Parallel Implementation for Algorithm for Computing Translocation Distance between Genomes

Kenneth Chin, Seh Leng Lim, Jatin Shah
Yale University
Computer Science Department

22 Dec 2000

Introduction

The aim of our project is to parallelize the algorithm by Sridhar Hannenhalli in his paper titled "Polynomial-time Algorithm for Computing Translocation Distance between Genomes.

What is a translocation?

A translocation is a rearrangement event, which exchanges genetic material between different chromosomes. A translocation on a pair of chromosomes of genome A transforms genome A into another genome. There are two types of translocation: prefix-prefix translocation and prefix-suffix translocation (See figure 1 below).

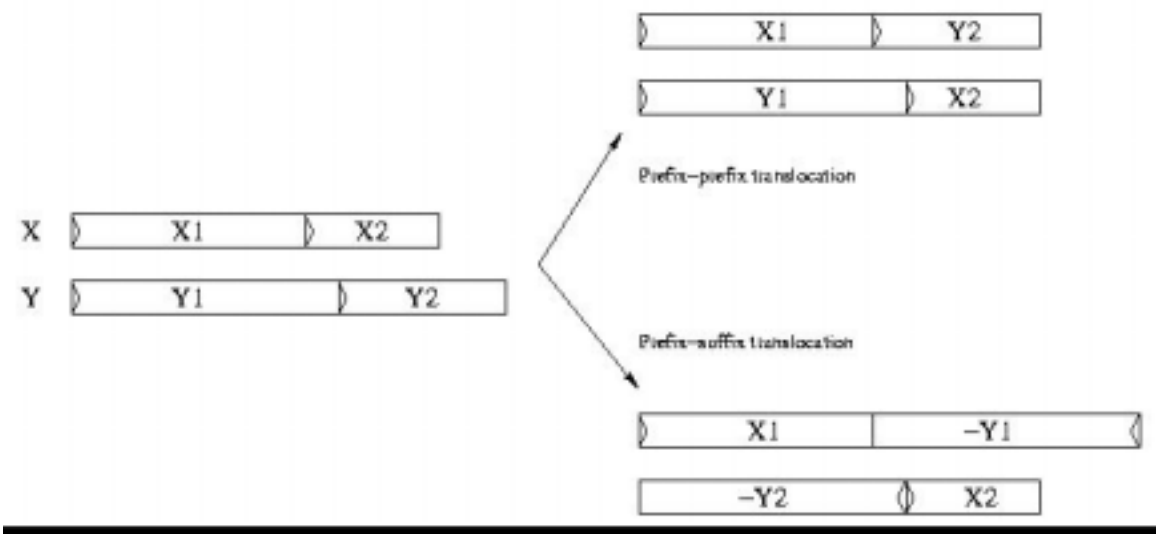

Figure 1

Genome Translocation Algorithm

The cycle graph of the genome is constructed as shown in Figure 2.

