分數以 $S_{i,j}$ 來表示，而 Δ_{del} 為缺失間隙罰分、Δ_{ins} 為插入間隙罰分、Δ_{match} 和 $\Delta_{mismatch}$ 分別為匹配得分和不匹配罰分。

doi:10.6342/NTU202304510

Algorithm 2: Sequence-to-Graph Smith-Waterman Algorithm

Input: query sequence $Q[1...m]$ and reference graph $G(V, E)$ with total n nucleotides

Output: best alignment score S_{max}

Initialization: $S[0, ...] \leftarrow 0, S[..., 0] \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** $|V|$ **do**
    $\Delta_{i,j,1} \leftarrow (Q[i] = V[j][1])\ ?\ \Delta_{match} : \Delta_{mismatch}$
    $S[i, j] \leftarrow max(0,\ \Delta_{i,j,1})$
    $S[i, j] \leftarrow max(S[i, j],\ S[i-1, j] + \Delta_{ins})$ // insertion
    for $k \leftarrow 1$ **to** $|E(j)|$ **do** // $E(j)$ means in-neighbors of V[j]
      $S[i, j] \leftarrow max(S[i, j],\ S[i-1, k] + \Delta_{i,j,1})$ // match or mismatch
      $S[i, j] \leftarrow max(S[i, j],\ S[i, k] + \Delta_{del})$ // deletion
      $S\_temp[j, 1] \leftarrow S[i, j]$
      $S_{max} \leftarrow max(S_{max},\ S[i, j])$
    end
    for $l \leftarrow 2$ **to** $|V[j]|$ **do**
      $\Delta_{i,j,l} \leftarrow (Q[i] = V[j][l])\ ?\ \Delta_{match} : \Delta_{mismatch}$
      $S\_temp[j, l] \leftarrow max(0,\ \Delta_{i,j,l})$
      $S\_temp[j, l] \leftarrow max(S\_temp[j, l],\ S\_prev[j, l] + \Delta_{ins})$
      $S\_temp[j, l] \leftarrow max(S\_temp[j, l],\ S\_prev[j, l-1] + \Delta_{i,j,l})$
      $S\_temp[j, l] \leftarrow max(S\_temp[j, l],\ S\_temp[j, l-1] + \Delta_{del})$
      $S_{max} \leftarrow max(S_{max},\ S\_temp[j, l])$
    end
    $S\_prev[j, ...] = S\_temp[j, ...]$
  end
end
return S_{max}

doi:10.6342/NTU202304510

3.3 兩階段計算分數

為了尋找檢索序列在參考基因組圖上最為匹配的起始與終點，我們採用了兩階段的分數計算策略，這是與序列對序列比對的算法中常用的方法 [13]，同樣的策略可以直接套用在序列對圖的比對上。首先在第一階段，我們同時處理檢索序列及其反向互補的鹼基序列，以全面涵蓋兩種可能的方向。此外，我們依據分數的相對高低決定保留正常檢索序列或其反向互補序列，而被選擇的一方將進入第二階段進行反向運算。在第二階段，我們輸入所選取序列的反向版本以及反向的參考基因組圖，以確定最佳匹配的起始位置。如此一來，我們可以得到最高相似性序列的計算分數以及對應的起始與終點位置。

透過兩階段分數計算的策略，我們能夠找出最佳的局部比對。這種雙向計算的方法使得算法更能深入理解檢索序列與參考基因組圖之間的相似性，不僅僅局限於單一方向的比對，而是綜合兩種方向的資訊，提升了比對的全面性與精確性，找出了序列正確的兩端，並且在第二階段中可以操作第一階段一半數量的檢索序列，更能有效的減少計算時間。

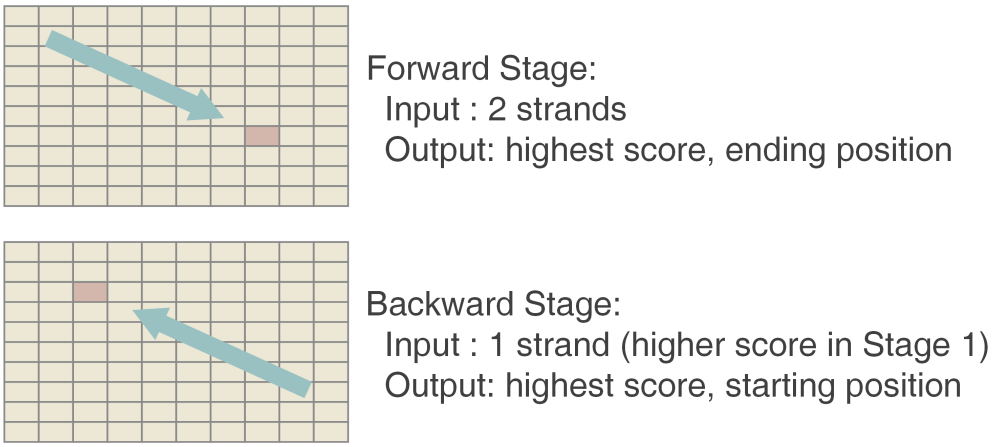

Fig. 3.4: 兩階段計算分數示意圖

3.4 平行化計算

上述的情形都建立在一條檢索序列對一張參考基因組圖的情況下，而因為操作史密斯-沃特曼演算法需要完成所有格子的計算，又因為並非以啟發式演算法來

doi:10.6342/NTU202304510

計算，難以使用條狀的技巧來加速，所以在我們的演算法設計中主要使用平行化來加速，也就是同時操作多條檢索序列對同一張參考基因組圖。

對於要完成同時操作多條檢索序列，演算法設計上有以下幾個特點：

3.4.1 全域記憶體

由於多條檢索序列是對上同一張參考基因組圖，因此我們將參考基因組圖放在全域記憶體 (Global Memory) 上，對於操作每條檢索序列都能看到同一張基因組圖，在儲存空間上也能有效的減少，不需要每條檢索序列都配置一張單獨的基因組圖。而參考基因組圖也正是需要最多儲存空間的資料，因此共享同一塊全域記憶體可以節省相當多儲存空間。當要取出參考基因組圖時，因為每條檢索序列同時只會需要參考基因組圖的某一片段，並且每條檢索序列的操作進度相同，只需要暫存出一塊，因此也不需要使用很大的儲存空間來暫存。

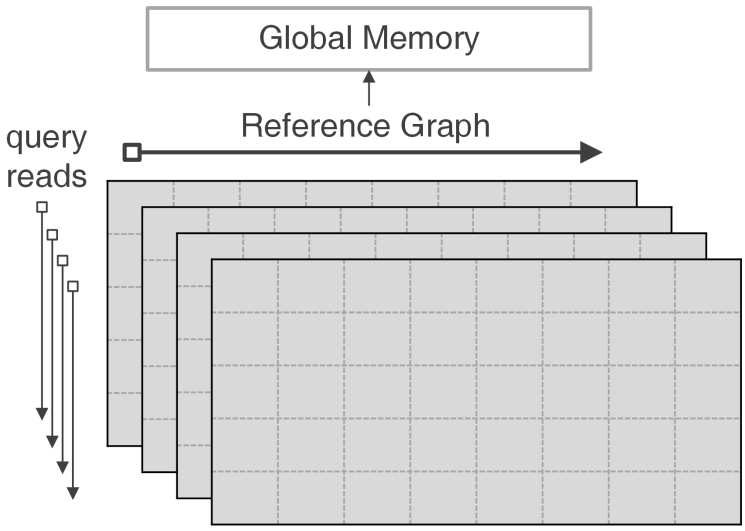

Fig. 3.5: 全域記憶體與平行化之示意圖

我們將處理每條檢索序列時的動態規畫表格稱為一個頁面，在同一個批次的平行化計算中，將會一次處理 $N_{parallel}$ 個頁面，每個頁面為獨立的計算單元。儘管每個頁面都共享相同的參考基因組圖，這部分資料可以存放在全域記憶體中，但是每個頁面的其餘計算內容，包括完整的動態規劃表格，都需要獨立的儲存空間。因此，我們必須有效地管理和控制每個頁面內部使用的記憶體，以確保系統能夠進行高度平行化計算。

doi:10.6342/NTU202304510

更詳細的記憶體配置會在硬體加速器章節4.2.5中來描述。

3.4.2 負載平衡

Illumina 定序技術生成的短片段基因序列，由於運作中使用的適配器 (Adapter) 是固定長度，因此生成的短片段基因序列長度亦為固定值，通常介於 50bp 至 300bp 之間，常見的長度為 100bp 或 150bp。這種特性確保了每次平行化操作的序列長度都是相同的，因此不存在由於序列長度不平衡而導致的操作負載不均的問題。

相對地，PacBio 定序技術生成的長片段基因序列具有不同特性，每條檢索序列的長度通常超過 10,000bp，並且每一條檢索序列的長度不一，存在長度差異大的情況。在同時操作多條檢索序列時，可能導致較短的檢索序列在較長的序列尚未完成計算之前就已經完成，進而使較短的序列需要等待較長序列的情況，容易造成負載上的不平衡。為了解決這一問題，我們首先對檢索序列的長度進行排序，讓相近長度的片段在同一批進行平行化操作。這樣的操作可以實現負載平衡，有助於克服不同長度檢索序列帶來的計算效率差異，提升整體計算效能。

3.5 資料傳輸

整體資料傳輸的架構參考自 RIFFA(Reusable Integration Framework for FPGA Accelerators) [14]，這是一個可以在電腦主機端 (CPU) 和硬體加速器 (FPGA) 之間溝通的框架。這個框架使用 PCIe(Peripheral Component Interconnect Express) 作為通信通道，經由 PCIe 來輸入和接收資料，以實現可靠且高頻寬的資料傳輸。

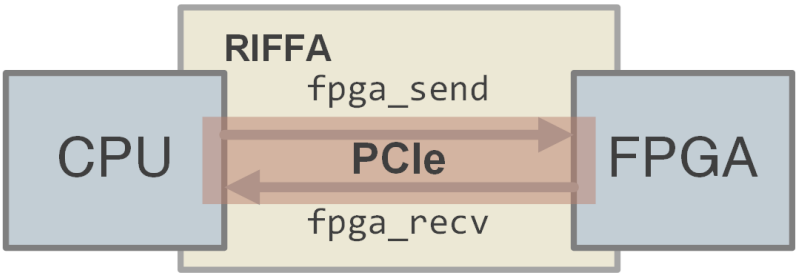

Fig. 3.6: RIFFA 連接主機端和硬體加速器示意圖。

我們的工作流程如下：首先，我們在主機端進行資料讀檔操作，接著進行前

doi:10.6342/NTU202304510

述的資料前處理，將資料整理並打包成每筆 128 位元的匯流排資料，這些資料隨後被傳送到硬體加速器進行的運算。等待加速器計算完成後，計算結果將被傳送回主機端。在主機端，我們執行驗證程序以確保接收到的計算結果是正確的，以確保整個操作流程的可靠性和正確性。

