Global Sequence Alignments
using C / MPI

CSE 633 - Fall 2012

State University of New York at Buffalo
Dr. Russ Miller

Ravi Patel (www.RaviPatel.me)
presented on November 29, 2012

Outline

- Global Sequence Alignment?
  - Applications
- Sequential Algorithm
  - Needleman-Wunsch Algorithm
- Parallel Algorithm
- Experiments and Results
- Future Extensions

The Problem

- Sequence Alignment
  - Think of sequences as strings of letters from a fixed alphabet
  - The goal is to describe sequence similarity, or how closely two sequences match each other
    - Can be a score (number)
    - Can be an “alignment” (visual representation)
  - Global (align sequences from end-to-end)
  - Local (align similar regions between sequences)

An Example

- Input: two DNA sequences
  - X: `GCGCATGGATTGAGCGA`
  - Y: `TGCGCCATTGATGACCA`
- Insert gaps (minimize) to align and match letters (maximize)
- Possible Alignments:
  - `-GCGC-ATGGATTGAGCGA`   4, 13, 2
  - `TGCGCCATTGAT-GACC-A`
  - `------GCGCATGGATTGAGCGA` 12, 5, 6
  - `TGCGCC----ATTGATGACCA--`

Applications

- Bioinformatics involves a lot of sequences
    - DNA, RNA, and Protein
    - Global Sequence Alignment used to understand evolutionary relationships
        - e.g., human DNA vs. chimps DNA
- Natural language processing
- Business and marketing research
    - e.g., analyze series of purchases over time

Global Sequence Alignment

- Scoring Function
  - s(x, y) ⟶ match (+2), mismatch (-1), gap (-2)
  - `-GCGC-ATGGATTGAGCGA`
  - `TGCGCCATTGAT-GACC-A`
  - 4(-2) + 13(+2) + 2(-1) = 16
  - `------GCGCATGGATTGAGCGA`
  - `TGCGCC----ATTGATGACCA--`
  - 12(-2) + 5(+2) + 6(-1) = -20

Global Sequence Alignment

- Needleman-Wunsch Algorithm
  - Based on dynamic programming
    - Build up an optimal alignment using previous solutions for optimal alignments of smaller substrings.
  - Guarantees an optimal global alignment of two sequences

Needleman-Wunsch Algorithm

- Given 2 sequences, X and Y, of lengths, n and m, respectively

$$T : \{0, 1, \ldots , n\} \times \{0, 1, \ldots , m\} \rightarrow R$$

- T(i, j) equals the best score of the alignment of the two prefixes $(x_1, x_2, \ldots , x_i)$ and $(y_1, y_2, \ldots , y_j)$.

Needleman-Wunsch Algorithm

	-	**x_1**	**...**	**x_i**	**...**
-	**T(0, 0)**				
y_1					
...			T(i-1, j-1)	T(i, j-1)	
y_j			T(i-1, j)	**T(i, j)**	
...					

- T(i, j) equals the best score of the alignment of the two prefixes
  - $(x_1, x_2, \ldots, x_i)$ and $(y_1, y_2, \ldots, y_j)$.

Needleman-Wunsch Algorithm

- Optimal alignment between X and Y can end with one of three possibilities:
    - y_j is aligned with a gap
    - x_i is aligned with y_j
    - x_i is aligned with a gap

- T(i, j) = max:
    - $T(i, j-1) + s('-', y_j)$ $\longrightarrow T(i, j-1) - 2$
    - $T(i-1, j-1) + s(x_i, y_j)$,
    - $T(i-1, j) + s(x_i, '-')$, $\longrightarrow T(i-1, j) - 2$

Sequential Example

- Initial Setup

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- Values predefined by scoring function
    - $s(x_i, \text{'-'}) = s(\text{'-'}, y_j) = -2$

Sequential Example

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2			
G	-4				
G	-6				

- T(1, 1) = max:
  - T(1, 0) - 2,
  - T(0, 0) + s('A', 'A'),
  - T(0, 1) - 2
- T(1, 1) = max(-4, 2, -4) = 2

Sequential Example

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4				
G	-6				

- Each row computed in O(n)

Sequential Example

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4	0			
G	-6				

- T(1, 2) = max:
  - T(1, 1) - 2,
  - T(0, 1) + s('G', 'A'),
  - T(0, 2) - 2
- T(3, 1) = max(-8, -3, -2) = -3

Sequential Example

- $O(mn) = O(n^2)$ to compute the table

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4	0	1	2	0
G	-6	-2	-1	3	1

- Optimal Alignment Score = T(4, 3) = 1

Sequential Example

- O(m+n) = O(n) to construct the alignment

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4	0	1	2	0
G	-6	-2	-1	3	1

- Optimal Alignments:
  - `ACGT` & `ACGT`
  - `AGG-` & `A-GG`

Sequential Example

- O(m+n) = O(n) to construct the alignment

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4	0	1	2	0
G	-6	-2	-1	3	1

- Optimal Alignments:
  - `ACGT` & `ACGT`
  - `AGG-` & `A-GG`

Sequential Algorithm Summary

- $O(1)$ to compute a score
- $O(mn) = O(n^2)$ to compute the table
- $O(m+n) = O(n)$ to construct the alignment

- Memory = $O(n^2)$
- Runtime = $O(n^2)$

Parallel Algorithm

- Initial values are predefined

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- Divide processors, p, by columns
    - each processor, gets O(n/p) columns and O(m) rows
    - compute row-by-row

Parallel Algorithm

- Step 1a: T(i, j-1)

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- Each processor has T(i, j-1) from previous row

Parallel Algorithm

- Step 1b: T(i-1, j-1)

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- After each row, each processor will send its T(i, j) to the proceeding processor
  - T(i-1, j-1) will be available to each processor
  - can be done in O(1)

Parallel Algorithm

- Step 2: T(i-1, j)

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- Each processor has
  - max { T(i-1, j-1) + s(x_i, y_j), T(i, j-1) - 2 }

Parallel Algorithm

- Step 2: to get T(i-1, j)
  - Let w[i] = max { T(i-1, j-1) + s(x_i, y_j), T(i, j-1) - 2 }
  - Let x[i] = T(i, j) - s('-', y_k) for k = 1 $\longrightarrow$ i
  - *... some proofs ...*
  - x[i] = max((w[i] + gi), (x[i-1]))
  - = max((w[i] + gi), max((w[i-1] + g(i-1)), x[i-2]))
  - *... some more proofs ...*
- T(i, j) = x[i] – gi
  - use parallel prefix with max operator

Parallel Algorithm

- Step 2: Parallel Prefix with MAX Operator

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2				
G	-4				
G	-6				

- $T(i, j) = x[i] - gi$
  - $x[i] = \max((w[i] + gi), (x[i-1]))$
    - parallel prefix in $O(\log(p))$

Parallel Algorithm

- Step 3: Compute the T(i, j)'s

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4				
G	-6				

- Step 4: Pass T(i, j) to next processor
  - O(1) to send/recv

Parallel Algorithm

- Repeat the 4 Steps for each row

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4				
G	-6				

Parallel Algorithm

- Last processor has optimal alignment score

	-	A	C	G	T
-	0	-2	-4	-6	-8
A	-2	2	0	-2	-4
G	-4	0	1	2	0
G	-6	-2	-1	3	1

- O(n) to construct the alignment

Algorithms Summary

- Runtimes
  - Sequential: $O(n^2)$
  - Parallel: $O(n(\log(p) + n/p)) = O(n\log(p))$ or $O(n^2/p)$
    - worst-case scenario $n >> p$
- Memory
  - Sequential: $O(n^2)$
  - Parallel: $O(n^2/p)$ per processor
    - can have a master node broadcast chunks of data

Experiment 1 - Setup

- Code the Parallel Algorithm using C / MPI
- Run the program with fixed $|X| = |Y|$
    - use 1, 2, 4, 8, 16, 32, 64 -cores
    - measure speedups
- Run the program with fixed number of cores
    - $|X| = |Y| = 1, 2, 4, 8, ..., 1024, 2048, 4096, 8192, ...$
    - measure effects of varying sequence lengths on runtimes
        - determine ideal number of columns per core
- Each test result will be an average of 30 runs

Experiment 1 - Setup

- DELL (2 cores per processor)
  - Number of nodes = 256
  - Primary SC1425 2-Way Compute Nodes
  - Processor Description:
    - 2x3.0GHz (256 nodes) Intel Xeon "Irwindale" Processors
    - Main memory size: 2048 Mbytes (160 nodes)
    - Instruction cache size: 16 Kbytes
    - Data cache size: 16 Kbytes

