Parallel BioInformatics

Sathish Vadhiyar

Parallel Bioinformatics

- Many large scale applications in bioinformatics – sequence search, alignment, construction of phylogenetic trees
- Many operations can be trivially parallelized

Sequence Alignment

Sequence Alignment/Comparison

- Sequences – to relate molecular structure and function to the underlying sequence
- Sequence comparison - to find similarity of species
- DNA and protein sequences can be treated as strings over a fixed alphabet of characters (C,G,A,T)
- Aligning 2 sequences to match characters from the sequences that lie in the same position

Terms

- Gaps – introduced in either of the sequences to deal with missing characters
- Scoring function – to evaluate an alignment
- Goal – find an alignment with the best score

Terms

- Affine gap penalty function: for a maximal consecutive sequence of k gaps, a penalty of h+gk is applied.
- First gap in a consecutive gap sequence is h+g, while the rest of the gaps are charged g each

Example

- Scoring function, f:
  - f(c1,c2) = 1 if c1=c2, 0 otherwise
- Gap penalty for a gap sequence of length k = 2+k

A	T	G	–	–	T	C	G	A
A	–	G	A	A	T	C	T	A
1	−3	1	−4		1	1	0	1

The Algorithm

- A=a1,a2,…,am and B = b1,b2,…,bn
- Let m <= n
- **Dynamic programming** to find an optimal alignment of A and B
  - Solution to a larger problem is expressed in terms of solutions to smaller problems
  - Recursive formulation

Data Structures

- Three tables, T1, T2 and T3 of size (m+1)x(n+1)
- [i,j] entry in each table corresponds to optimally aligning A and B
  - T1: ai is matched with bj
  - T2: - matched with bj
  - T3: ai matched with -

Table Meanings

- T1: [i,j] entry – gives the score of:
  - $(a_1,\ldots,a_{i-1}:b_1,\ldots,b_{j-1})_{\text{aligned}}(a_i,b_j)$
- T2: [i,j] entry – gives the score of:
  - $(a_1,\ldots,a_i:b_1,\ldots,b_{j-1})\text{aligned}(-,b_j)$
- T3: [i,j] entry – gives the score of:
  - $(a_1,\ldots,a_{i-1}:b_1,\ldots,b_j)\text{aligned}(a_i,-)$

T1[i,j]

$$T_1[i,j] = f(a_i,b_j) + \max \begin{cases} T_1[i-1,j-1] \\ T_2[i-1,j-1] \\ T_3[i-1,j-1] \end{cases}$$

- (a_i,b_j) and best alignment of $(a_1,\ldots,a_{i-1}:b_1,\ldots,b_{j-1})$
- Best alignment of $(a_1,\ldots,a_{i-1}:b_1,\ldots,b_{j-1})$: Max of
  - $(a_1,\ldots,a_{i-2}:b_1,..,b_{j-2})_{aligned}(a_{i-1},b_{j-1})$
  - $(a_1,\ldots,a_{i-1}:b_1,..,b_{j-2})_{aligned}(-,b_{j-1})$
  - $(a_1,\ldots,a_{i-2}:b_1,\ldots,b_{j-1})_{aligned}(a_{i-1},-)$

T2[i,j]

$$T_2[i,j] = \max \begin{cases} T_1[i,j-1] - (g+h) \\ T_2[i,j-1] - g \\ T_3[i,j-1] - (g+h) \end{cases}$$

- Best of ((-,b_j) and alignment of (a_1,…,a_i:b_1,…,b_{j-1}))

T3[i,j]

$$T_3[i,j] = \max \begin{cases} T_1[i-1,j] - (g+h) \\ T_2[i-1,j] - (g+h) \\ T_3[i-1,j] - g \end{cases}$$

- Best of ((a_i,-) and alignment of (a_1,…,a_{i-1}:b_1,…,b_j))

Table Initialization

- First row and column of each table initialized to –inf except:
  - T1[0,0] = 0;
  - T2[0,j] = h+gj;
  - T3[i,0] = h+gi

Filling Up of Table Entries

- For parallelism, fill diagonal by diagonal (diagonal scan)
- Entries required for computing a diagonal depends only on previous two diagonals
- However, some diagonals are too short for parallelism; hence processors will be load unbalanced and hence idle
- Row scan or column scan will be better for parallelization

Row-by-row scan

- For this, computation of a row i should depend only on (i-1)-th row
- For parallelization, distribution of tables should be along columns
- i-th row of T1 and T3 depends on only the (i-1)th rows
- But, i-th row of T2 depends on values in i-th row

Row-by-row scan

After computing T1 and T3, T2 can be computed as:

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases} \quad (4)$$

Then,

$$T_2[i,j] = \max\begin{cases} w[j], \\ T_2[i,j-1] - g. \end{cases}$$

Let

$$\begin{aligned} x[j] &= T_2[i,j] + jg \\ &= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases} \\ &= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases} \end{aligned}$$

For column distribution of table to processors, x[j] can be computed using prefix operation using MPI_Scan

Parallelization

- 1-d block distribution along columns of Tables T1, T2 and T3; each processor responsible for n/P columns
- For sequences:
- B: bj is needed only in computing column j; hence block distribution of B; ith segment of n/P characters given to processor Pi
- A: ai is needed by all processors at the same time when row is computed; but block distribution followed to reduce space; Pi broadcasts its block to all processors when row i is computed

Parallelization

- T1[i,j] needs:
  - T1[i-1,j-1], T2[i-1,j-1], T3[i-1,j-1]
- w[j] needs:
  - T1[i,j-1], T3[i,j-1]
- A processor needs to get these 5 elements from the preceding processor for computing its left most column

Complexity

- t_l – latency time; t_b – time for a single message
- Computing each row takes
  - $O(n/p + (t_l+t_b)\log p)$ time – time for a prefix operation using binary tree
- Each of p broadcasts for broadcasting portions of A:
  - $O((t_l+t_b m/p)\log p)$
- Total (for m rows for each processor):
  - $O(mn/P + t_l(m+p)\log p + t_b m \log p)$

References

- Parallel Biological Sequence Comparison using Prefix Computations. Aluru et. al. JPDC 2003.