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

Chapter 4

硬體架構設計

4.1 系統架構

4.1.1 整體架構設計

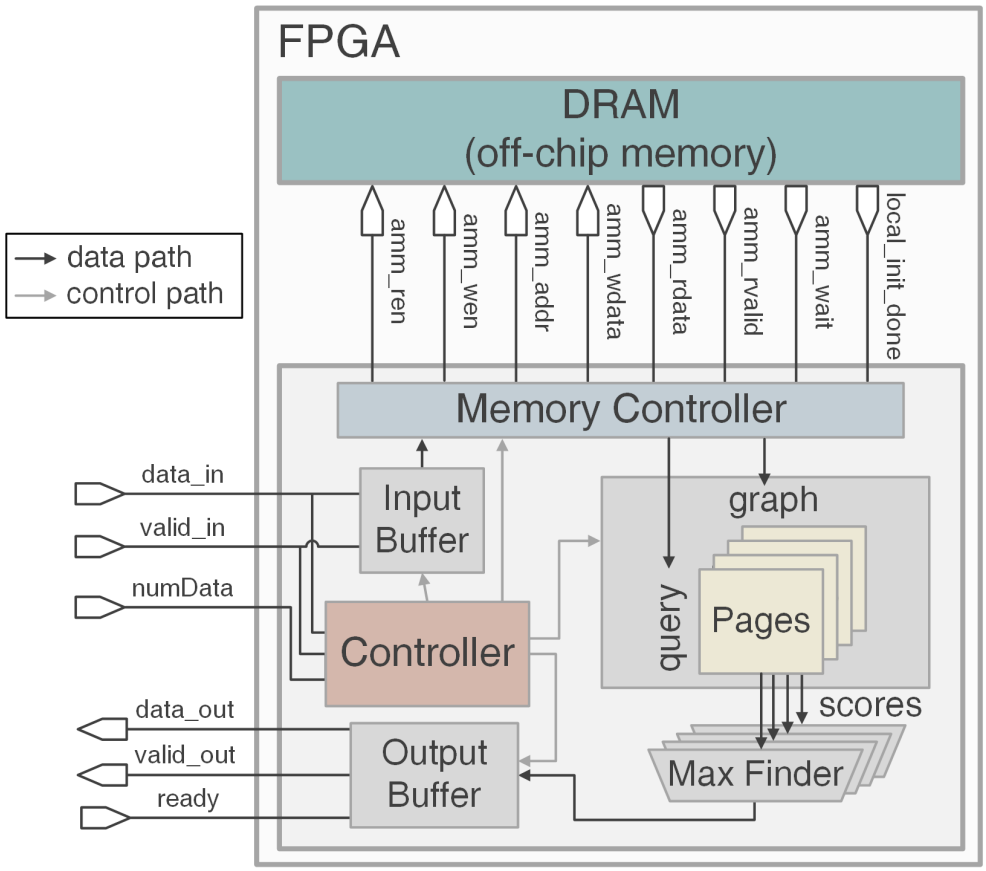

Fig. 4.1: 本設計的硬體主架構圖

doi:10.6342/NTU202304510

本設計的硬體架構設計圖如圖4.1，由一個主控制模組 (Controller Module)、一個記憶體控制模組 (Memory Controller Module)、數個頁面子模組 (Page Submodule) 所組成。資料由主機端傳輸至輸入端，接著由記憶體控制模組將資料儲存於動態隨機存取記憶體 (Dynamic Random-Access Memory, DRAM)，接著由主控制模組與記憶體控制模組溝通取得所需資料，再將資料發送至各個頁面子模組，等待頁面子模組運算完後再將分數傳回主控制模組，由主控制模組計算最大值並且安排頁面子模組的下一份資料，最終主控制模組判斷檢索序列已完成計算後，將輸出資料由輸出端回傳至主機端，即完成整個系統架構的流程。

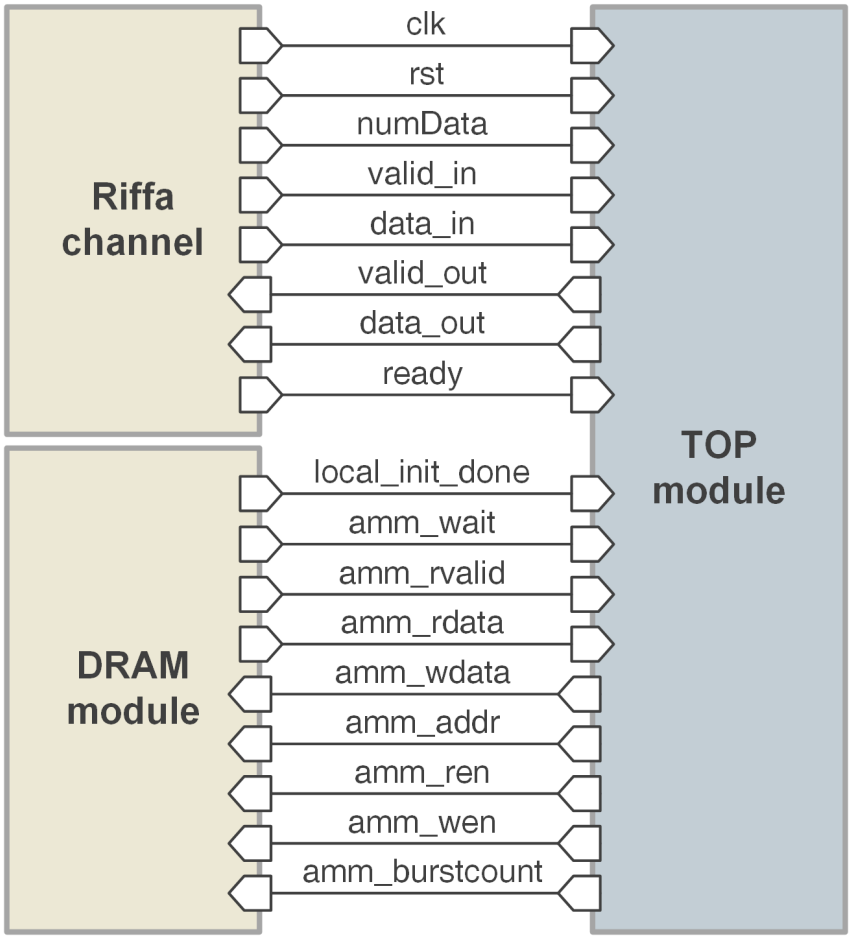

Fig. 4.2: 本設計的規格示意圖

4.1.2 規格說明

由於 PCIe 頻寬的限制，因此整個系統只會有一個輸入端，並且僅有 128 位元。在這個限制下，我們必須分批傳入資料，最一開始傳入的是整個系統的設定

doi:10.6342/NTU202304510

參數，包含參考基因組圖的長度、檢索序列的長度以及個數，接著傳入參考基因組圖的資料，最後再傳入檢索序列的資料，而檢索序列資料中又分為序列的長度以及資料本身。

關於詳細的規格說明可以參考圖4.2以及表格4.1。

Table 4.1: 本設計模組的輸入輸出規格描述

Port Name	I/O	Width	Description
clk	Input	1	系統時脈訊號。本設計為時脈的正緣觸發。
rst	Input	1	非同步重置訊號。當此訊號為高位準時，進行系統重置。
numData	Input	20	系統輸入總數。此訊號代表輸入訊號 data_in 的總數。
valid_in	Input	1	輸入有效訊號。當此訊號為高位準時，代表輸入訊號 data_in 為可讀取狀態。
data_in	Input	128	外部資料輸入匯流排。傳入系統的設定參數、參考基因組圖資料、及檢索序列資料。
valid_out	Output	1	輸出有效訊號。當此訊號被設為高位準時，代表輸出訊號 data_out 為可被 PCIe 端接收的狀態。
data_out	Output	128	外部資料輸出匯流排。由模組向 PCIe 端傳出的資料。
ready	Input	1	PCIe 端就緒訊號。當此訊號為高位準時，代表 PCIe 端處於可接收資料狀態。

4.1.3 時序圖

圖4.3中顯示了硬體的執行時序圖，此時序圖僅包含了兩階段演算法中的任一個階段。最一開始由主機端將輸入資料送進硬體加速器中，包含參數、參考基因組圖、檢索序列，接著便由主控制模組來啟動頁面子模組，經過大約 $Length_{graph} \times Length_{query}$ 個時脈週期後，便可以算出最高分，並將算好的值輸出回主機端，完成一個批次的計算，緊接著可以開始進行下一批次的計算，直至處理完所有序列。

doi:10.6342/NTU202304510

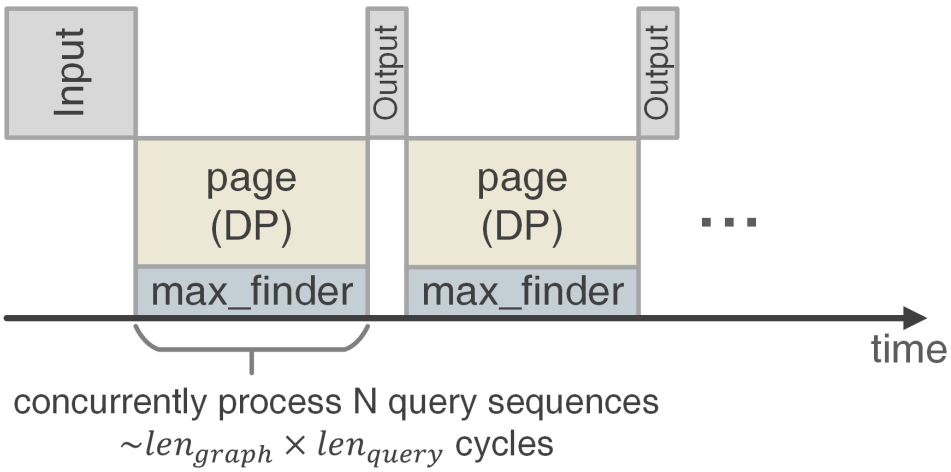

Fig. 4.3: 硬體加速器執行時序圖

4.2 動態隨機存取記憶體

4.2.1 塊狀記憶體的限制

由於在我們的設計中所使用的參考基因組圖具有兩項特性，其一是資料量龐大，例如實驗資料中的主要組織相容性複合體 (Major Histocompatibility Complex) 基因大小已經達到約 53Mb，而現場可程式化邏輯閘陣列上的塊狀記憶體 (Block RAM) 大小上限約為 54Mb，且因為塊狀記憶體的使用特性而無法全部達到 100% 使用率，因此使用塊狀記憶體是不足以容納所有參考基因組圖；另一項特性則是因為參考基因組圖會重複使用到，雖然可以藉由暫時停止傳輸來分段讀取參考基因組圖，但重複使用也就會向主機端多次索取資料，反而會增加傳輸上的延遲以及增加了對主機端的溝通，以硬體設計的角度來看，由主機端發送所有輸入資料至現場可程式化邏輯閘陣列，而運算完後再傳輸輸出資料至主機端，會是較好的硬體設計思路。

4.2.2 動態隨機存取記憶體的特性與協定

由於以上兩項特性，在此設計中採用的是現場可程式化邏輯閘陣列上的動態隨機存取記憶體，其為外置的記憶體，本設計所使用的是 DE5a-Net 板子的第四代雙倍資料率同步動態隨機存取記憶體 (Double-Data-Rate Fourth Generation Synchronous Dynamic Random Access Memory, DDR4 SDRAM)，其規格為 2400MHz、CL17、SODIMM 64，容量上可以達到單塊 4GB 的大小，其特性可

doi:10.6342/NTU202304510

以在單一個時脈週期內傳輸資料兩次，也就是在時脈的正緣及負緣都可以進行一次傳輸，並且一次傳輸的位寬為 64 位元。後面章節4.2.3介紹的突發可以使等效的讀取寫入位寬呈倍數，因此在以下的規格描述中會是寬度 128 位元的讀取和寫入資料。