Experiment 1 - Initial Results

- Runtimes, in seconds

	Cores						
|X| = |Y|	1	2	4	8	16	32	64
1	0.00023						
2	0.00028	0.00052					
4	0.00037	0.00056	0.00075				
8	0.00052	0.0009	0.00048	0.00133			
16	0.00008	0.00144	0.00202	0.00238	0.00249		
32	0.00019	0.00036	0.00112	0.00153	0.00144	0.00543	
64	0.00025	0.00062	0.00188	0.00272	0.00257	0.01099	0.0482
128	0.00118	0.00119	0.00377	0.00529	0.00553	0.02076	0.07447
256	0.00409	0.00295	0.00771	0.0106	0.0122	0.04517	0.11682
512	0.02004	0.01034	0.01733	0.02176	0.0248	0.09112	0.21341
1024	0.25436	0.10694	0.04927	0.04917	0.05473	0.18373	0.41552
2048		0.60298	0.29186	0.1318	0.1241	0.38933	0.89107
4096				0.72205	0.43027	0.88532	2.00383
8192						2.80122	4.90714

Experiment 1 - Initial Results

- Issue: Can only retain ~1,048,576 cells
  - Fix: retain only the last computed row
    - allows for |X| = 1,048,576 and |Y| = infinite?
    - cannot construct the alignment

	-	A	C	G	T
-	x	x	x	x	x
A	x	x	x	x	x
G	-4	0	1	2	0
G	-6	-2	-1	3	1

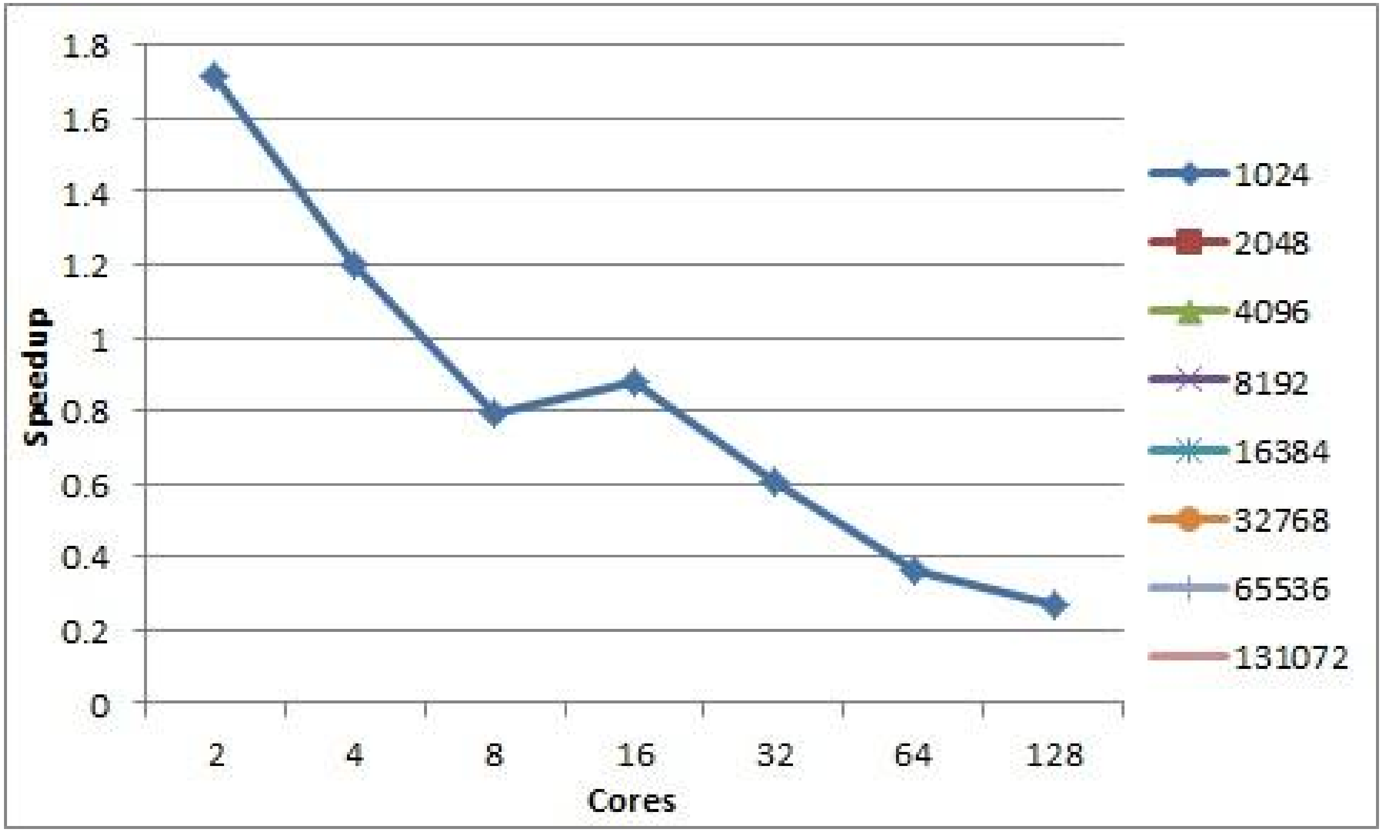
Experiment 1 - Final Results

- Runtimes, in seconds
  - 2-core Speedup: |X| = 256
  - 64-core Speedup: |X| = 4096
  - Optimal Speedup: ~512-1024 columns/core

	Cores							
|X| = |Y|	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1024	0.047000	0.027443	0.039197	0.059140	0.053290	0.077310	0.128710	0.172160
2048	0.188287	0.101977	0.100877	0.104797	0.112717	0.164320	0.254253	0.329250
4096	0.748267	0.393693	0.295803	0.256267	0.258613	0.343583	0.619823	0.627917
8192	2.981063	1.541523	0.965177	0.703053	0.604823	0.731540	0.946080	1.228043
16384	11.850543	6.093013	3.429087	2.149447	1.586450	1.669060	2.464673	5.985753
32768	46.293303	23.371277	12.572553	7.231727	4.646417	4.103897	4.370867	5.200650
65536	184.706410	93.350403	48.527290	26.224470	15.275367	11.326050	10.342973	11.326317
131072	743.696027	374.205590	190.516247	99.517057	54.744023	34.594393	26.799047	26.255567

