Cameron Anderson
Project Report

Parallel Implementation of the Smith Waterman Algorithm

Problem Description

The Smith Waterman Algorithm was designed as a solution to find the best pair-wise local alignment of DNA sequences. There are many different scoring systems to determine the optimal alignment. The scoring system used for my implementation is: +1 for match, -2 for miss, and -2 for gap. Scores are kept in a two dimensional array with each cell representing a pair of characters from the two sequences. The minimum score is 0, scores are calculated by taking the maximum score of adding the the match or miss score to the score of the cell diagonal to it or the adding the gap penalty to the cell above or left of it.

To trace the sequence, another two dimensional array is created. The traceback array is filled along with the scoring array. The direction taken is determined by which score is chosen for the cell in the scoring array. If a match or miss is chosen, traceback moves diagonally. If there is a gap, traceback moves in the direction of the insertion or deletion. If the score is zero, traceback stops.

Serial Approach

The first step is to initialize each cell in the first row and column to 0.

-		a	g	c	t	c	a
	0	0	0	0	0	0	0
a	0						
g	0						
a	0						
t	0						
t	0						
a	0						
g	0						

Then the second row is calculated by comparing each column to the first character.

-		a	g	c	t	c	a
	0	0	0	0	0	0	0
a	0	1	0	0	0	0	1
g	0						
a	0						
t	0						
t	0						
a	0						
g	0						

Each row is then calculated until all rows are complete.

-		a	g	c	t	c	a
	0	0	0	0	0	0	0
a	0	1	0	0	0	0	1
g	0	0	2	0	0	0	0
a	0	1	0	0	0	0	1
t	0	0	0	0	1	0	0
t	0	0	0	0	1	0	0
a	0	1	0	0	0	0	1
g	0	0	2	0	0	0	0

Parallel Approach

With two threads:

The main thread creates the desired number of threads and passes them their thread id (0 – (n-1)). Each thread is responsible for an equal portion of each column. Thread 0 can loop without restriction because it does not have any rows above it that it depends on for calculations. The other threads must wait for the previous thread to release a mutex variable before they can begin calculation on each column. The mutex variable are stored in a two dimensional array with thread count - 1 rows and sequence length + 1 columns.

-		a	g	c	t	c	a
	0						
a	0						
g	0						
a	0						
t							
t							
a							
g							

After Thread 0 completes its section of the first column, it unlocks the mutex variable Thread 1 needs to begin its section of the first column then continues on to its portion of the second column.

-		a	g	c	t	c	a
	0	0					
a	0	1					
g	0	0					
a	0	1					
t	0						
t	0						
a	0						
g	0						

When Thread 1 completes its section of the first column, it checks to see if Thread 0 has released the mutex variable for its section of the second column. Once it has been released, Thread 1 is free to complete its section of the second column.

-		a	g	c	t	c	a
	0	0	0				
a	0	1	0				
g	0	0	2				
a	0	1	0				
t	0	0					
t	0	0					
a	0	1					
g	0	0					

The threads continue on in this manner until the array is complete.

-		a	g	c	t	c	a
	0	0	0	0	0	0	0
a	0	1	0	0	0	0	1
g	0	0	2	0	0	0	0
a	0	1	0	0	0	0	1
t	0	0	0	0	1	0	0
t	0	0	0	0	1	0	0
a	0	1	0	0	0	0	1
g	0	0	2	0	0	0	0

Output from program on above sequences:

```
Serial:
Score:
0 0 0 0 0 0 0
0 1 0 0 0 0 1
0 0 2 0 0 0 0
0 1 0 0 0 0 1
0 0 0 0 1 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 1
0 0 2 0 0 0 0

Traceback:
0 0 0 0 0 0 0
0 1 0 0 0 0 1
0 0 1 0 0 0 0
0 1 0 0 0 0 1
0 0 0 0 1 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 1
0 0 1 0 0 0 0

r1: ag
r2: ag
In: 2.71797e-05 seconds

Parallel:
Score:
0 0 0 0 0 0 0
0 1 0 0 0 0 1
0 0 2 0 0 0 0
0 1 0 0 0 0 1
0 0 0 0 1 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 1
0 0 2 0 0 0 0

Traceback:
0 0 0 0 0 0 0
0 1 0 0 0 0 1
0 0 1 0 0 0 0
0 1 0 0 0 0 1
0 0 0 0 1 0 0
0 0 0 0 1 0 0
0 1 0 0 0 0 1
0 0 1 0 0 0 0

r1: ag
r2: ag
In: 0.000216007 seconds
```

Performance

The parallel implementation takes significantly longer than the serial implementation on small sequences due to all the extra overhead required. For larger sequences, computation time for the two start to get closer together. Performance analysis will be done on sequence 2 from the sequence file (~5,800 characters) and 500 and 5000 lines of the chr1 file (~25000 – 250000 characters). The command line arguments are seq, lines, and thread count, respectively. Scoring and traceback arrays will be left out due to size.

```
./proj 2 500 2

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.63834 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 21.232 seconds

./proj 2 500 4

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.5665 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 15.5988 seconds

./proj 2 500 8

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.49035 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 12.6735 seconds

./proj 2 500 16

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.47714 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
```

```
In: 11.3957 seconds

./proj 2 500 32

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.45636 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 7.09072 seconds

./proj 2 500 64

Serial:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 9.54083 seconds

Parallel:
r1: AGCCC-AG-CCCTTGCC-CACA-GGGCCACACT-CCACG-TGCAGAGCAGCCTCAGC
r2: AGCCCGAGCCCCTTGCCTCACACCGGCC-C-CTCCCACGCTG-AGAG-AG-GTCAGC
In: 5.43262 seconds
```