Table 4.2: 動態隨機存取記憶體的輸入輸出規格描述

Port Name	I/O	Width	Description
local_init_done	Input	1	DRAM 校正狀態訊號。當此訊號為低位準時，代表 DRAM 初始化尚未完成，此時不能與 DRAM 溝通。
amm_wait	Input	1	DRAM 就緒訊號。當此訊號為高位準時，代表 DRAM 正在進行讀取或寫入，此使不能與 DRAM 溝通。
amm_rvalid	Input	1	DRAM 讀取有效訊號。當此訊號為高位準時，代表輸入訊號 amm_rdata 為可讀取狀態。
amm_rdata	Input	128	DRAM 讀取資料。接收來自 DRAM 之記憶體讀取資料。
amm_wdata	Output	128	DRAM 寫入資料。傳送到 DRAM 之記憶體寫入資料。
amm_addr	Output	32	DRAM 位址。DRAM 讀取或寫入的位址。
amm_ren	Output	1	DRAM 讀取訊號。當此訊號為高位準時，代表對 DRAM 進行讀取。
amm_wen	Output	1	DRAM 寫入訊號。當此訊號為高位準時，代表對 DRAM 進行寫入。
amm_burstcount	Output	6	DRAM 突發模式次數。進行突發模式傳輸的次數，預設值為 1。

動態隨機存取記憶體的最小單元為 1 個位元，是由 1 個電晶體以及 1 個電容所組成，以電晶體來控制電容的充放電，也就是控制了讀取以及寫出，而電容的內部電荷則代表了該位元的值為 0 或是 1，在其他種類的記憶體中與動態隨機存取記憶體中的原理不同，例如靜態隨機存取記憶體 (Static Random-Access Memory, SRAM) 中使用了 6 個電晶體來控制其狀態，而無使用到電容。由於動態隨機存取記憶體因為是外置於晶片的，再加上需要對電容充放電，因此與靜態隨機存取記憶體比較，它的傳輸時間會更長，且功耗的表現上會較差，但其單位面積下可以提供更多的儲存空間，因此容量可以設計的較大，在本次設計中作為來存取資料的記憶體。

動態隨機存取記憶體有其專屬的協定，要直接控制其協定是比較複雜的，而 Intel 對 DE5a-Net 所屬的 Arria 10 系列有提供已整合好的模組，可以直接控制動態

doi:10.6342/NTU202304510

隨機存取記憶體，其名為外置記憶體操作介面 (External Memory Interfaces, EMIF)，使用此模組就可以用 Avalon 記憶體映射 (Avalon Memory-Mapped) 的操作方式來控制動態隨機存取記憶體；而前面有提到動態隨機存取記憶體的工作頻率是較快的 (2400MHz)，因此需要使用跨時脈域的溝通橋接器 (Clock-Crossing Bridge) 來處理與我們的設計頻率不同的問題，並和外置記憶體操作介面中再使用管線化橋接器 (Pipeline Bridge) 來避免傳輸過程中的時序違反。關於這整套協定的架構可以參考圖4.4，而在 Quartus 中動態隨機存取記憶體的 Qsys 可以參考圖4.5。

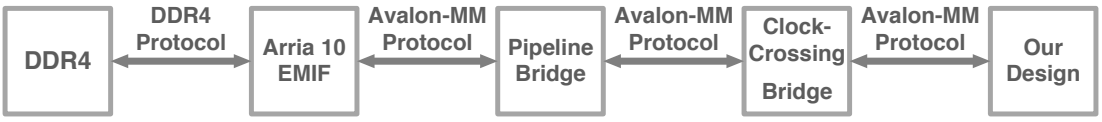

Fig. 4.4: 動態隨機存取記憶體的連接模組與協定示意圖，參考自 Intel 的操作手冊 [15]

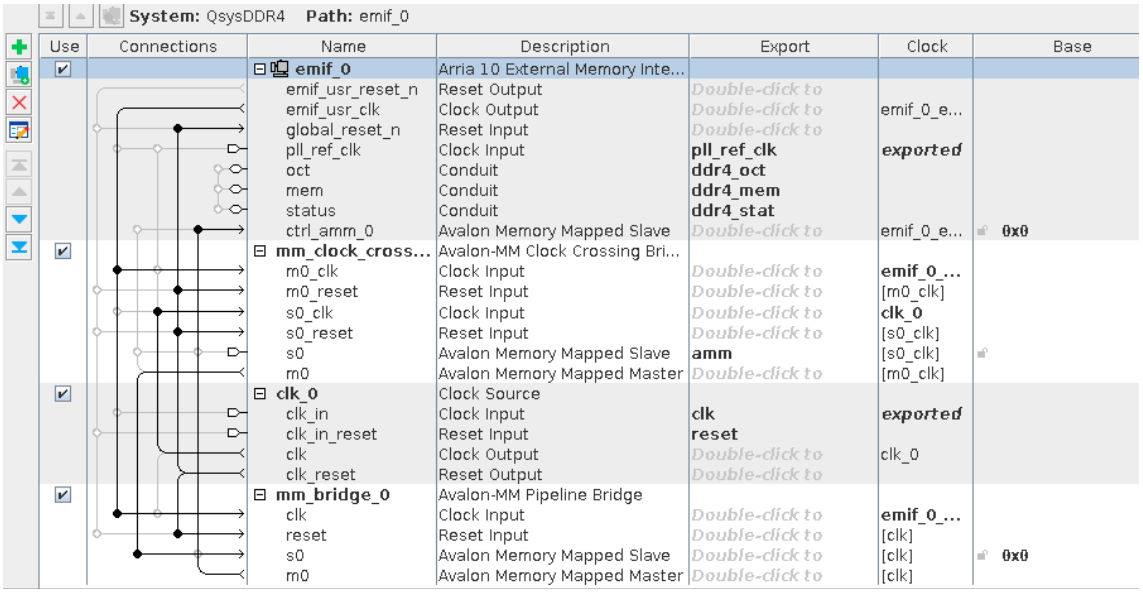

Fig. 4.5: Quartus 中動態隨機存取記憶體的 Qsys 接線介面

Avalon 記憶體映射由是 Intel 所制定、用於現場可程式化邏輯閘陣列上的一種高速傳輸模式，有讀取介面及寫入介面兩部分，可參考圖4.6與圖4.7的時序，而當中許多訊號的反應回傳時間並不固定，若送出讀取或寫入的要求，必須將部分值固定住不能變動，否則傳輸過程有可能會出錯。操作寫入介面時，需要先拉高 amm_wen 值，並同時送出並固定住寫入位址 amm_addr 以及寫入值 amm_wdata，等待傳回的 amm_wait 訊號從 1 拉低至 0，此時代表一筆寫入完成，可以接著進行

doi:10.6342/NTU202304510

下一筆傳輸操作；操作讀取介面時，需要先拉高 amm_ren 值，並同時送出並固定住讀取位址 amm_addr，等待傳回的 amm_wait 訊號從 1 拉低至 0，才可以改動下一筆 amm_ren 及 amm_addr，而需要等到 amm_rvalid 訊號從 0 拉高至 1，此時的讀取值 amm_rdata 才是有效的資訊，也因此送出的讀取位址當下不一定代表該位址的讀出值，需要自己另外設一個計數器以正確的使用讀出值。

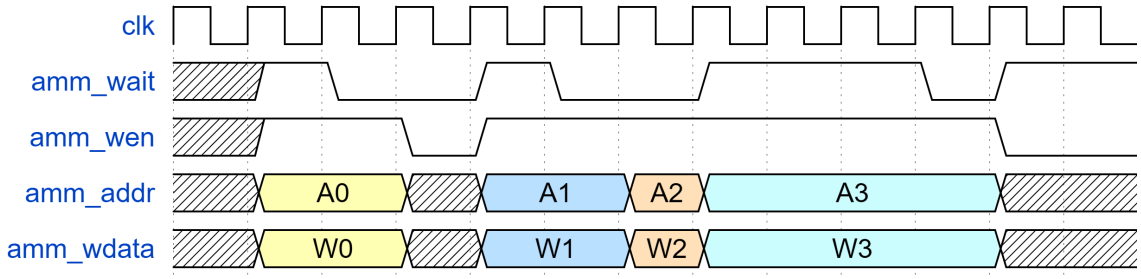

Fig. 4.6: Avalon 記憶體映射中寫入介面的波型示意圖，參考自 Intel 的操作手冊 [15]

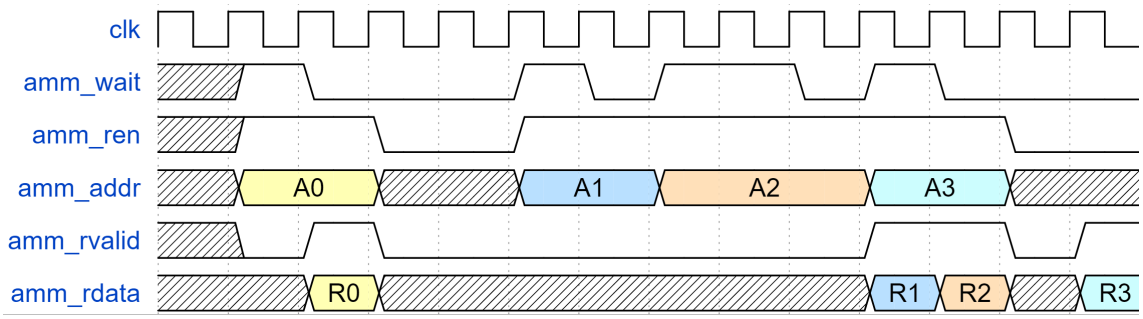

Fig. 4.7: Avalon 記憶體映射中讀取介面的波型示意圖，參考自 Intel 的操作手冊 [15]

在等待 amm_wait 訊號從 1 拉低至 0 所需的時間並非一個定值，需要一個或數個時脈週期才能得到，所以在使用 Avalon 記憶體映射時，我們可以使用暫存器或塊狀記憶體作為緩衝器，而也可以將動態隨機存取記憶體上的資料在閒置時預先讀取出來，在後續運用資料時才可以避免因無法預測時間而無法很好地管線化。

4.2.3 動態隨機存取記憶體的突發模式

本次使用的 DE5a-Net 板子上的動態隨機存取記憶體有一項規格為 SODIMM 64，其代表的意義是一次傳輸的位寬僅有 64 位元，對於整個系統的輸入和輸出是一個時脈週期 128 位元的規格來說，使用上會較不方便。我們可以使用動態隨機存取記憶體的突發模式 (Burst Mode) 來解決此狀況，其原理為連續進行多筆位址

doi:10.6342/NTU202304510

相鄰的傳輸，例如我們可以直接對某位址進行 8 倍突發傳輸，其代表會對該位址及往後的 7 個相鄰位址進行傳輸，可以更有效地利用傳輸時所需等待回傳的時間。而在本次的設計中，將突發模式的值設為 2 倍，便等效於一次傳輸的位寬為 128 位元，與輸入和輸出的 128 位元相符，對於在一開始將輸入資料存起來的設計來說是相當有幫助的。

4.2.4 環形緩衝器

我們從主機端將輸入資料傳入時並非一個時脈週期就能獲得所需資料，因此需要緩衝器來作為暫時保存資料時用的儲存空間，接著再傳入動態隨機存取記憶體。但在章節4.2.1中有提及塊狀記憶體的儲存空間不足的問題，因此當輸入資料過大時，將所有資料先儲存於塊狀記憶體是不可行的，因此我們可以使用環形緩衝器 (Circular Buffer) 作為輸入端與動態隨機存取記憶體之間的緩衝器。在經過實測後，平均而言從輸入端獲取一筆資料的所需時脈週期是高於將資料寫入動態隨機存取記憶體所需的時脈週期，將輸入端讀取的位址作為環形緩衝器結尾指標位置，而動態隨機存取記憶體的寫入位址作為環形緩衝器起始指標位置，當成功接收到一筆輸入端資料時，結尾位置向後方移一格，當成功輸入一筆資料進動態隨機存取記憶體時，起始位置向後方移一格，而當有任何位置超過了當前緩衝器的上限時，就回歸到從位置 0 開始，利用環狀結構的特性，便可以在使用有限的塊狀記憶體下，將所有資料由輸入端存進去動態隨機存取記憶體。