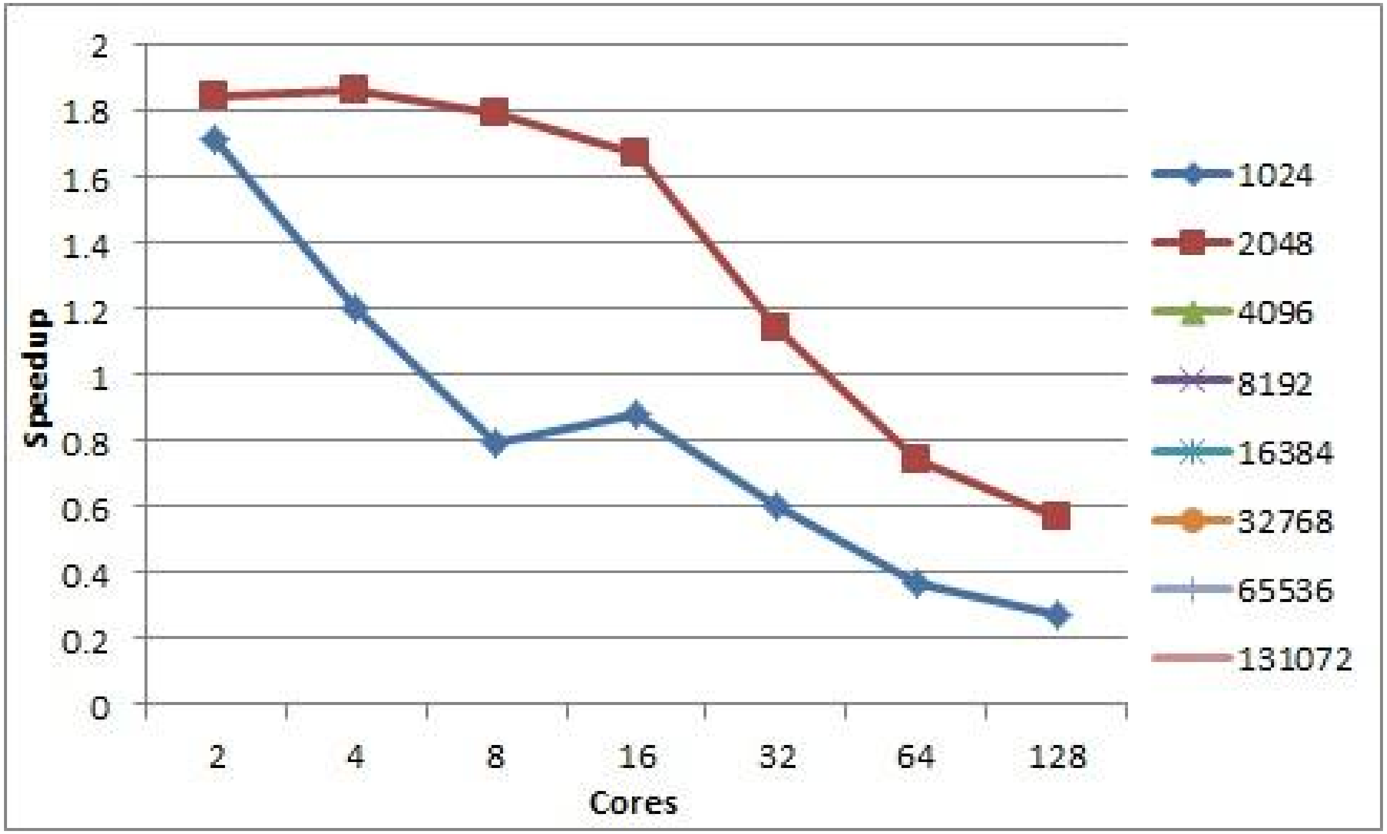
Experiment 1 - Final Results

- Speedups with |X| = 1024
  - 1.71, 1.19, 0.79, 0.88, 0.60, 0.36, 0.27



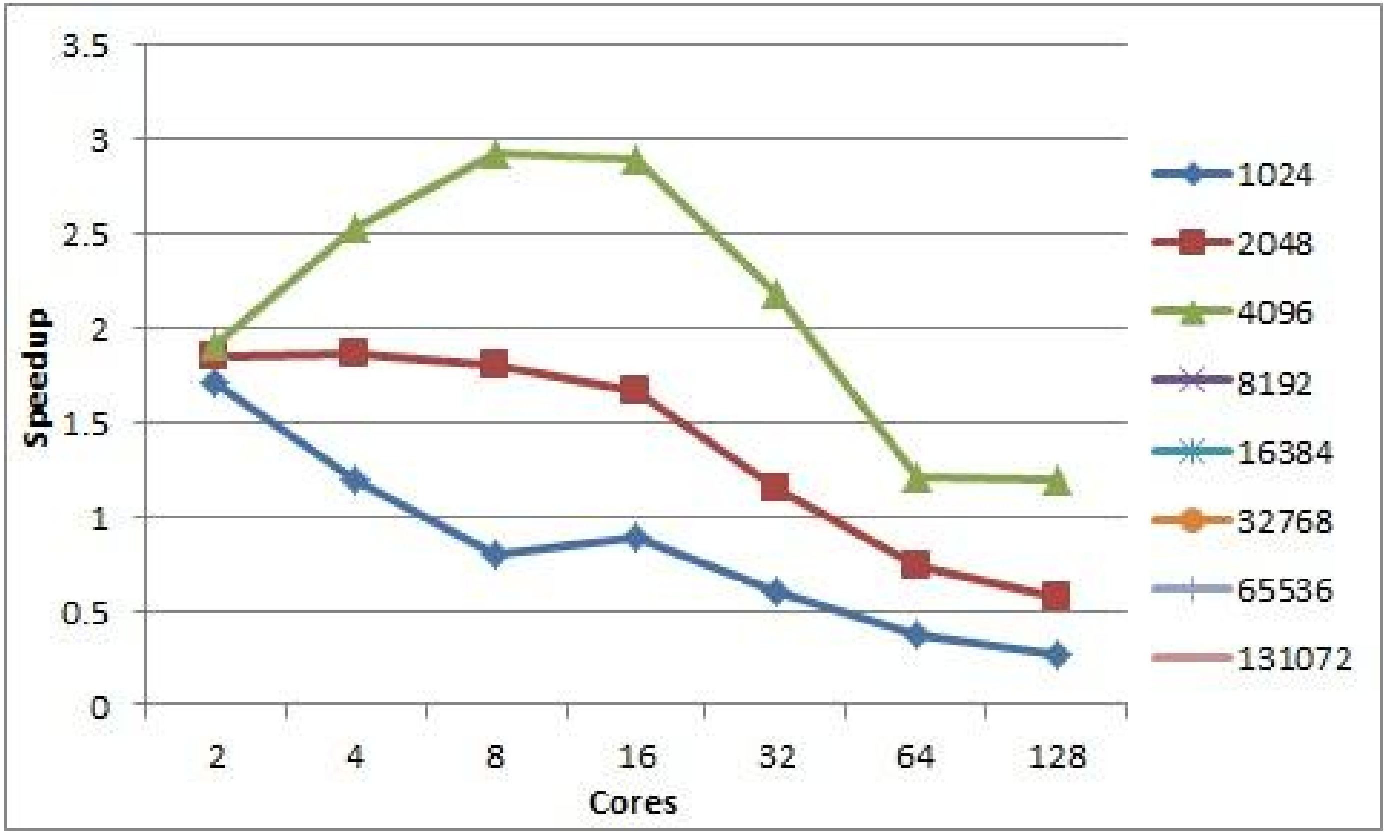
Experiment 1 - Final Results

- Speedups with |X| = 2048
  - 1.84, 1.86, 1.79, 1.67, 1.14, 0.74, 0.57



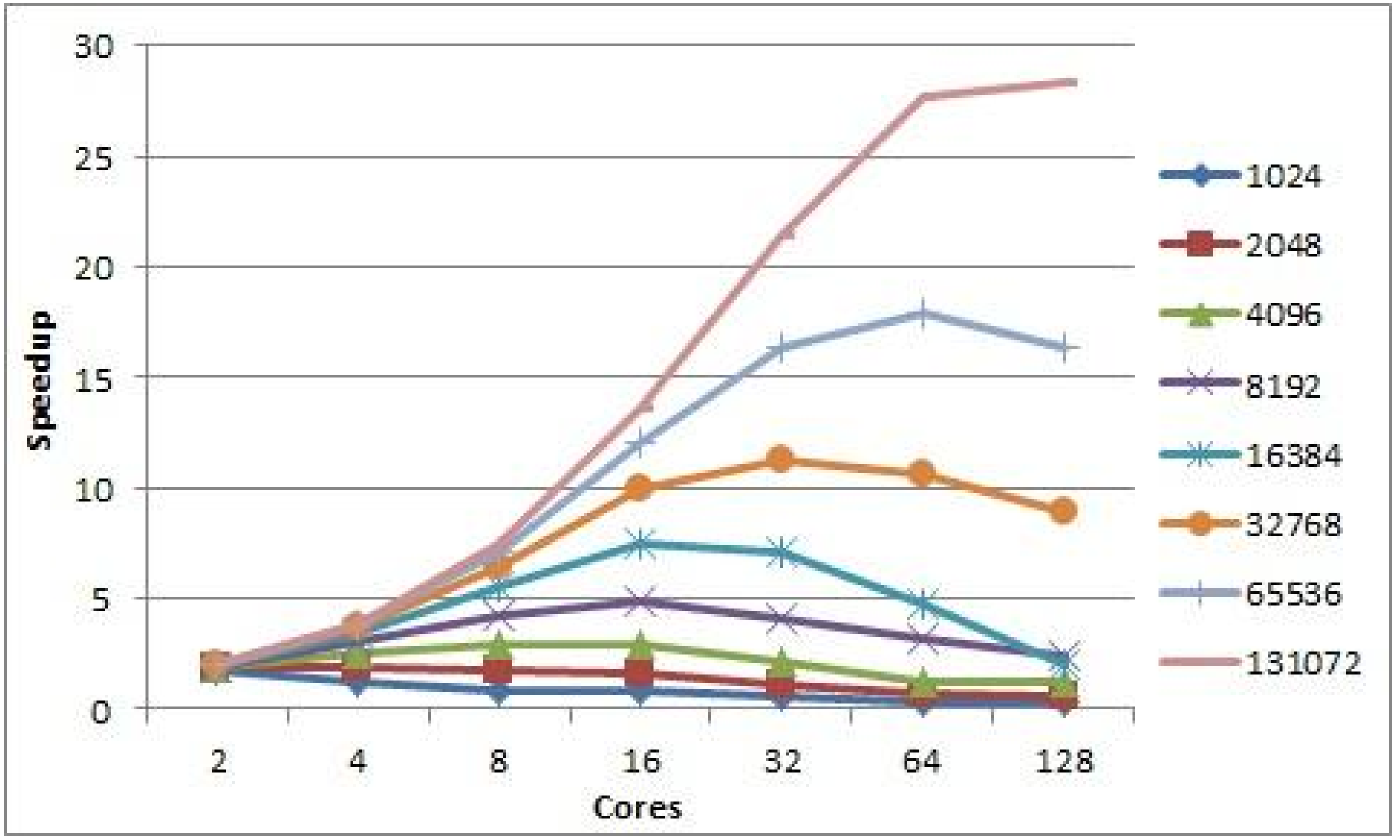
Experiment 1 - Final Results

- Speedups with |X| = 4096
  - 1.90, 2.52, 2.91, 2.89, 2.17, 1.20, 1.19



Experiment 1 - Final Results

- Speedups with |X| = 131072
  - 1.98, 3.90, 7.47, 13.58, 21.49, 27.75, 28.32



Experiment 1 - Findings

- Cannot keep T(i, j) table in memory for larger sequence lengths
  - haploid human genome has about 3 billion base pairs
- Minimum of |X| = 256 to see any speedup
- Speedups peak at |X| = ~512-1024 columns per core

Experiment 2 - Setup

- Divide the program into steps and observe runtimes with increasing cores
  - Parallel Runtime: $O(n(\log(p) + n/p))$

1. Calculate w[i]'s and x[i]'s (sequential prefix)
2. Calculate last x[i]'s (parallel prefix)
3. Calculate scores T(i, j)'s
4. Send last score to next processor

- Each test result will be an average of 30 runs

Experiment 2 - Setup

- IBM (8 cores per processor)
  - Number of nodes = 128
  - PowerEdge C6100 - dual quad-core Compute Nodes
  - Processor Description:
    - 8x2.27GHz Intel Xeon L5520 "Westmere" (Nehalem-EP) Processor Cores
    - Main memory size: 24576 Mbytes
    - Instruction cache size: 128 Kbytes
    - Data cache size: 128 Kbytes