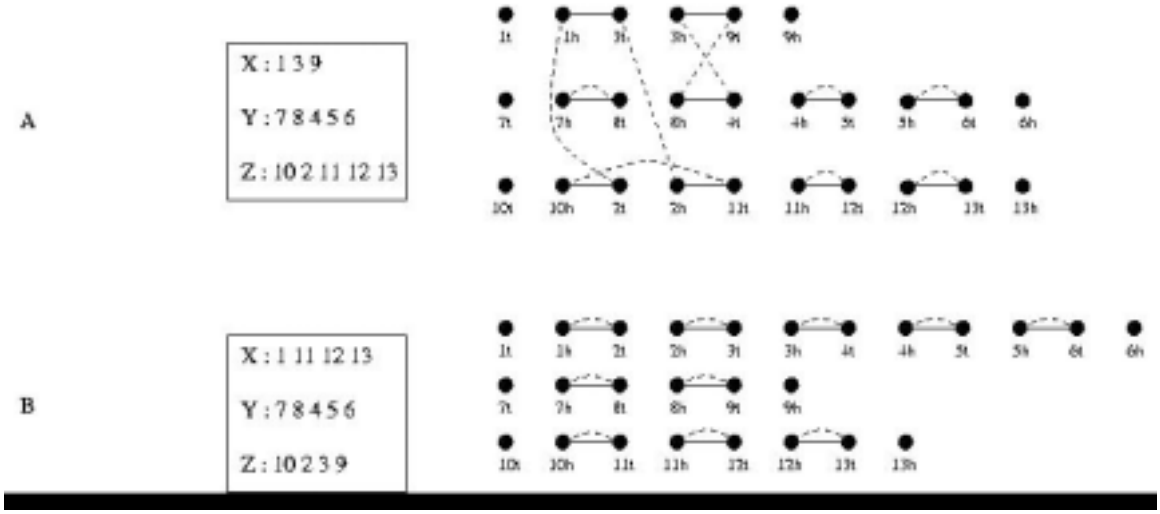

Figure 2

Each gene is split into two nodes, viz. head and tail. The dark edges denote the adjacency of genes in each chromosome and the gray edges denote the adjacency of the genes in the final genome. The degree of each vertex in this graph is two; hence the graph is a set of disjoint cycles. The final genome has all cycles of length two; hence the number of cycles in the final genome is maximum.

The genome translocation algorithm transforms the given genome into the final genome by a series of translocations. There are three kinds of translocations possible, viz. proper translocation, improper translocation and bad translocation as shown in Figure 3. Proper translocations increase the number of cycles; bad translocations decrease the number of cycles and improper translocations do not affect the number of cycles. Ideally, we would like to perform only those translocations, which increase the number of cycles in each stage. However, due to formation of *sub-permutations*, it is not always possible to do so. In such a case, a bad translocation has to be performed. Proper translocations can then be performed on the genome until we reach the final genome or the genome again reaches the stage containing a *sub-permutation*, in which case we perform a bad translocation again.

It can be proved that if the genome has not reached its final stage, there is either a proper translocation, which does not affect the number of sub-permutations or it is possible to perform a bad translocation on the genome.