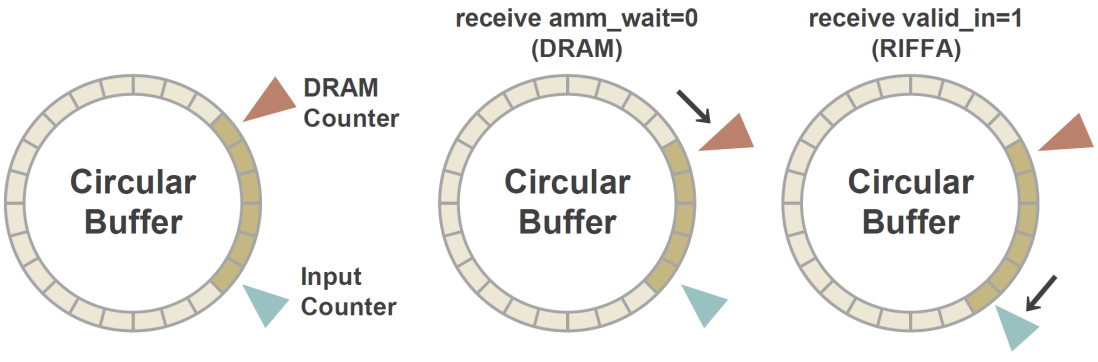

Fig. 4.8: 環形緩衝器運作原理示意圖

doi:10.6342/NTU202304510

4.2.5 記憶體映射及使用配置

在動態隨機存取記憶體中所儲存的資料有三種，參考基因組圖、檢索序列、以及檢索序列的長度資訊，而參考基因組圖需要較大的空間來儲存，所以使其在動態隨機存取記憶體上的位址的最前方 1 位元設為 0，檢索序列的位址最前方 2 位元設為 10，檢索序列長度資訊的位址最前方 2 位元設為 11，如此以來可以從動態隨機存取記憶體的位址最前方的不同位元來決定要對哪些資料做讀取或是寫入，整個動態隨機存取記憶體的使用配置可參考圖4.9。

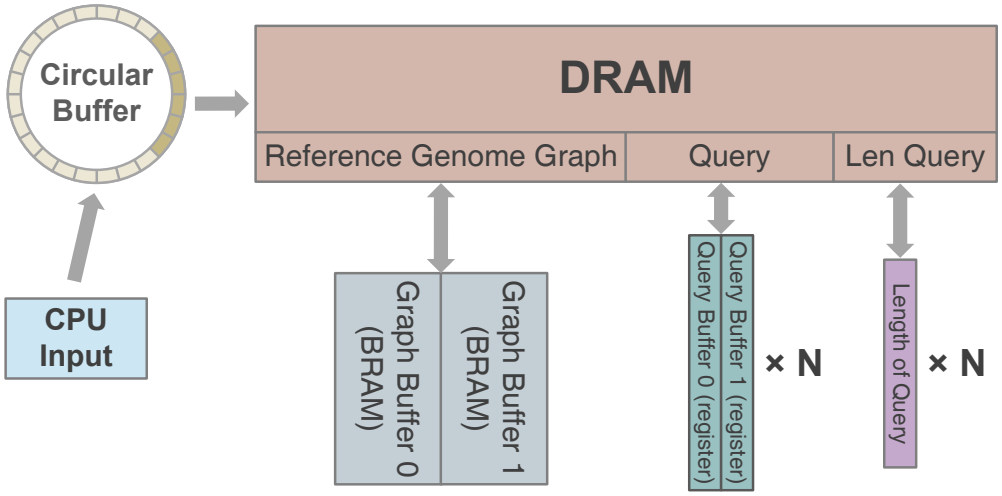

Fig. 4.9: 動態隨機存取記憶體的使用配置

對於參考基因組圖，在操作同一塊頁面子模組時都是相同的基因組圖，一塊頁面子模組將處理的參考基因組圖為長度 4096bp 的片段，而每個鹼基使用 10 位元來表達 (參考章節3.1)，因此同一塊頁面子模組會需要 4096×10 位元的儲存空間，我們使用 4 個塊狀記憶體作為緩衝器來儲存，而為了減少在進入下一頁面子模組時需要重新向動態隨機存取記憶體讀取下一塊參考基因組圖的時間，我們使用了兩倍的緩衝器 (Ping-Pong Buffer)，當一邊是當前的頁面子模組在操作讀取時，另一邊則可以開始預先寫入下一塊參考基因組圖，而當結束當前的頁面子模組後，另一邊緩衝器則改為操作讀取，原本這一邊則變為操作寫入，如此交替使用兩邊的緩衝器以節省向動態隨機存取記憶體讀取的時間，是以空間換取更短時間的策略。

對於檢索序列，從輸入端一次可以取得 1 筆 128 位元的資料，而一個鹼基片段可以用 2 位元來表達，因此一筆 128 位元的資料可以含有 64 筆鹼基資訊，於是

doi:10.6342/NTU202304510

我們需要讀取 $64 \times N_{parallel}$ 位元，而這 $N_{parallel}$ 筆檢索序列之間是需要同時讀取的，因此我們選用暫存器來儲存頁面子模組所需的檢索序列。與參考基因組圖使用雙倍緩衝器的原因相同，在檢索序列上我們也使用了兩倍的緩衝器，但將其從塊狀記憶體替換成了暫存器。

對於檢索序列的長度資訊，每筆檢索序列只需要一個長度資訊，而每筆檢索序列都需要將其結果輸出到主機端，我們可以利用在輸出的時間一邊向動態隨機存取記憶體發送讀取請求，因此在檢索序列的長度資訊上不需要使用兩倍的緩衝器，只需要利用暫存器存下當前檢索序列的長度資訊即可。

4.3 頁面子模組

在我們的設計中，頁面子模組為最主要能進行平行化的計算單元。在每次頁面子模組的重新刷新期間中，我們遵循由左至右、由上至下的運算方向，總共會處理 4096 個參考基因組圖的鹼基乘以檢索序列長度的鹼基，由主控制模組向動態隨機存取記憶體發送請求，以取得所需的參考基因組圖以及檢索序列。一旦取得這些資料，它們將被傳入頁面子模組中以計算分數，最終每個時脈週期的分數都被傳送至最大值尋找器中，直到所有需要計算的格子都完成計算後，將由主控制模組發出重新刷新訊號，通知頁面子模組可以接著進行下一張頁面的計算。

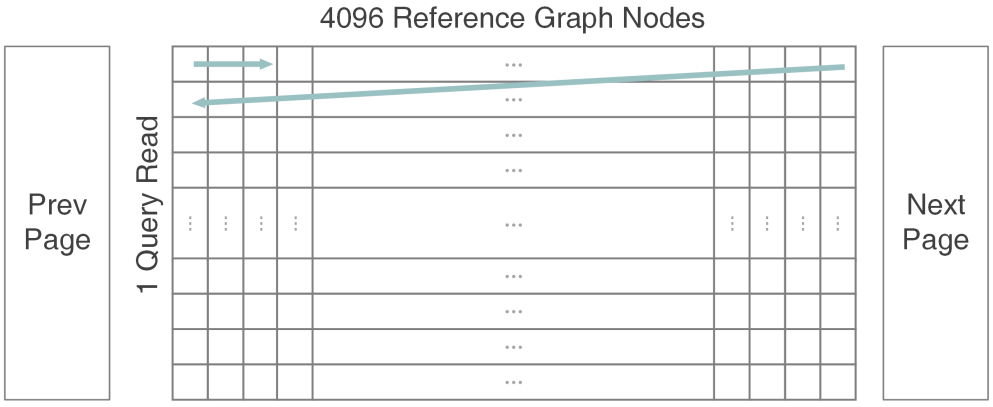

Fig. 4.10: 頁面子模組設計示意圖

doi:10.6342/NTU202304510

4.3.1 頁面子模組的規格說明

主控制模組向頁面子模組傳遞的信號結構如圖4.11所示，其中頁面子模組會平行化 $N_{parallel}$ 倍。這些頁面子模組會收到許多由主控制模組發送的控制訊號，用以重設、啟動、或是暫停；主控制模組會向動態隨機存取記憶體中取得所需的資料，並將當前處理中的參考基因組圖和檢索序列的片段發送至各個頁面子模組；對於剛重置的子模組，主控制模組也會向其中發送初始值。行或列的處理編號統一由主控制模組進行累加並發送，以確保每個平行化頁面子模組處理進度的一致性，同時節省運算資源。最終，各個頁面子模組輸出計算結果，將每一格的分數傳送至主控制模組。

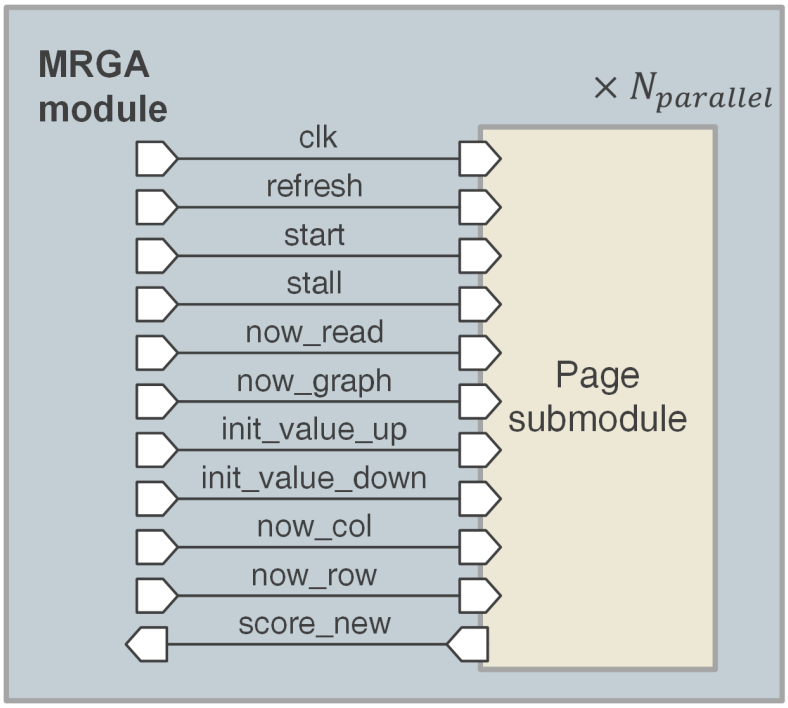

Fig. 4.11: 頁面子模組的規格示意圖

4.3.2 頁面子模組的硬體設計

在先前的章節中3.1，已經詳細討論了資料前處理的程序，其中每個格子的搜索範圍為向前查看 8 個鹼基的長度。而這幾個鹼基又會因為當前的參考基因組圖的前綴資訊，得出各自的匹配/不匹配得分、插入間隙得分、缺失間隙得分。最終，我們選擇最高分作為當前格子的評分。頁面子模組的設計具有以下特點：

doi:10.6342/NTU202304510

Table 4.3: 頁面子模組的輸入輸出規格描述

Port Name	I/O	Width	Description
clk	Input	1	系統時脈訊號。本設計為時脈的正緣觸發。
refresh	Input	1	同步重置訊號。當此訊號為高位準時模組重置。
start	Input	1	啟動訊號。當此訊號為高位準時模組啟動。
stall	Input	1	暫停訊號。當此訊號為高位準時模組暫停。
now_read	Input	2	當前檢索序列。對於每個頁面子模組會收到不同的檢索序列片段。
now_graph	Input	10	當前參考基因組圖。對於每個頁面子模組會收到相同的參考基因組圖片段。
init_value_up	Input	16	上列初始值。用以進行模組初始化。
init_value_down	Input	16	下列初始值。用以進行模組初始化。
now_col	Input	13	當前欄。
now_row	Input	15	當前列。
score_new	Output	16	分數計算結果。傳送計算完的分數至主控制模組。