  - InfiniBand Mellanox Technologies MT26428 Network Card

Experiment 2 - Results

- 2-core Speedup: |X| = 128
- 64-core Speedup: |X| = 2048
- Optimal Speedup: ~32-128 columns/core

	Cores						
|X| = |Y|	**1**	**2**	**4**	**8**	**16**	**32**	**64**
64	0.00026	0.00027	0.00027	0.00029	0.00072		
128	0.00077	0.00073	0.0007	0.00084	0.00125	0.00192	
256	0.00312	0.00227	0.00165	0.00143	0.00283		
512	0.01144	0.00767	0.00493	0.00367	0.00534		
1024	0.04663	0.02796	0.01625	0.010390	0.012590	0.016170	0.045120
2048	0.18322	0.10814	0.05815	0.033860	0.030380	0.032360	0.097950
4096	0.72727	0.417590	0.220350	0.122670	0.087310	0.078800	0.078520
8192	3.16608	1.649350	0.849620	0.444460	0.283710	0.202630	0.395770
16384	11.53747	7.291560	4.579580	1.705270	0.965050	0.659020	0.765790
32768	60.5801	38.705300	28.192130	6.745440	3.691470	2.060120	1.541910
65536	187.02227	105.267360	53.168120	26.732400	13.867650	7.493900	5.925580
131072	748.73332	422.938950	306.787620	106.582590	54.259310	28.307290	19.027350

Experiment 2 - Results

- Sequential runtime increased w/ slower cores
- Parallel runtime decreased w/ faster network

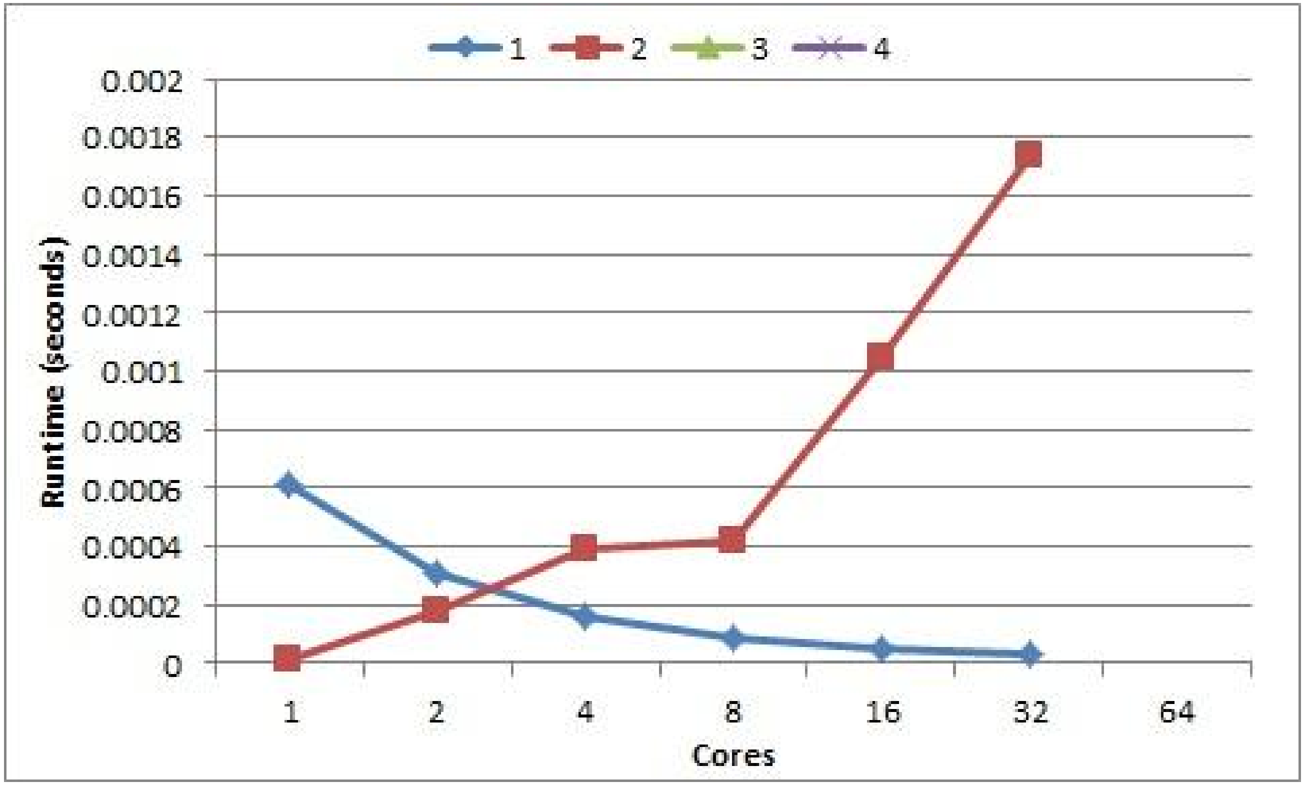
	Experiment 1	Experiment 2
Processor / Network	3.0 GHz / GM	2.27 GHz / IB2
2-Core Speedup	|X| = 256	|X| = 128
64-Core Speedup	|X| = 4096	|X| = 2048
Optimal Speedup	|X| = ~512-1024	|X| = ~32-128

  - parallel computations become less expensive, relative to the sequential computations
    - $O(n(\log(p) + n/p))$