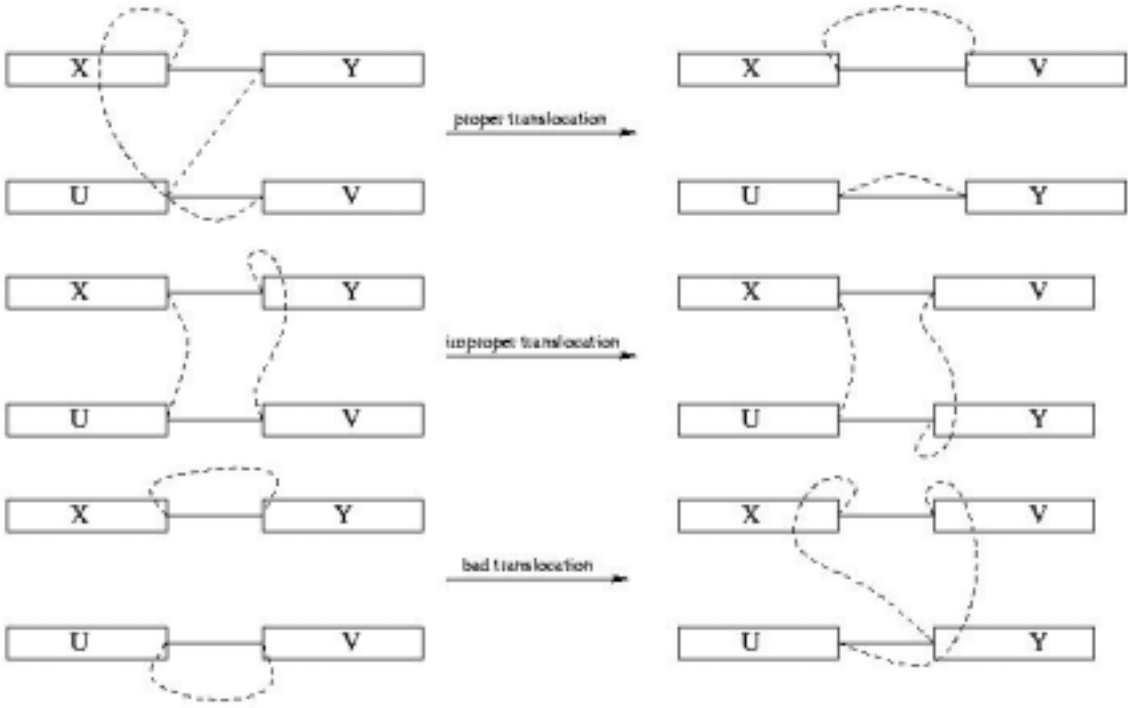

Figure 3

Parallel Programming Implementation

We are implementing the program using shared memory model. The entire genome is stored into the shared memory, which can be simultaneously accessed by parallel threads.

Data Modeling

In our program, each gene is represented by a data structure that contains its name in integer format, its links to the previous gene and the next gene in the chromosome. Similarly, each chromosome is represented by a data structure that contains its name in integer format, its link to its gene sequence, its links to the previous chromosome and next chromosome in the genome. Each genome is represented by its name in integer format, the number of chromosomes it contains, the number of genes it contains as well as its link to its chromosome sequence.

```
typedef struct gene_t {
   int name;
   subgene left;
   subgene right;
   struct gene_t *prev; // black edge;
   struct gene_t *next; // black edge;
} gene;
```

```
typedef struct chromosome_t {
   int name;
   gene *g;
   struct chromosome_t *prev;
   struct chromosome_t *next;
} chromosome;

typedef struct genome_t {
   int name;
   int nchromo;
   int ngene;
   chromosome *c;
} genome;
```

To facilitate graph traversal, each gene further contains a left subgene and right subgene. The data structure for the subgene is as shown below :

```
typedef struct subgene_t {
   int name; // tail:0; head:1;
   int done; // 0: not done; 1: done;
   int chromoid;
   struct gene_t *parent;
   struct subgene_t *gray;  // gray edge;
} subgene;
```

The gray field represents the traversal of a gray link to the subgene representing the left half or the right half of the next gene. Whether the left half or the right half contains a head or tail will be determined by the value of the name field of the subgene.

Program Flow

When the program is started, it spawns off two threads, each to read from one data file representing one genome. Each thread will then build up the cycle graph for each genome, without the gray links. Then the threads are killed, and the main program takes over to draw the gray edges for the first genome, with reference to the second genome.

Next, the main program then goes through the first genome to identify points of cleavages in its chromosomes that lead to proper translocations, and store these points in a task queue.

Thereafter, if the proper translocation task queue is not empty, the main program proceeds to process the proper translocation task queue. Otherwise, it will perform a bad translocation.

Next, the main program counts the number of cycles of length two in the first genome, and checks for the condition as to whether it is equal to the number of chromosomes subtracted from the number of genes. If not, the program will loop back to identify the proper translocations and continue from there until the condition is satisfied.

Parallel Processing of Proper Translocations

When processing the proper translocation task queue, the program spawns a thread to pick up one translocation each from the table. The number of threads spawned is equal to

the number of proper translocations contained in the task queue. As the task queue is implemented as a linked list, a mutex lock is set for the head of the linked list, each time a proper translocation is picked up by a thread. Then the head advances to the next position in the linked list, and the mutex lock for the head is unset.

This means that the threads process the proper translocations (cycles) simultaneously. If the proper translocations are valid, the pair of chromosomes will be swapped at their points of cleavage. This involves a swapping of the next and the previous pointers in the genes at the cleavage. The chromosome id (chromoid) is required to determine whether one gene is on the same chromosome as another, while determining whether the proper translocation is valid. The individual threads do not update the chromoid; hence each thread is not aware of the changes made to the genome by the other threads and the translocations can proceed in parallel without the need for any specialized locking mechanisms.