1. 雙塊狀記憶體作為分數緩衝器

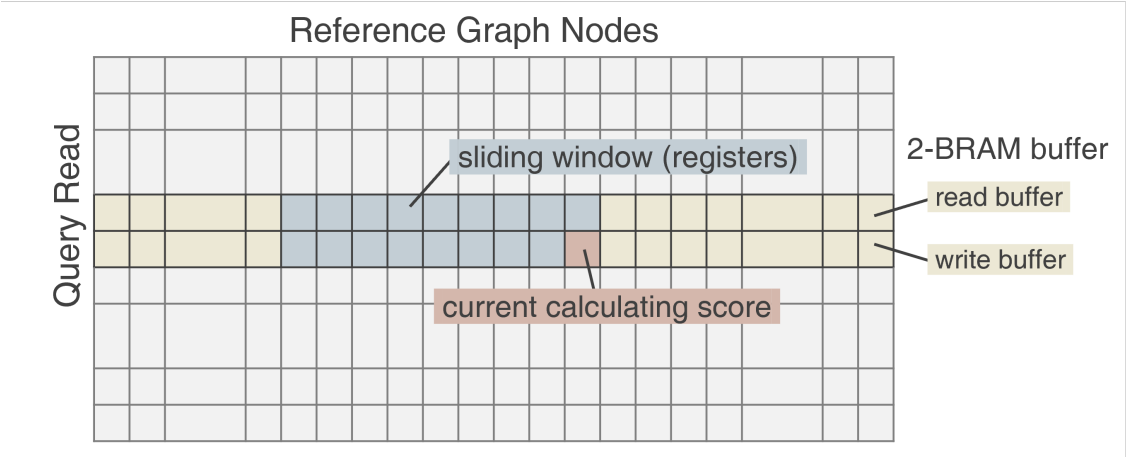

Fig. 4.12: 雙塊狀記憶體作為分數緩衝器示意圖

在分數的計算過程中，當前格子需要使用到當前列的前方 8 格、正上方列的格子及其前方 8 格的分數，這 17 格我們會以暫存器形式來處存，而我們還需要將正上方列的分數資訊儲存下來才能得到正上方列的分數。為了實現這一需求，我們採用了兩個塊狀記憶體的儲存方式。具體而言，在我們操作當前列時，我們將其中一條塊狀記憶體標記為寫入模式，同時另一條塊狀記憶體則是讀出正上方列的分數，供計算分數使用，示意圖可參考圖4.12；而當我們需要換列時，原本被標記為寫入模式的塊狀記憶體變為正上方列的記憶體，改為讀出模式以讀取正上

doi:10.6342/NTU202304510

方列剛存下來的分數，同時原本被用於讀取的塊狀記憶體則被清空，以準備作為寫入模式來接受新計算出分數的寫入。這樣交替使用兩個塊狀記憶體的設計，讓我們能夠在有限的儲存空間中完成對整張頁面的分數計算，無需將整張頁面所有格子的分數都儲存起來，如此一來，節省了儲存資源，有助於實現頁面子模組的更高倍率平行化設計。

由於分數所需的位元數較多，理論上需要檢索序列長度的位元數乘以匹配時的得分。為了有效地減少儲存空間的使用，我們將存入塊狀記憶體的值設計為與當前格子與當前列前一行格子的分數之間的差值，而非直接將分數存入。當我們需要讀取這些分數時，使用一個累加器來重新計算分數，將差值還原為原始分數。這樣的設計也能節省儲存空間，也維持了計算過程的效率。對照 Arria 10 上塊狀記憶體 M20k 的規格，位址使用 12 位元，即每列包含 4096 格是最能最大化利用一個塊狀記憶體的方式，也因此在一張頁面子模組的設計中，處理的參考基因組圖數量為 4096 個。因此在處理到下一格時，正上方列的暫存器會從前一列的緩衝器中讀出分數並累加，同時將剛剛算出的新分數存進去當前列的緩衝器，並且將暫存器中的值都平移一格，即完成一次分數計算的操作，關於換列時處理至下一格的操作可參考圖4.13

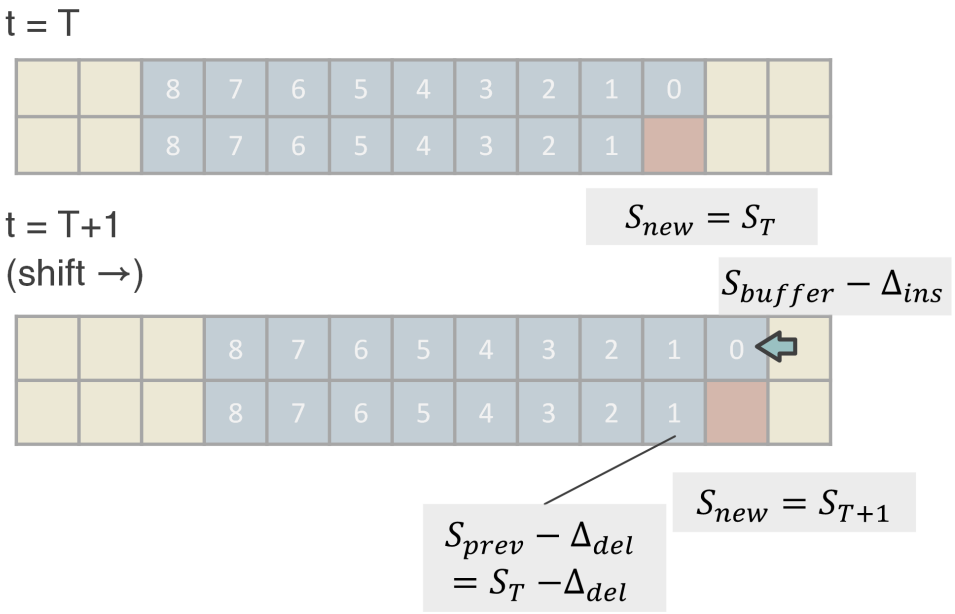

Fig. 4.13: 換列時對於記憶體及暫存器的寫入和讀出操作示意圖

藉由雙塊狀記憶體作為分數緩衝以及拆分成頁面的設計，原先需要 $O(Q \times R)$ 的空間複雜度來儲存整張動態規劃的表格，現在可以用兩個塊狀記憶體 (4096×2)

doi:10.6342/NTU202304510

來實現，降低了運作時所需的記憶體用量。

2. 頁面子模組中的管線化設計

使用現場可程式化邏輯閘陣列來實作硬體加速器，其一優勢在於可以針對許多部份做管線化設計 (Pipeline)。由前面章節的公式3.1中可以看到，相比於最原版的史密斯-沃特曼演算法只要比較 5 個值，在應用於基因組圖之史密斯-沃特曼演算法會需要比較 $3+2|E_j|$ 個值，其中 $|E_j|$ 代表參考基因組圖上第 j 行所擁有的來源邊數量。

在我們的硬體實作中，將來源邊數量設了上限 8 個，因此最多會需要比較 19 個值，也就是至少需要 $\lceil log_2(19) \rceil = 5$ 層比較器，並且比較器中還含有加上匹配/不匹配或是間隙的罰分，若要在一個時脈週期中完成計算，將會使加速器的頻率被此步驟限制，變為需時最長的路徑 (Critical Path)。因此我們藉由改變讀取和寫入分數緩衝器的位址，將這 19 個待比較的值拆分為三級的管線化。這個管線化操作會使每一列、也就是 4096 個時脈週期，多了 2 個時脈週期的延遲，但換來的是需時最長路徑可以大幅減少，進而提升整個系統的操作頻率。

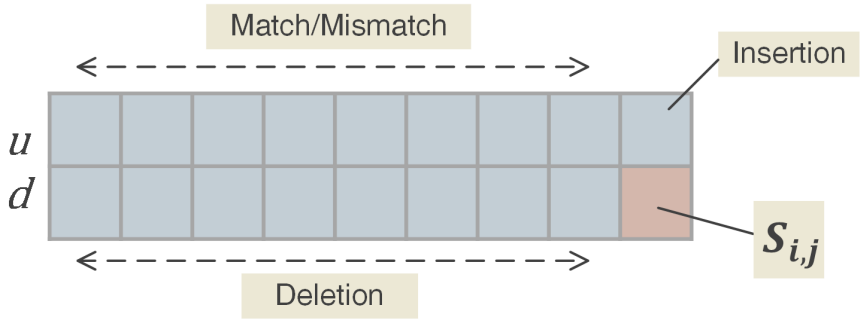

Fig. 4.14: 暫存器運用示意圖

在圖片4.14的說明中，我們以 u 表示上一列暫存器中的分數、d 表示當前列暫存器中的分數。由於當前列中的格子只會代表缺失間隙的分數，因此可以在 d 中都存入已經扣過缺失間隙罰分的分數 Δ_{del}，以節省運算資源和計算時間。在上一列中，正上方格子會是插入間隙的分數，而其左方的可能會是匹配或是不匹配的分數，因此正上方的格子向左方傳遞時，我們把插入間隙的罰分 Δ_{ins} 先加回，再扣除不匹配的罰分 $\Delta_{mismatch}$，而若是匹配的情況再加回 $\Delta_{mismatch} + \Delta_{match}$。

doi:10.6342/NTU202304510

3. 調整計算順序

觀察章節3.2中的演算法2，會發現第 7 行的迴圈中會去尋找所有相鄰的邊，若要在一個時脈週期完成此操作，會使資料路徑太長，無法很好的進行管線化設計。因此我們再把演算法的迴圈部分做調整，將計算的順序調整而不影響整體的演算法，最終形成更利於硬體加速器的架構。

Algorithm 3: (Hardware) Sequence-to-Graph Smith-Waterman Algorithm

Input: query sequence $Q[1...m]$ and reference graph $G(V', E')$ with total n nucleotides
Output: best alignment score S_{max}

1 Initialization: $S[0, ...] \leftarrow 0, S[..., 0] \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** m **do**
3 **for** $j \leftarrow 1$ **to** n **do**
4 $\Delta_{i,j} \leftarrow (Q[i] = V'[j])\ ?\ \Delta_{match} : \Delta_{mismatch}$
5 $S[i,j] \leftarrow max(0,\ \Delta_{i,j})$
6 $S[i,j] \leftarrow max(S[i,j],\ S[i-1,j] + \Delta_{ins})$
7 **for** $k \leftarrow 1$ **to** N **do**
8 **if** $(j-k,\ j) \in E'$ **then**
9 $S[i,j] \leftarrow max(S[i,j],\ S[i-1,k] + \Delta_{i,j})$
10 $S[i,j] \leftarrow max(S[i,j],\ S[i,k] + \Delta_{del})$
11 $S_{max} \leftarrow max(S_{max},\ S[i,j])$
12 **end**
13 **end**
14 **end**
15 **end**
16 **return** S_{max}

修改後的演算法可參考演算法3，其中 V' 和 E' 為經過章節3.1.2中前處理操作後的節點和邊，而 N 即為設定的來源邊上限，在本設計中 $N = 8$。在中間 For 迴圈的改動 (第 7 行) 帶來的好處是改變了查找來源邊的方式，對於硬體而言即是從一個大小為變量的迴圈改回大小為定值的迴圈，減少數據依賴 (Data Dependency) 的情況發生，進而使這一部分的演算法能夠管線化。