Experiment 2 - Results

1. Calculate w[i]'s and x[i]'s (sequential prefix)
2. Calculate last x[i]'s (parallel prefix)

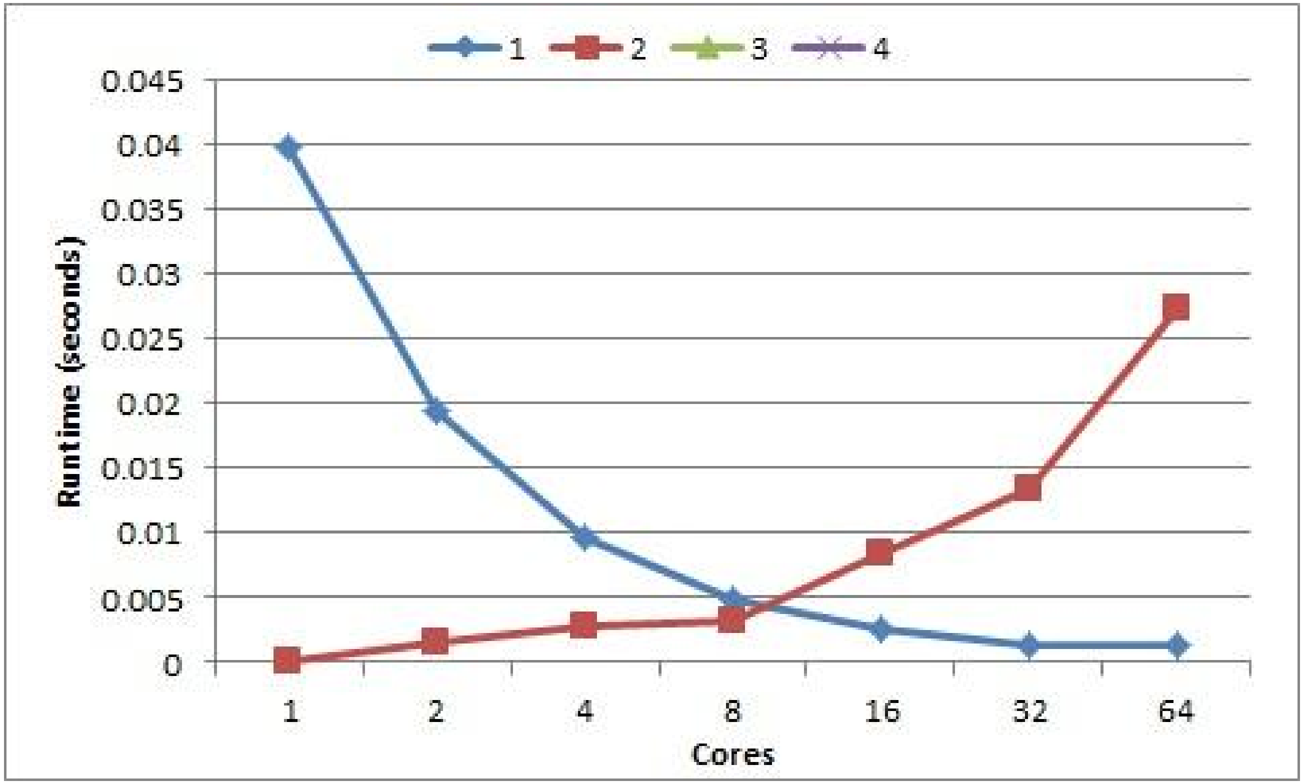
- |X| = 128 , Best Runtime: 2-cores



Experiment 2 - Results

1. Calculate w[i]'s and x[i]'s (sequential prefix)
2. Calculate last x[i]'s (parallel prefix)

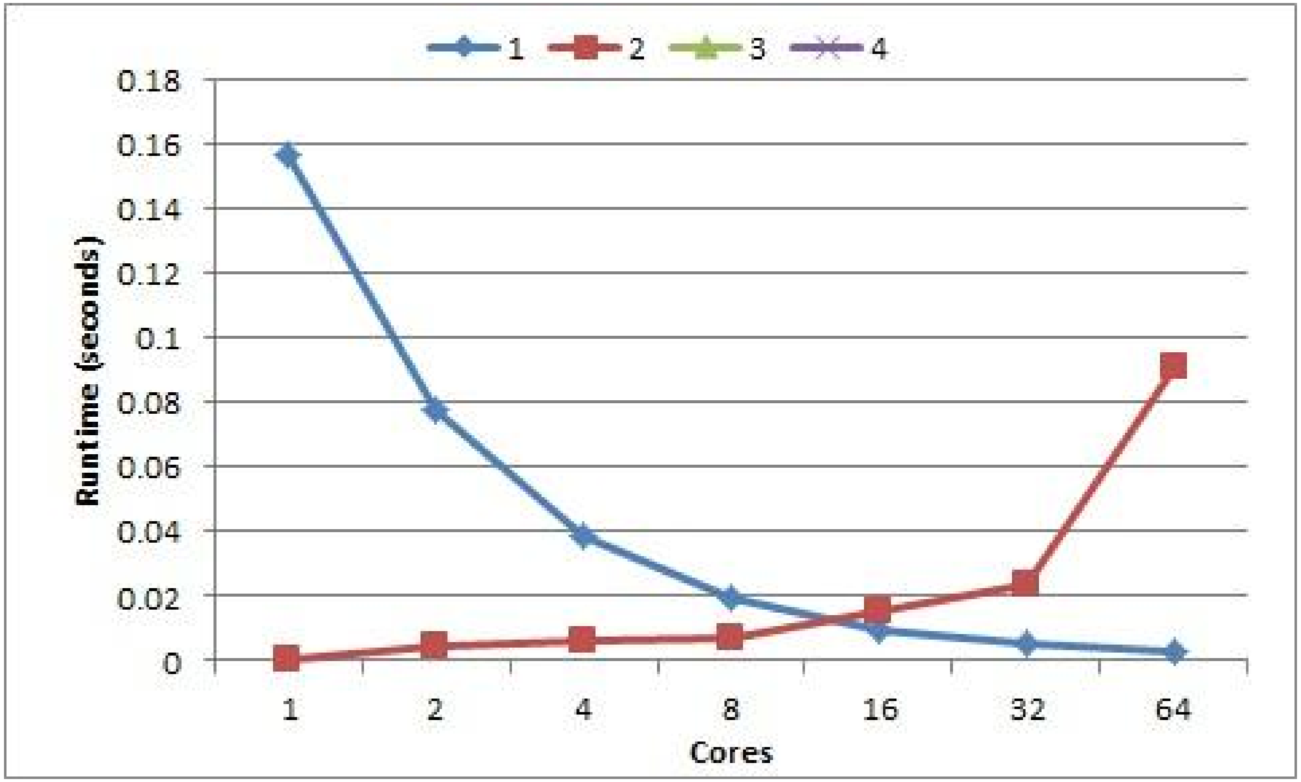
- |X| = 1024, Best Runtime: 8-cores



Experiment 2 - Results

1. Calculate w[i]'s and x[i]'s (sequential prefix)
2. Calculate last x[i]'s (parallel prefix)

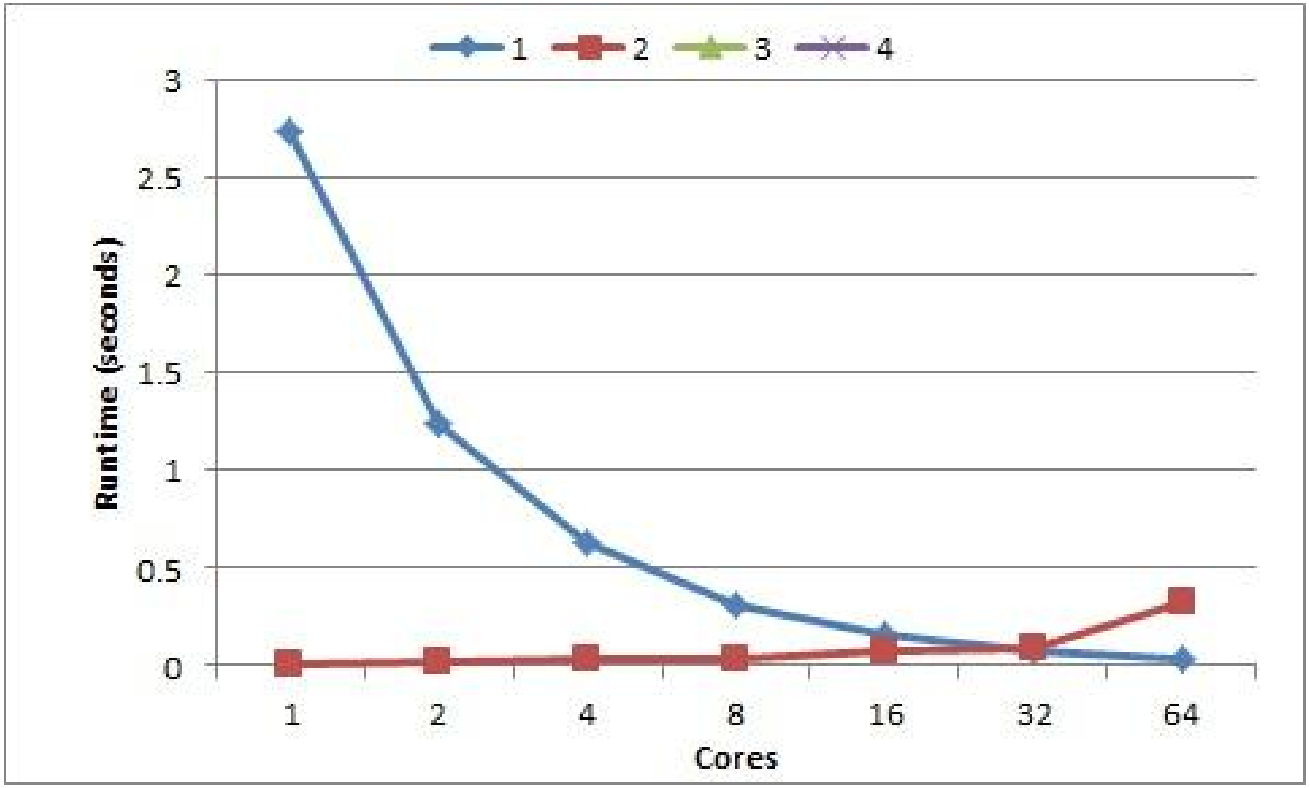
- |X| = 2048, Best Runtime: 16-cores



Experiment 2 - Results

1. Calculate w[i]'s and x[i]'s (sequential prefix)
2. Calculate last x[i]'s (parallel prefix)

- |X| = 8192, Best Runtime: 32-cores



Experiment 2 - Findings

- Optimal Speedups acquired by balancing the equation: $O(n(\log(p) + n/p))$
  - number of cores