A thread join is performed to ensure that all the threads are killed at the same time after they process the translocations. By this time, the chromosome ids (chromoids) in the subgenes may be messed up after the simultaneous translocations that are performed. Therefore, the main program takes over to update and synchronize the chromosome ids of all the subgenes in the first genome.

Parallel Processing of Bad Translocations

The parallel processing of bad translocations is achieved in two stages. In the first stage (detect_minsp function), all the minSPs in the genome are added to a task queue. Another queue also keeps track of cycles of length two. In the second stage, each process takes two minSPs, which are residing on different chromosomes from the task queue and destroys the translocation. If it is not possible to choose such a pair (even isolation and odd number of minSPs), then the minSP is paired with a cycle of length two. The processes continue in this fashion until the task queue is empty. Finally, the chromoids of the genes are updated. It is important to note that during the time of translocations, the chromoids do not reflect the actual location of the genes, but the location of the genes before the translocation started. Thus, each process is oblivious to the translocation carried out by other processes. This helps in achieving concurrency without locks on the genes or chromosome. *This implementation of bad translocations differs slightly from that described in the original paper.*

Results

Below are the 3 test cases tested.

Test 1

A : 1 2 3 4 13 14 15 16
    9 10 11 5 12 6 7 8
B : 1 2 3 4 5 6 7 8
    9 10 11 12 13 14 15 16

After running the program, the outcome is as shown below :

```
gene 14 in chromosome 1
Gene(14,15) is done
returning from classify cycle
gene 15 in chromosome 1
Gene(15,16) is done
returning from classify cycle
gene 16 in chromosome 1
in while loop in classify
before g1cycle mutex lockafter g1cycle mutex lockafter pthread join for
genome1
after count1 and count2
Translocation achieved
Genome1 is as follows
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
```

Test 2 (minSP case)