4. 頁面子模組間的銜接

在本設計中，每個頁面子模組會計算 4096 個鹼基。然而，實際的參考基因組圖往往包含遠遠超過 4096 個鹼基，這帶來了頁面子模組間的銜接問題。我們在每次計算完一列後，將該列的尾端分數以塊狀子模組的形式儲存起來。在下一輪使

doi:10.6342/NTU202304510

用頁面子模組時，我們再度取出尾端的分數，並將新的分數重新寫入。

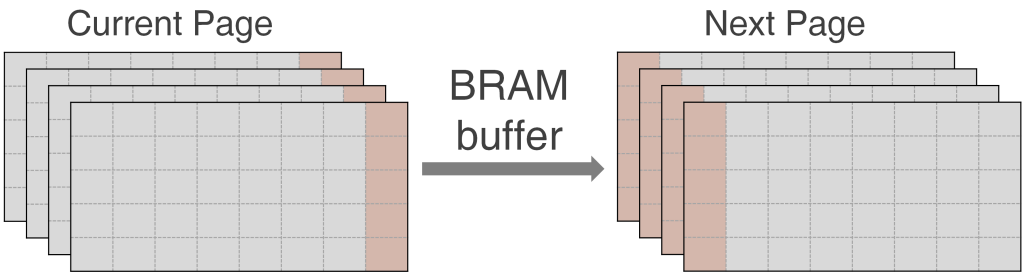

Fig. 4.15: 以塊狀記憶體作為頁面子模組間銜接示意圖

在這個設計中，為了節省儲存空間，我們採用將上下列分數的差值儲存起來，而非儲存實際的分數。而在讀出分數時，需要在讀取時進行差值的累加，這樣的設計以少量的運算資源來換取大幅減少儲存需求，使我們在有限的硬體資源下可以完成頁面子模組間的銜接。

doi:10.6342/NTU202304510

Chapter 5

實驗與結果分析

在這個章節中，將會列出實驗所使用的資料，並呈現把硬體設計燒錄至現場可程式化邏輯閘陣列的實驗結果與分析，當中也包含所使用的硬體規格及資源用量。

5.1 實驗測試資料

5.1.1 參考基因組圖

前面章節2.3中有提到，參考基因組圖是由參考基因組序列以及基因變異經過vg toolkit[7] 來生成，我們使用真實世界的資料來產生實驗用的參考基因組圖。而本次的設計目標為處理變異較多的區域，因此我們會以變異較多的區域為挑選的目標。

首先是當作基底的參考基因組序列，我們使用了三筆著名的基因變異較多區間，由小到大列出分別是乳腺癌一號基因 (Breast Cancer Type 1, 簡稱 BRCA1)、白血球受體複合體 (Leukocyte Receptor Complex, 簡稱 LRC)、主要組織相容性複合體 (Major Histocompatibility Complex, 簡稱 MHC)，而這三筆參考基因組序列都是來自人類的基因 (GRCh38)，關於這三段變異較多區域的詳細資料可以參考表格5.1。

而作為基因變異的來源，是參考自千人基因組計劃 (1000 Genomes Project) 的 Phase 3[16]，這是一個自 2008 年啟動的大型研究項目，匯集了來自不同國家和地區的科學家和研究機構，目標是識別人類基因組中的各種變異，科學家們可以藉

doi:10.6342/NTU202304510

Table 5.1: 變異較多區域的詳細資料

Region	Chromosome	Length (bp)	Coordinate (GRCh38)	Number of Genes
BRCA1	17	81,189	43,044,293-43,125,482	1
LRC	19	1,058,685	54,025,633-55,084,318	35
MHC	6	4,970,458	28,510,119-33,480,577	172

著這個計劃更好地理解基因與疾病之間的關聯，以及個體間的基因差異。此計劃通過採集和定序超過一千名受試者的 DNA 樣本，代表了不同地理區域和族群，從而提供了更全面的基因組資料。因此，本次實驗的測試資料以此作為基因變異的來源。

5.1.2 短片段檢索序列

短片段檢索序列為 Illumina 所產出的序列，其長度為固定值，並以 150bp 最為常見，我們使用 Mason2[17] 來模擬短片段檢索序列，將上述三個參考基因組圖做為基底，並固定模擬長度為 150bp 以生成測試資料。

Illumina 的短片段序列中有分為單端定序 (Single End) 與雙端定序 (Paired End)，這是從一個方向或是雙向都開始做定序的差別，雙端定序可以把經過定序和比對完的資料再做額外的處理，延伸成更長的序列，使等效定序出的長度為兩倍。在本次實驗中模擬的參數雖為雙端定序，但並未處理將兩邊合併，因此是將雙端定序視為兩個單端定序來操作。

5.1.3 長片段檢索序列

長片段檢索序列為 PacBio 所產出的序列，與短片段檢索序列不同的是，長片段的技術會產生長度不一且錯誤率較高的檢索序列，我們使用 PacBio Simulator(簡稱 PBSIM)[18] 來模擬長片段檢索序列，設定其模擬的模型為連續長片段 (Continuous Long Read, CLR)，並生成兩種平均長度的長片段檢索序列，分別是 10Kbp 及 25Kbp。

使用 CLR 模型產生出來的結果，準確率平均值為 0.781643，標準差

doi:10.6342/NTU202304510

0.018141，其中不匹配的發生率為 0.021788，插入間隙的發生率為 0.131215，缺失間隙的發生率為 0.06607。

5.2 現場可程式化邏輯閘陣列

我們使用了 Intel 的 DE5a-Net Arria 10 現場可程式化邏輯閘陣列作為硬體加速器，並以 Intel Quartus Prime 進行編譯。

5.2.1 執行頻率調整

在原本可使用動態隨機存取記憶體的 Quartus 模板中，必須與 PCIe 接口在 125MHz 或 250MHz 的頻率下一起運作，使整個硬體加速器的頻率也必須和 PCIe 的頻率一致。然而，使用 DE5a-Net 在頻率為 250MHz 下實現相對複雜，即使經過高度精緻的管線化設計，使用 Quartus 合成時仍然存在約 -0.4 奈秒的緩衝時間不足 (Negative Slack Time)。對於 125MHz 的頻率而言，則有比較寬裕的條件，可以輕鬆實現穩定的運作。

因此，在 Qsys 中，我們增加了鎖相迴路模組 (Phase-Locked Loops, PLL)，使其可以產生額外的時脈週期，達到高於 125MHz 的工作頻率，以提升整體加速性能。值得注意的是，有兩個主要的時間上瓶頸：首先是在頁面子模組中的比較器，即使經過管線化設計，仍然包含了兩層比較器；另一部分是和塊狀記憶體的溝通也佔用相當多的時間，因為現場可程式化邏輯閘陣列是有固定的位置來放置塊狀記憶體，當塊狀記憶體的使用率較高時，可能需要更長的佈線路徑，使延遲增加。有關如何在 Qsys 中使用鎖相迴路模組的詳細資訊，可以參考圖5.1

最終，我們成功地運用鎖相迴路模組，實現了頻率 200MHz 的時脈，並且使最差條件下 (Worst Case) 的緩衝時間皆為非負值，這代表著我們的硬體加速器能夠在各種情況下穩定運作。表格5.2是以 BRCA1 的不同數量短片段序列作為測試，並且只比較 Forward 階段，以體現加了鎖項迴路模組後頻率提升帶來的加速，可以觀察到在速度上有 1.6 倍的提升，並且接下來的測試中都將會以 200MHz 作為測試比較對象。

doi:10.6342/NTU202304510

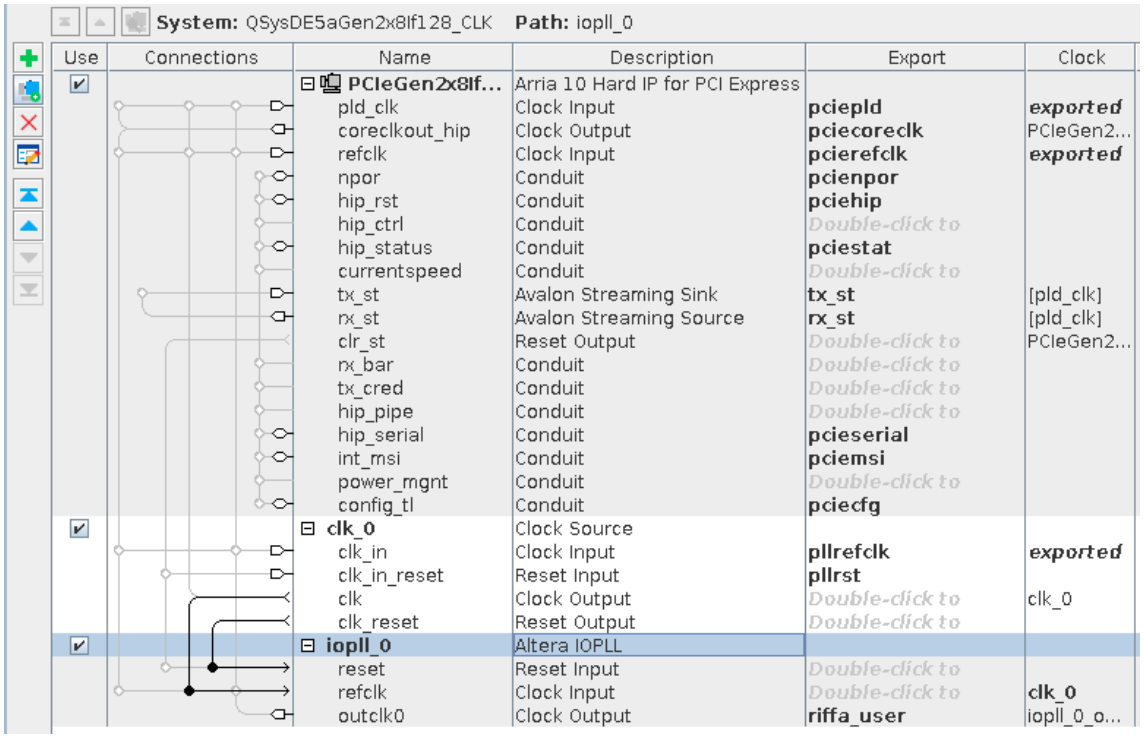

Fig. 5.1: 在 Qsys 中使用鎖相迴路模組的接線圖

Table 5.2: 固定檢索序列長度的測試資料 (時間單位: s)

Length (bp)	Number of Reads	200MHz	125MHz	Speed-Up
150	1024	1.03320	1.65431	1.60x
	10240	10.3309	16.5385	1.60x
	102400	103.307	165.383	1.60x

5.2.2 硬體資源用量

表5.3為使用 Quartus Prime 合成完的資源使用量，而功耗部分則是由 Quartus Powerplay 測量而得。

在 DE5a-Net Arria 10 上的塊狀記憶體是由小記憶體塊 M20k 所構成，它可以由不同的資料位寬和深度所構成，本設計中常用的組合為 4×4096 和 32×512，盡量使塊狀記憶體可以最大化的利用。然而，若要最大化的運用 M20k，必須使用到寬度為 5×4096 和 40×512，由於資料在硬體加速器中通常是以 2 的冪次方來儲存，因此要使用滿一塊 M20k 並不容易，對於每個 M20K 而言，我們常用的組合只用到了其中的 16Kbit。因此在表格中的用量方面，顯示的用量為 55%，雖然看似還有許多空間可以運用，但實際上已經使用了 1,871 個 M20k 記憶體，佔了總數的 69%，並且因為頻率較高會導致難以低延遲地佈線，因此平行化的倍率

doi:10.6342/NTU202304510

Table 5.3: 在 DE5a-Net Arria 10 的硬體資源用量

Device	Intel Arria 10 10AX115N2F45E1SG	
Frequency	200 MHz	
	Resource Usage	Utilization
Logic (ALMs)	137,506	32%
LABs	18,737	44%
Registers	176,747	20%
Block Memory Bits	30,345,936	55%
Power	13.205 W	