  - cores' computational power (GHz)
  - network speed
  - columns per core

Experiment 3 - Setup

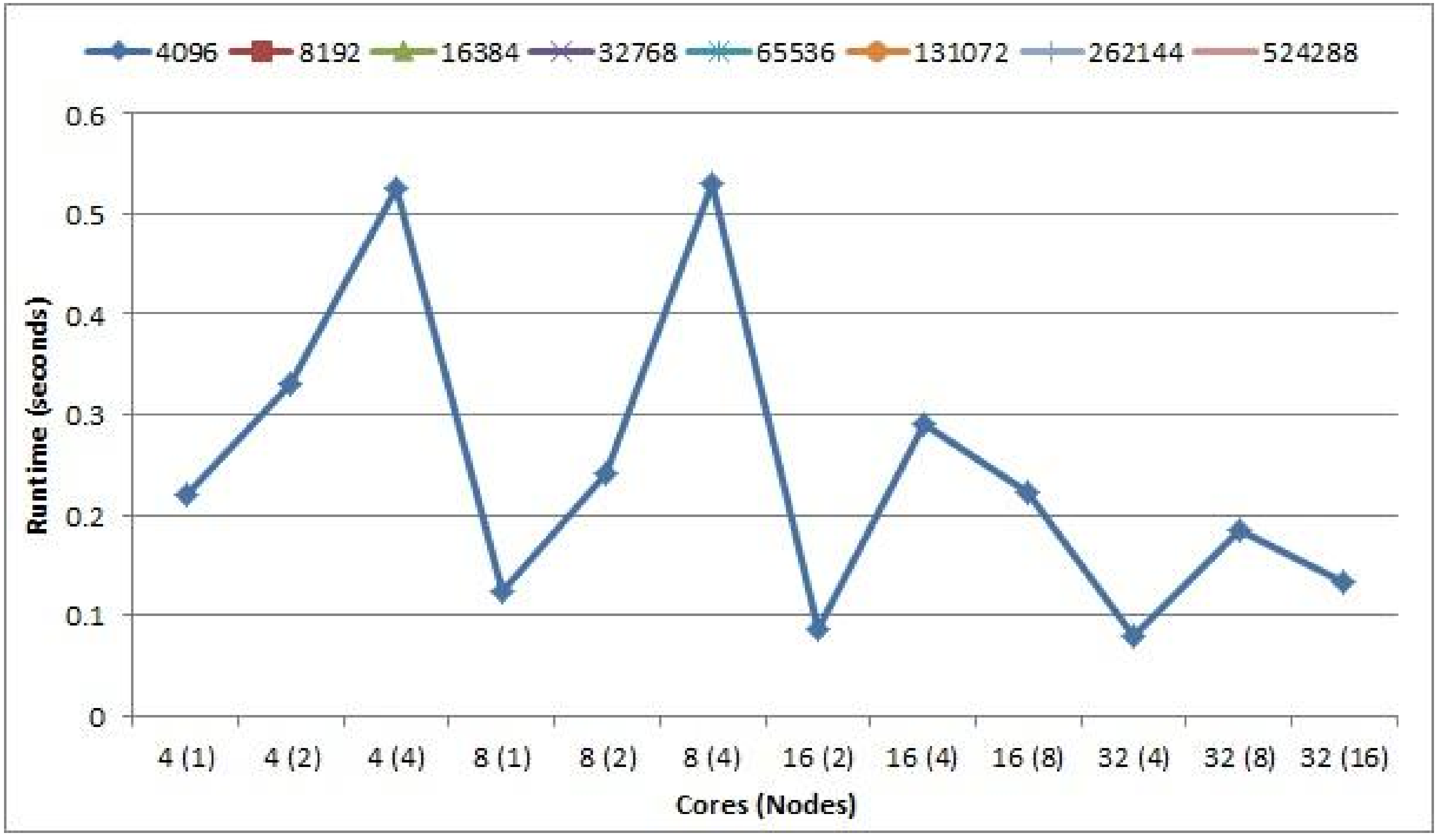
- Run the program with fixed |X| = |Y|
  - **4 cores**: 1 node, 2 nodes, 4 nodes
  - **8 cores**: 1 node, 2 nodes, 4 nodes
  - **16 cores**: 2 nodes, 4 nodes, 8 nodes
  - **32 cores**: 4 nodes, 8 nodes, 16 nodes
  - measure and compare runtimes as the number of cores per node decreases
- Each test result will be an average of 30 runs

Experiment 3 - Setup

- IBM (8 cores per processor)
  - Number of nodes = 128
  - PowerEdge C6100 - dual quad-core Compute Nodes
  - Processor Description:
    - 8x2.27GHz Intel Xeon L5520 "Westmere" (Nehalem-EP) Processor Cores
    - Main memory size: 24576 Mbytes
    - Instruction cache size: 128 Kbytes
    - Data cache size: 128 Kbytes
  - InfiniBand Mellanox Technologies MT26428 Network Card