A : 1 2 4 3 5 6
    7 8 9
B : 1 2 3 4 5 6
    7 8 9

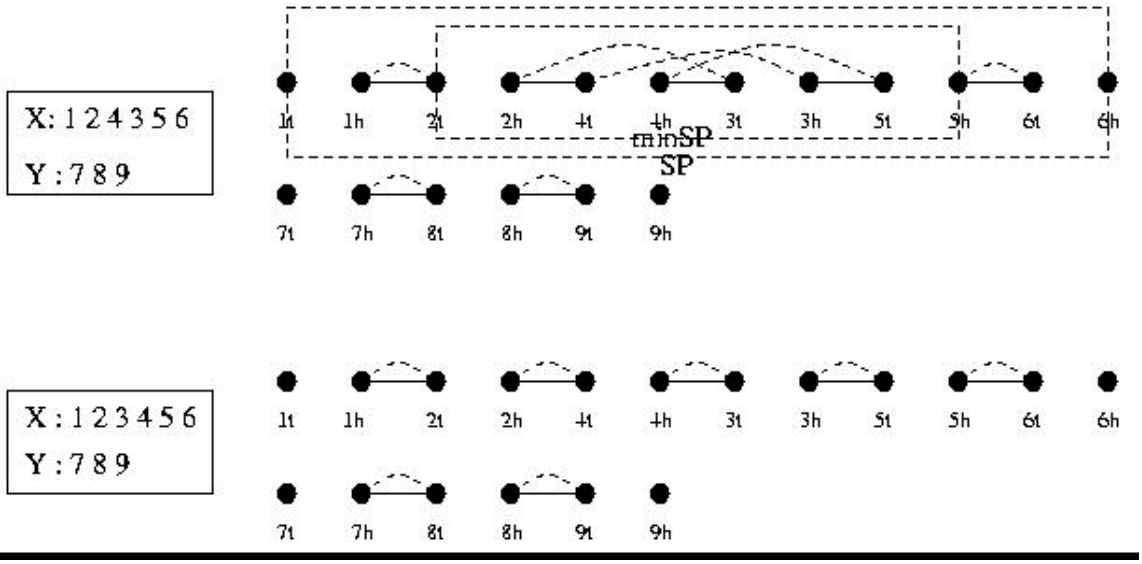

Figure 4

After running the program, the outcome is as shown below :

```
returning from classify cycle
gene 6 in chromosome 0
in while loop in classify
before g1cycle mutex lockafter g1cycle mutex lockgene 7 in chromosome 1
```

```
Gene(7,8) is done
returning from classify cycle
gene 8 in chromosome 1
Gene(8,9) is done
returning from classify cycle
gene 9 in chromosome 1
in while loop in classify
before g1cycle mutex lockafter g1cycle mutex lockafter pthread join for
genome1
after count1 and count2
Translocation achieved
Genome1 is as follows
1 2 3 4 5 6
7 8 9
```

Test 3

A : 1 3 9
7 8 4 5 6
10 2 11 12 13

B : 1 11 12 13
7 8 4 5 6
10 2 3 9

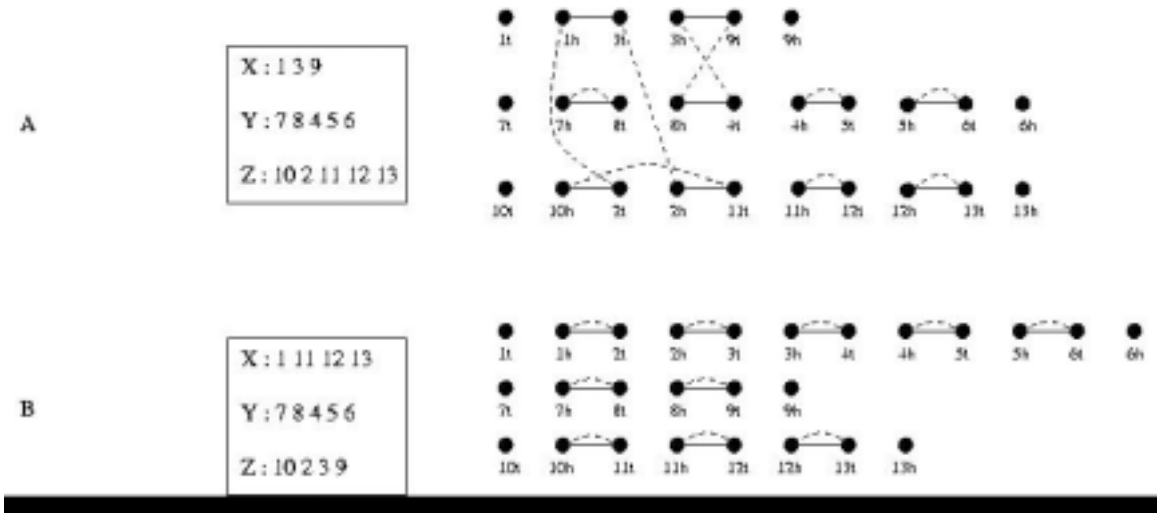

Figure 5

After running the program, the outcome is as shown below:

```
gene 3 in chromosome 2
Gene(3,9) is done
returning from classify cycle
gene 9 in chromosome 2
```

```
in while loop in classify
before g1cycle mutex lockafter g1cycle mutex lockafter pthread join for
genome1
after count1 and count2
Translocation achieved
Genome1 is as follows
1 11 12 13
7 8 4 5 6
10 2 3 9
```

Future Work

We have several alternatives to extend this work in future:

- The parallel version of the algorithm differs from the sequential version in significant manner. A more formal study of the various issues involved in parallelizing the algorithm should be done.
- In our algorithm, bad translocations are performed only when it is not possible to do any proper translocations. The viability of a variant of the algorithm where the task queue contains both bad and proper translocations should be analyzed.
- The algorithm implemented should be tested on genomes of organisms and the performance should be compared with the sequential version of the program