受限於塊狀記憶體的數量與佈線路徑延遲，最終使用了現版本的 128 倍平行化設計。

此外，因為本設計中不涉及浮點數或其他複雜函數的計算，因此不會用到數字信號處理器 (Digital Signal Processor, DSP)，而是以邏輯陣列方塊 (Logic Array Block, LAB) 來合成所需的運算，而這些陣列方塊的運作方式是使用查找表 (Look-Up Table, LUT) 來組成的，我們計算以定點數為主，因此主要用到的是這些陣列方塊資源。

5.3 實驗設計

5.3.1 實驗平台

我們的現場可程式化邏輯閘陣列使用的是 DE5a-Net Arria 10，而這片板子是置於 Intel i7-4790 的主機上，使用的是 CentOS 7 的系統。由於硬體加速器的計算時間主要在板子上，因此使用的主機平台不需要選用較新型。

而我們的比較對象 PaSGAL[12]，我們以 Intel i7-13700 的主機來運作，搭配 64GB 的記憶體，使用的是 Ubuntu 20.04.4 LTS 的系統。此處理器為 Intel 13 代的架構，共有 8 大核 (Performance-Core) 以及 8 小核 (Efficient-Core)，總共有 24 個執行

doi:10.6342/NTU202304510

緒 (Threads)，也就是說我們可以在此環境中運行 24 緒的 PaSGAL 軟體。PaSGAL 是純軟體的程式，因此我們以較新的處理器作為比較的標準。

5.3.2 前處理時間與來源邊個數討論

對於每筆參考基因組圖，我們都需要經過一次性的前處理操作3.1，將其轉為適合送入硬體加速器的格式3.2，在這裡我們列出對於三筆參考基因組圖的前處理花費時間。這些參考基因組圖經過前處理後，可以先另外儲存起來，若需要對不同的檢索序列來進行比對，可以直接取出已經儲存起來的基因組圖，而不需要再次進行前處理。

Table 5.4: 資料前處理的花費時間 (單位: ms)

Region	Preprocessing Time
BRCA1	22.59
LRC	248.40
MHC	1090.47

表格5.5統計了各筆資料中，若採用不同位元數來做為來源邊上限，未涵蓋到相對應來源邊的發生率。最終我們選用與軟體中相同的 8 位元作為前處理的來源邊上限。

Table 5.5: 資料前處理採用不同位元數未涵蓋到完整數量來源邊的發生率

Region	6-bit	8-bit	10-bit	16-bit
BRCA1	0.251%	0.218%	0.151%	0.050%
LRC	0.194%	0.132%	0.109%	0.049%
MHC	0.241%	0.170%	0.128%	0.058%

5.3.3 測試資料設計

為了做不同情況下的執行時間比較，我們設計了兩種比較情況：

doi:10.6342/NTU202304510

1. 固定檢索序列數量

首先是固定檢索序列數量的測試資料，檢索序列平均長度為 150bp、10Kbp、25Kbp，於三張參考基因組圖上的測資。生成的檢索序列長度及其標準差如表5.6所示，其中 150bp 的是使用 Mason2 所模擬的短片段序列，而另外兩種長度為使用 PBSIM 所模擬的長片段序列。

Table 5.6: 固定長度檢索序列數量為 1024 的測試資料

Region	Dataset	Mean Length (bp)	SD	Number of Reads
BRCA1	B1	150	0	1024
	B2	10856.02	401.33	
	B3	25698.85	495.76	
LRC	L1	150	0	1024
	L2	10848.12	553.81	
	L3	25788.68	1722.02	
MHC	M1	150	0	1024
	M2	10848.12	553.81	
	M3	25854.15	1682.53	

2. 固定覆蓋率

另一組測試資料則是固定模擬的覆蓋率 (Coverage)，或稱為深度 (Depth)，其代表的是對同一個參考基因組圖上的某個位置測序的次數，深度越高代表在這個位置被測序的次數越多，提高測序數據的可靠性和準確性。由於 BRCA1 的長度較短，因此採用 40x 的覆蓋率，LRC 使用的是 10x 的覆蓋率，而 MHC 由於資料量較大，因此在本實驗中以 1x 的覆蓋率進行速度的評估。生成的檢索序列長度及其標準差如表5.7所示。

5.4 硬體加速倍率分析

5.4.1 固定檢索序列數量

在固定檢索序列數量的測試中，測試的時間如表5.8，從中我們可以觀察到幾個現象：

doi:10.6342/NTU202304510

Table 5.7: 固定覆蓋率的測試資料

Region	Dataset	Mean Length (bp)	SD	Number of Reads	Coverage
BRCA1	B40x_1	150	0	22528	40x
	B40x_2	8787.35	373.36	384	
	B40x_3	26420.71	1041.85	128	
LRC	L10x_1	150	0	73344	10x
	L10x_2	9604.46	367.63	1152	
	L10x_3	28630.30	1172.67	384	
MHC	M1x_1	150	0	34304	1x
	M1x_2	10040.99	340.40	512	
	M1x_3	20106.26	1211.60	256	

Table 5.8: 固定檢索序列數量的測試結果 (時間單位: s)

		PaSGAL		Our Design		Speed-Up		
		FW	BW	FW	BW	FW	BW	Total
BRCA1	B1	6.716	3.222	1.033	0.517	6.50x	6.24x	6.41x
	B2	493.756	235.563	75.250	37.910	6.56x	6.21x	6.45x
	B3	1174.060	555.737	178.051	89.900	6.59x	6.18x	6.46x
LRC	L1	115.257	55.596	13.233	6.617	8.71x	8.40x	8.61x
	L2	7984.18	3825.93	962.28	484.00	8.30x	7.90x	8.17x
	L3	18844.90	9161.07	2289.92	1153.51	8.23x	7.94x	8.13x
MHC	M1	604.80	292.20	61.74	30.87	9.80x	9.46x	9.69x
	M2	44176.9	21316.5	4489.45	2258.08	9.84x	9.44x	9.71x
	M3	102783.0	49215.9	10702.85	5396.81	9.60x	9.36x	9.44x

首先是在 Forward 階段的加速倍率優於 Backward 階段一些，以下是造成此現象的原因推測：從表格中可以觀察到在我們的設計中，Backward 階段通常稍慢於 Forward 階段時間的一半，這是因為在資料傳輸時，兩個階段所需傳輸相同大小的參考基因組圖，並且需要處理多一倍檢索序列的儲存，而傳輸及與記憶體溝通時間佔了一小部分，使得 Forward 階段時間的一半是比 Backward 階段還要短一些。另一方面，由於 PaSGAL 對於參考基因組圖的調度上不同於我們的硬體加速器，PaSGAL 無需經過傳輸預先儲存起整張參考基因組圖，而是根據需求調度部分圖，所以在 Backward 階段中，PaSGAL 反而因為需要調度的量較小，減少了與主機上的動態隨機存取記憶體的溝通，因此在 PaSGAL 中，Forward 階段時間的一半是比 Backward 階段還要略長。總結來說，由於軟體和硬體加速器對於資料的調度方式不同，整體而言，Forward 階段的加速倍率通常高於 Backward 階段，對於加速倍率的視覺化比較可參考圖5.2

doi:10.6342/NTU202304510

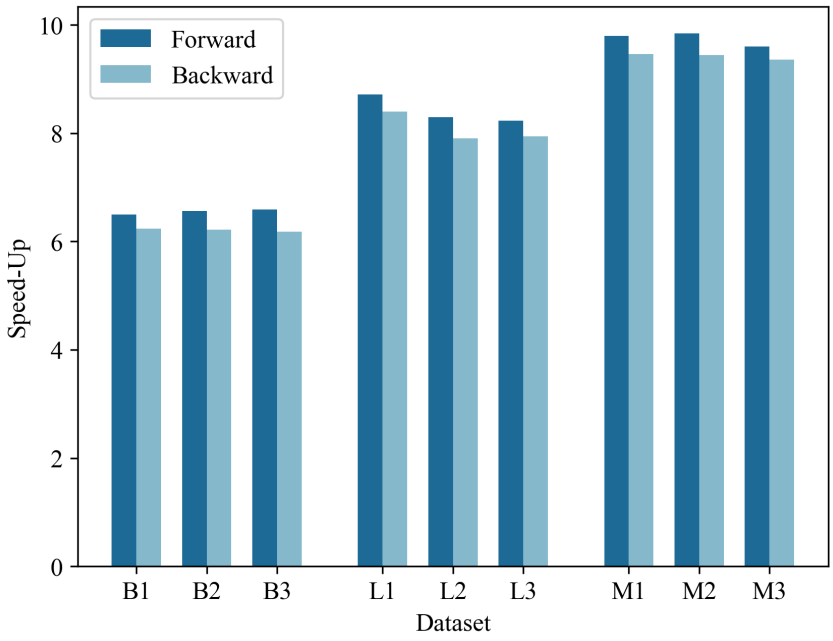

Fig. 5.2: 固定檢索序列個數下，各筆資料的加速倍率比較

接著是同一張參考基因組圖下，三種不同的檢索序列長度，對於加速倍率而言，基本相差不多。這是因為在面對不同長度的檢索序列時，操作動態規劃表格時的縱軸方向上策略相似，都是先經過切成小區塊來操作，因此隨著檢索序列的長度漸增，我們的設計和 PaSGAL 都是接近相同比例時間成長。總結來說，在操作同一張參考基因組圖，操作時間幾乎是正比變長，因此加速倍率上基本持平。從圖5.3中也可以看出，對於每一筆資料的三個點，幾乎是連成一直線的，代表最主要計算的動態規劃表格呈現隨著檢索序列長度漸增，執行時間成正比成長 (圖表的橫軸縱軸為對數軸)。