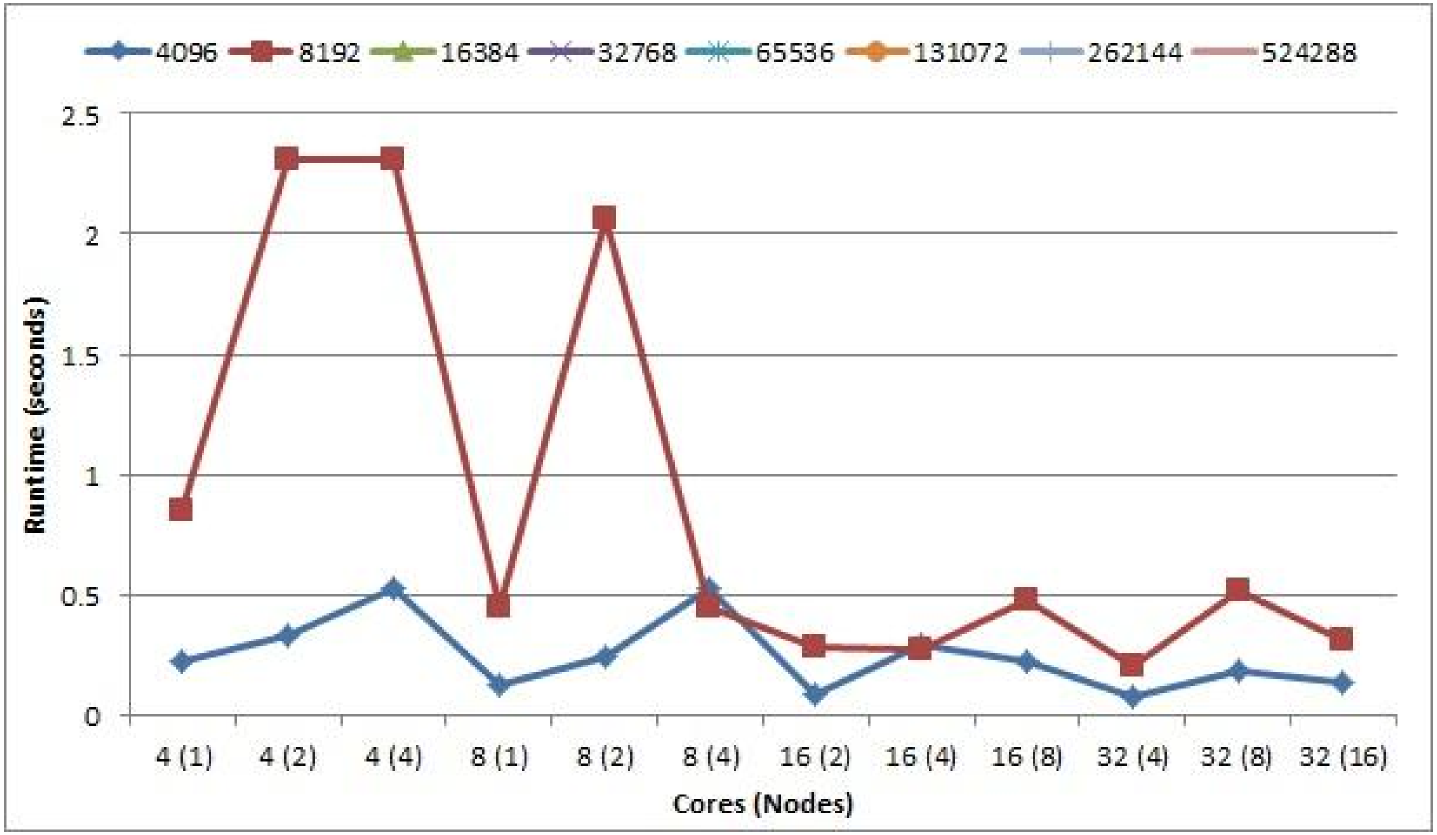
Experiment 3 - Results

- |X| = 4096
  - unusual runtimes with 16 (8) and 32 (16)



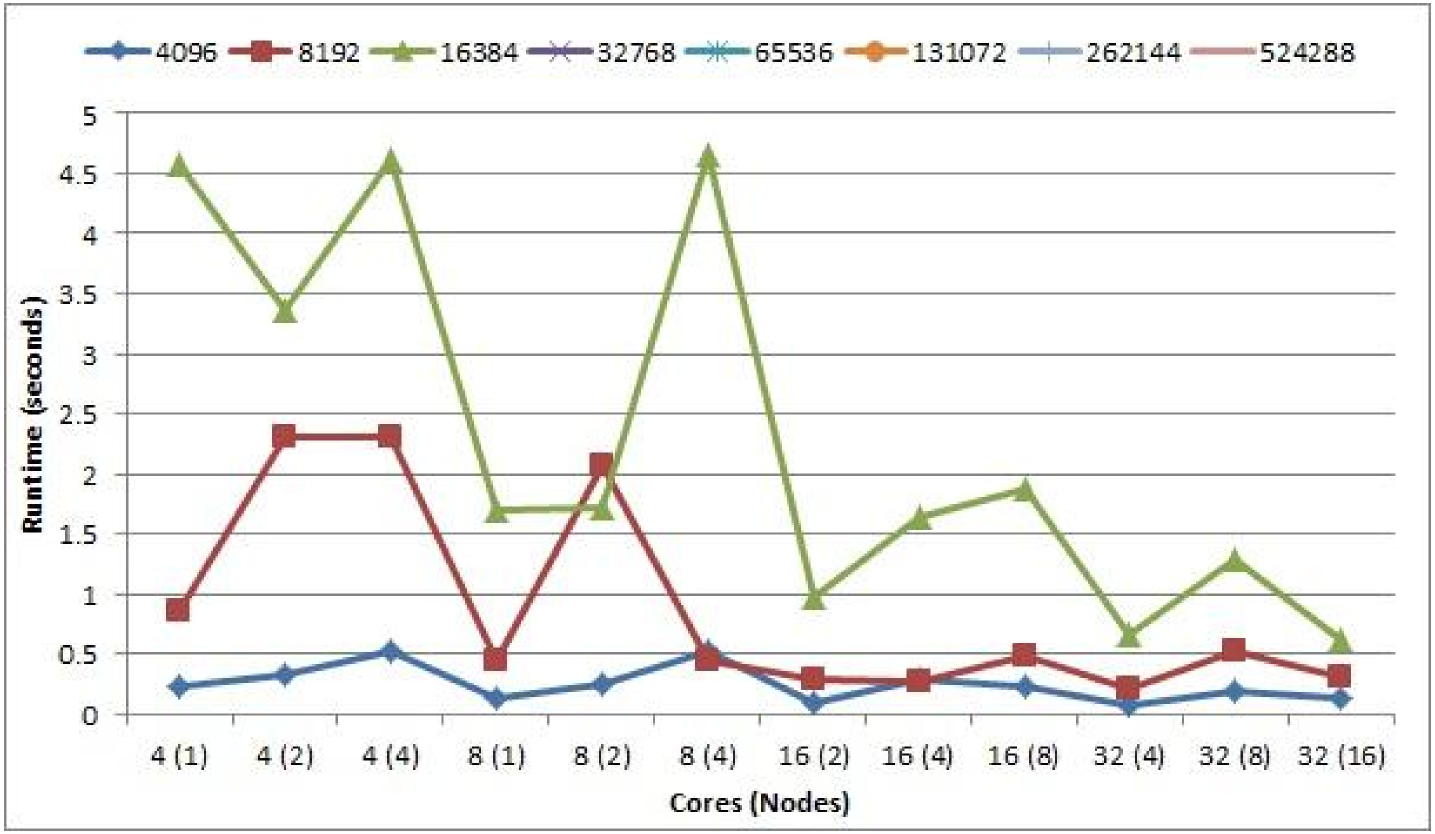
Experiment 3 - Results

- |X| = 8192
  - unusual runtimes with 8 (4) and 32 (16)