The parallel implementation's performance does not pass the serial's until 32 threads are called for. Testing for 5000 rows will be done with 32 and 64 threads.

```
./proj 2 5000 32

Serial:
r1: CAGCTCCC-GCCTGCGTTTGTAGACCCCAAAGTTTCTGCAACCAAGCTCTTCA-GACCCACATCCCT-
TCTCCCAGT-
GACTGAACAGTCCCAGCTCCGGCTGGAGAAGGGTGTCTGCAGACCCCGCTGTTGCCTCCCAGGGGAGT
CTCCAGGCCCAGCTCTCG-
CCCCACCGCGACCTCCCAGGCCCAAGTCCCTGCCTACCTCCCAGCAGCCCGAGTGCGATCCTGTTCCTC
CCTCACGGTGGCCTGTTGA
r2: CAGCTCCCAACC-GCCTTTGTAGGCCCCGAACTTTCTCCAGCCAAGCTC-TGAGGGCCCACCT-
CCTGCCTCCTGGTGGCCTGTACAGTTCTAGCACTGGTTGGAGAACAGCCTCTGCAGGCCCCGCCCTTGC
CTCCCAGGGGCCTCTCCAGGCCCAGCTCTTGCCCCCACGGCGGCCTCCCGGGGCCAAGTCCCTGCCTG
CCTCCCAGCAGCCCGCGTGCGGCCCAGCTCCTCCCTCACGGTGGCCTGTTGA
In: 97.7332 seconds

Parallel:
r1:
TCAACAGGCCACCGTGAGGGAGGAGCTGGGCCGCACGCGGGCTGCTGGGAGGCAGGCAGGGACTTGG
CCCCGGGAGGCCGCCGTGGGGGCAAGAGCTGGGCCTGGAGAGGCCCCTGGGAGGCAAGGGCGGGGC
CTGCAGAGGCTGTTCTCCAACCAGTGCTAGAACTGTACAGGCCACCAGGAGGCA-GGAGGTGGGCCCT-
CAGAGCTTGGCTGGAGAAAGTTCGGGGCCTACAAAGGC-GGTTGGGAGCTGGGC
r2:
TCAACAGGCCACCGTGAGGGAGGAACAGGATCGCACTCGGGCTGCTGGGAGGTAGGCAGGGACTTGG
GCCTGGGAGGTCGCGGT-
GGGGCGAGAGCTGGGCCTGGAGACTCCCCTGGGAGGCAACAGCGGGGTCTGCAGACACCCTTCTCCA
GCCGGAGCTGGGACTGTTCA-GTCACTGGGA-GAAGGGATGTGGG-
TCTGAAGAGCTTGGTTGCAGAAACTTTGGGGTCTACAAACGCAGG-CGGGAGCTGAGC
In: 180.265 seconds
```

```
./proj 2 5000 64

Serial:
r1: CAGCTCCC-GCCTGCGTTTGTAGACCCCAAAGTTTCTGCAACCAAGCTCTTCA-GACCCACATCCCT-
TCTCCCAGT-
GACTGAACAGTCCCAGCTCCGGCTGGAGAAGGGTGTCTGCAGACCCCGCTGTTGCCTCCCAGGGGAGT
CTCCAGGCCCAGCTCTCG-
CCCCACCGCGACCTCCCAGGCCCAAGTCCCTGCCTACCTCCCAGCAGCCCGAGTGCGATCCTGTTCCTC
CCTCACGGTGGCCTGTTGA
r2: CAGCTCCCAACC-GCCTTTGTAGGCCCCGAACTTTCTCCAGCCAAGCTC-TGAGGGCCCACCT-
CCTGCCTCCTGGTGGCCTGTACAGTTCTAGCACTGGTTGGAGAACAGCCTCTGCAGGCCCCGCCCTTGC
CTCCCAGGGGCCTCTCCAGGCCCAGCTCTTGCCCCCACGGCGGCCTCCCGGGGCCAAGTCCCTGCCTG
CCTCCCAGCAGCCCGCGTGCGGCCCAGCTCCTCCCTCACGGTGGCCTGTTGA
In: 102.693 seconds

Parallel:
r1:
TCAACAGGCCACCGTGAGGGAGGAGCTGGGCCGCACGCGGGCTGCTGGGAGGCAGGCAGGGACTTGG
CCCCGGGAGGCCGCCGTGGGGGCAAGAGCTGGGCCTGGAGAGGCCCCTGGGAGGCAAGGGCGGGGC
CTGCAGAGGCTGTTCTCCAACCAGTGCTAGAACTGTACAGGCCACCAGGAGGCA-GGAGGTGGGCCCT-
CAGAGCTTGGCTGGAGAAAGTTCGGGGCCTACAAAGGC-GGTTGGGAGCTGGGC
r2:
TCAACAGGCCACCGTGAGGGAGGAACAGGATCGCACTCGGGCTGCTGGGAGGTAGGCAGGGACTTGG
GCCTGGGAGGTCGCGGT-
GGGGCGAGAGCTGGGCCTGGAGACTCCCCTGGGAGGCAACAGCGGGGTCTGCAGACACCCTTCTCCA
GCCGGAGCTGGGACTGTTCA-GTCACTGGGA-GAAGGGATGTGGG-
TCTGAAGAGCTTGGTTGCAGAAACTTTGGGGTCTACAAACGCAGG-CGGGAGCTGAGC
In: 140.32 seconds
```

Performance for the parallel implementation slows down past the serial implementation again when the problem size is increased. It improves with more threads but still doesn't reach the serial algorithm's performance.