最後，在不同張參考基因組圖的情況下，在越大的參考基因組圖上，其加速倍率越顯著。這是因為我們硬體加速器有頁面子模組的設計，在動態規劃的橫軸方向也有切成小區塊來操作，並且有預讀取下一小塊所需的參考基因組圖，這部分相較於 PaSGAL 而言，我們在處理較大的參考基因組圖上可以加速倍率較高。總結來說，在操作較大的參考基因組圖時，我們的設計是更有加速倍率上的優勢。

doi:10.6342/NTU202304510

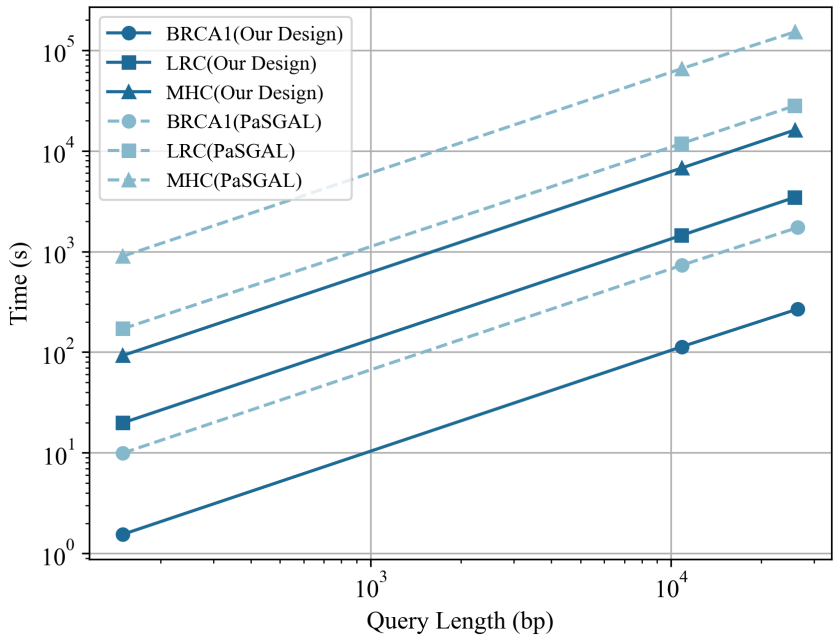

Fig. 5.3: 固定檢索序列個數下，各筆資料的執行時間比較

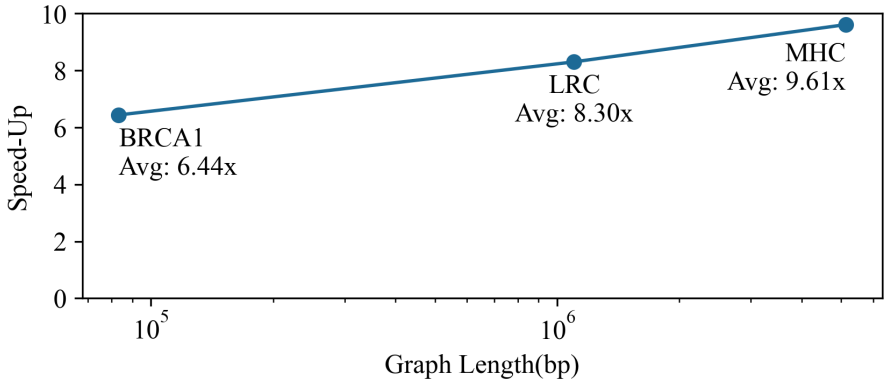

Fig. 5.4: 固定檢索序列個數下，各筆資料的平均加速倍率比較

Table 5.9: 固定覆蓋率的測試結果 (時間單位: s)

		PaSGAL		Our Design		Speed-Up		
		FW	BW	FW	BW	FW	BW	Total
BRCA1	B40x_1	155.223	73.1961	22.7277	11.3640	6.83x	6.44x	6.70x
	B40x_2	154.656	73.8316	22.9600	11.6368	6.74x	6.34x	6.60x
	B40x_3	155.485	75.4242	23.2294	12.0178	6.69x	6.28x	6.55x
LRC	L10x_1	7794.57	3874.33	947.756	473.879	8.22x	8.18x	8.21x
	L10x_2	7845.88	3876.20	956.594	480.018	8.20x	8.08x	8.16x
	L10x_3	7953.18	3878.82	956.571	481.859	8.31x	8.05x	8.23x
MHC	M1x_1	19778.8	9491.16	2068.08	1034.04	9.56x	9.18x	9.44x
	M1x_2	19168.1	9534.80	2081.55	1048.74	9.21x	9.09x	9.17x
	M1x_3	19877.1	9834.68	2123.47	1092.00	9.36x	9.01x	9.24x

doi:10.6342/NTU202304510

5.4.2 固定覆蓋率

在固定覆蓋率的測試中，測試的時間如表5.9，從中我們可以觀察到幾個現象：與固定檢索序列數量的結果相同，我們依然可以得出三點結論：Forward 階段加速倍率優於 Backward 階段，在同一張參考基因組圖的不同檢索序列長度加速倍率持平，參考基因組圖越大加速倍率越顯著，其中第二點更能從表格上看出每一張參考基因組圖內的各筆資料執行時間大致相同。

在固定覆蓋率的實驗中，對於每一張參考基因組圖而言，所測量的鹼基數量大致相同，因此可以直接從執行時間中直接觀察出從短片段到長片段檢索序列的改變。在這裡我們引入 GCUPS 的概念，其代表的是每秒鐘所執行的運算數量 (Giga Cell Updates Per Second)，其定義如公式5.1：

$$GCUPS = \frac{Length_{graph} \times Length_{query} \times N_{query}}{Total\ Execution\ Time} \tag{5.1}$$

其中 $Length_{graph}$ 為參考基因組圖的總鹼基數，$Length_{query}$ 為檢索序列的平均長度，N_{query} 為檢索序列的個數，分母則是執行時間。對於同一張參考基因組圖，在同樣覆蓋率的情況下，其分子的三項相乘會是相同的值，我們分別把每筆資料的 GCUPS 製成長條圖5.5，可以看出在不同測試資料中，我們的硬體加速器維持在 25 GCUPS 左右，而 PaSGAL 的 GCUPS 則是隨著參考基因組圖的大小漸增而減少。

5.5 其他討論

5.5.1 功耗比較

現場可程式化邏輯閘陣列的功耗表現上是十分優異的，從前述的規格圖當中可以觀察到在 200MHz 下的功耗為 13.205W，而另外我們也對比了 125MHz 下的功耗為 11.148W，在頻率變高的同時，功耗雖有增加，但若乘上總執行時間來看，總共消耗的能量會是更少的，因此將頻率提升是對於硬體加速器而言有諸多好處的。

另一方面，我們也用了 CPU Energy Meter[19] 來測量 PaSGAL 的功耗，而這

doi:10.6342/NTU202304510

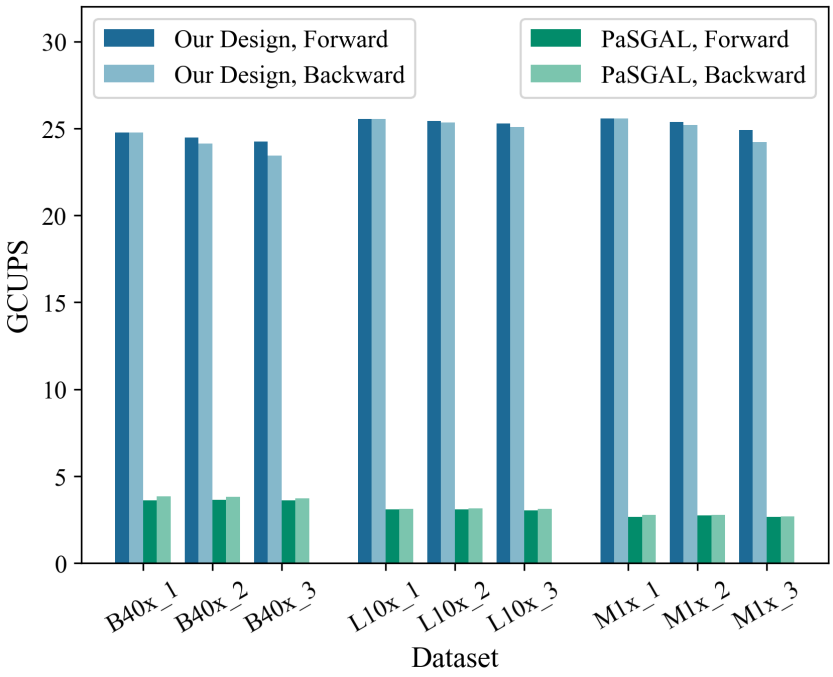

Fig. 5.5: GCUPS 比較

個工具顯示的是瞬時功率，因此我們分別測量在運行軟體下和閒置下的功耗，相減而得它的運作功耗。

整理的功耗比較表格如表格5.10所示：

Table 5.10: 功耗比較表格

	FPGA		**PaSGAL**
Language	Verilog-HDL		C++
Frequency	125 MHz	200 MHz	4.10 GHz
Power	11.148 W	13.205 W	178 W

5.5.2 記憶體使用量比較

Table 5.11: 記憶體使用量比較表格

	FPGA		**PaSGAL**
Memory Usage	BRAM	DRAM	1.33 GB
	30.3 Mb	0.125 GB	

以參考基因組圖為 MHC 的情況來討論，我們可以用 Linux 中的指令來查看 PaSGAL 的記憶體用量，得到的結果是運行時使用了 1.33GB。

doi:10.6342/NTU202304510

而對於我們的硬體加速器可以分成兩部分來討論，其一是塊狀記憶體，已於前述的硬體資源用量中討論過其用量為約 30.3Mb；動態隨機存取記憶體方面，由於硬體是必須先規劃一塊位址來儲存資料，也就是說就算未使用滿這塊位址，也視作已經使用，我們總共用了寬度 128bit 乘上 28bit 位址的記憶體，也就是最大情況會使用到 0.125GB 的記憶體，以實際情況來說，這 0.125GB 用量分為參考基因組圖與檢索序列，分別都是各占一半用量，參考基因組圖部分至多可以容納 12,582,912 個鹼基，以易變異區域的大小而言，這個值是足夠大的，使用的測試資料中 MHC 是相當大的一個區域，包含了 172 個基因在內，其鹼基數約為 5,000,000bp，不到上限值的一半；檢索序列部分，可以容納的大小為 8,388,608 條長度 256bp 以下的短片段序列，或者是 65,536 條長度 32,768bp 以下的序列。

doi:10.6342/NTU202304510

doi:10.6342/NTU202304510

Chapter 6

結論與展望

6.1 結論

本篇論文提出了一個參考基因組圖的非啟發式序列比對硬體加速器，藉由將參考基因序列拓展至參考基因組圖，可以將基因的多樣性也納入序列比對中，適合應用於基因變異較多的區域。此加速器可以平行化一次處理多個序列對同一張參考基因組圖的比對，而檢索序列可為短片段序列或長片段序列，並且可以處理輸入的各段檢索序列長度相異的情況。

本篇論文提出的硬體加速器設計架構，將參考基因組圖拆分為小塊的頁面子模組，並運用兩條塊狀記憶體來作為分數計算的緩衝，減少了運算中的記憶體需求，使我們能達到 128 倍的平行化計算；此外，我們運用了動態隨機存取記憶體與相對應的緩衝器設計，用以存放較大的資料並且可以重複取用，使我們可處理的參考基因組圖上限達到 12,582,912 個鹼基，並且長度 256bp 以下的短片段序列至多處理 8,388,608 條、長度 32,768bp 以下的長片段序列可以處理至多 65,536 條。

在速度方面，我們透過鎖相迴路模組使操作頻率達到 200MHz，此時非啟發式的序列比對可以達到 25GCUPS，對比功能設計的純軟體程式 PaSGAL[12]，視參考基因組圖由小到大可達 6.44 倍至 9.61 倍加速。並且在記憶體使用量與能耗方面上也有更好的表現。

doi:10.6342/NTU202304510

6.2 未來展望

在硬體加速器方面，我們無法再做更高倍率平行化主要是受限於現場可程式化邏輯閘陣列的塊狀記憶體數量，並且即使經過高度管線化設計仍無法將頻率從200MHz 提升至更高，因此若有資源更加充足、製程更加先進的加速器平台，我們可以達到更高的平行化倍率以及更高的操作頻率，使計算速度更加提升。

在生物資訊學的應用方面，本篇的目標參考基因組圖是基因變異較多的區域，因此採用了非啟發式的比對設計，以較多的運算時間換取較精準的計算結果，但是在這種演算法底下仍無法處理較複雜的圖狀結構，例如拷貝數變異(Copy Number Variation, 簡稱 CNV) 或是超長串聯複製 (Extra-Long Tandem Repeats, 簡稱 ETRs)，若有更佳的演算法能夠整合進此硬體加速器中，在面對變異較多區域的研究能夠有更廣的應用。

doi:10.6342/NTU202304510