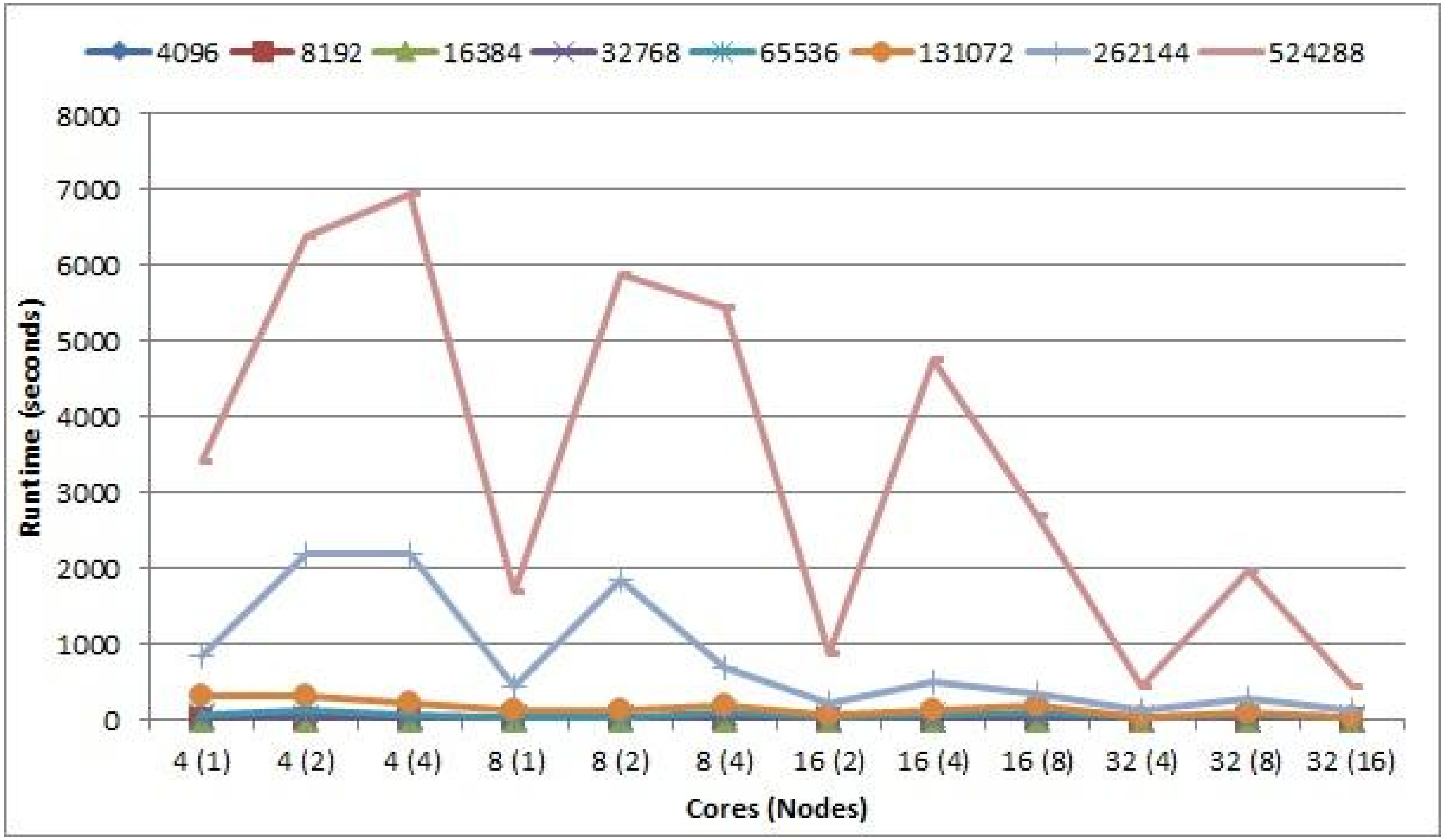
Experiment 3 - Results

- |X| = 16384
  - unusual runtimes with 4 (2) and 32 (16)



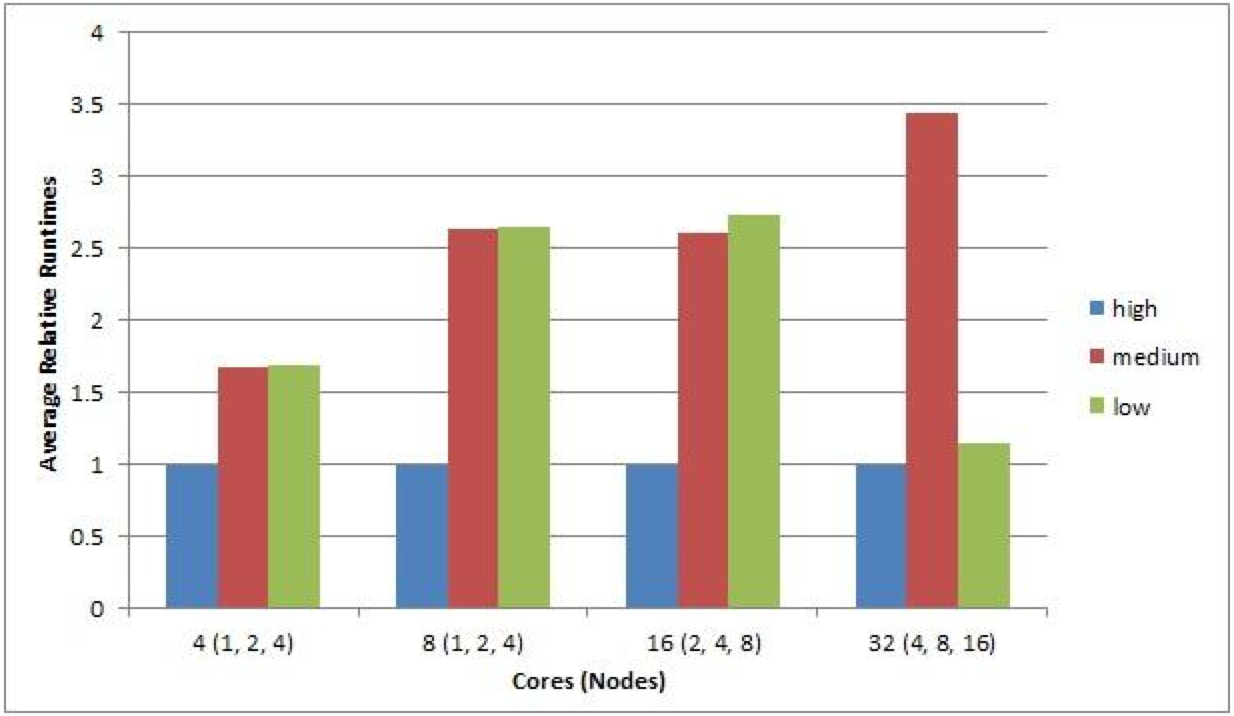
Experiment 3 - Results

- |X| = 524288
    - unusual runtimes with 8 (4), 16 (8), and 32 (16)



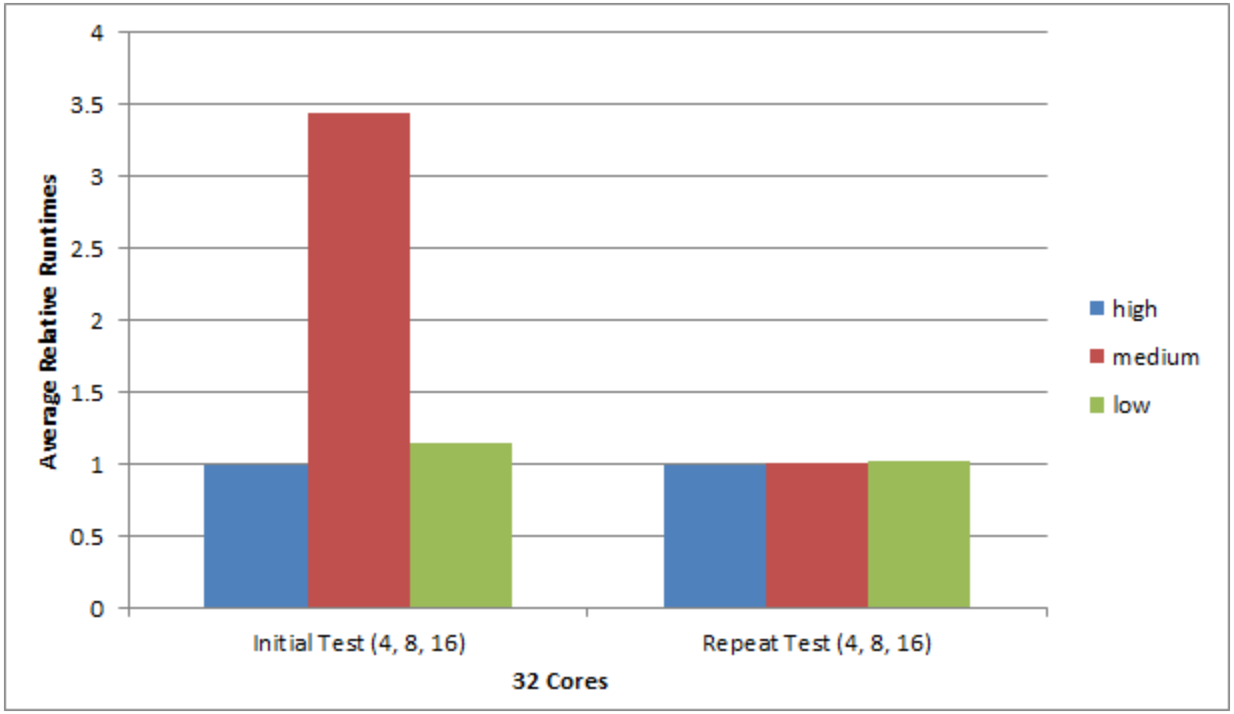
Experiment 3 - Findings

- On average, runtimes increase more going from high concentration to medium concentration, rather than medium to low
    - Unusual runtimes with 32 cores, going from 8 nodes to 16 nodes



Experiment 3 - Findings

- Tests were repeated for 32 cores
  - Unusually good runtimes with 8 nodes and 16 nodes
  - Inability to monitor or control the network traffic makes for difficult analyses of the effects of varying distributions of cores across nodes



Future Extensions

1. Scalability
   - allow larger sequences to be aligned
   - construct table in blocks, retaining the last column and last row after computing each block
2. Construct Alignment
   - may require I/O
     - will substantially slow down the program
3. Local Sequence Alignments
   - Smith–Waterman algorithm
     - compares segments of sequences and optimizes the similarity score

Questions?

- References
  - Parallel biological sequence comparison using prefix computations
    - Srinivas Aluru, Natsuhiko Futamura, and Kishan Mehrotra
  - Computational Biology on Parallel Computers
    - Srinivas Aluru
  - http://www.cs.hunter.cuny.edu/