

A Parallel, High Performance Implementation of the Dot Plot Algorithm

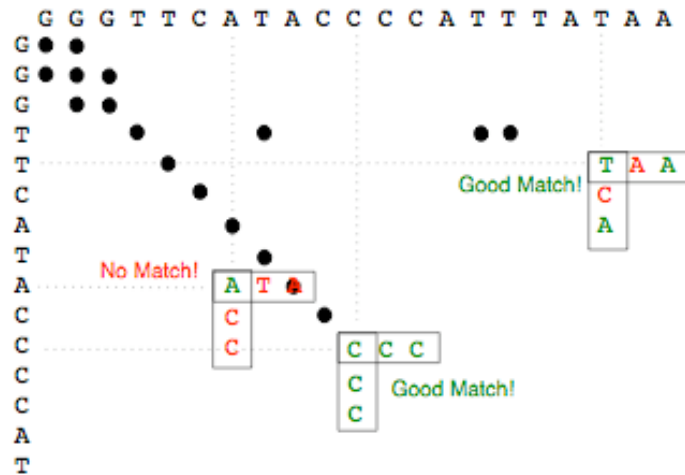
Chris Mueller

July 8, 2004

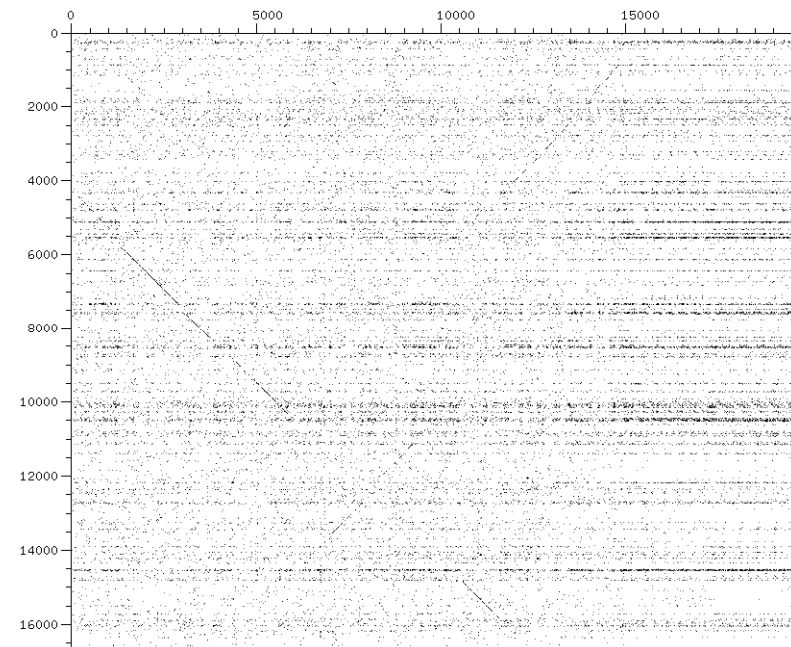
Overview

- Motivation
 - Availability of large sequences
 - Dot plot offers an effective direct method of comparing sequences
 - Current tools do not scale well
- Goals
 - Take advantage of modern processor features to find the current practical limits of the technique
 - Study how well the dot plot visualization scales to large data sets on large and high-resolution displays
 - Constrain data to DNA

Dotplot Overview



Basic Algorithm



Dotplot comparing the human and fly mitochondrial genomes (generated by DOTTER)

```
qseq, sseq = sequences
win = number of elements to compare for each point
Strig = number of matches required for a point
```

```
for each q in qseq:
  for each s in sseq:
    if CompareWindow(qseq[q:q+win], s[s:s+win], strig):
      AddDot(q, s)
```

Existing Tools

- Web Based
 - Java and CGI based tools exist
- Standalone
 - DOTTER (Sonnhammer)
- Precomputed
 - Mitochondrial comparison matrix

Optimization Strategy

- Better algorithms?
- Parallelism
 - Instruction level (SIMD/data parallel)
 - Processor Level (multi-processor/threads)
 - Machine Level (clusters)
- Memory
 - Optimize for memory throughput

SIMD

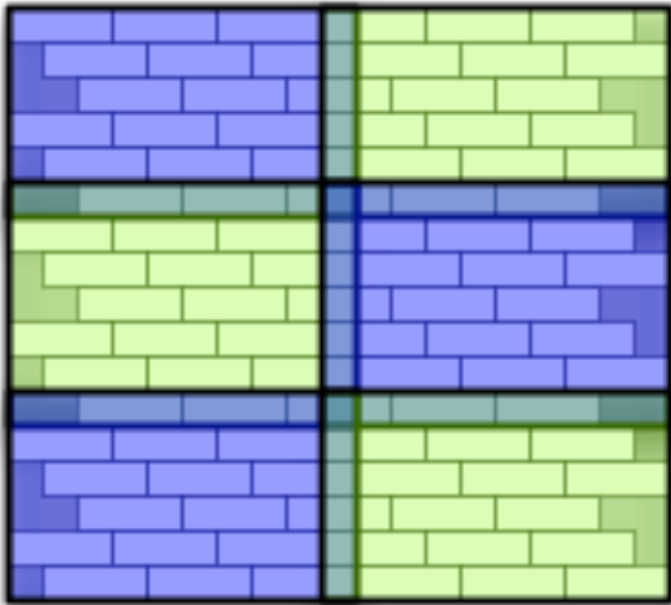
- Single Instruction, Multiple data
- Perform the same operation on many data items at once.

Normal	SIMD					
<table border="1"><tr><td>3</td></tr></table>	3	<table border="1"><tr><td>3</td><td>2</td><td>1</td><td>4</td></tr></table>	3	2	1	4
3						
3	2	1	4			
+ <table border="1"><tr><td>2</td></tr></table>	2	<table border="1"><tr><td>2</td><td>4</td><td>5</td><td>9</td></tr></table>	2	4	5	9
2						
2	4	5	9			
<hr/>						
<table border="1"><tr><td>5</td></tr></table>	5	<table border="1"><tr><td>5</td><td>6</td><td>6</td><td>13</td></tr></table>	5	6	6	13
5						
5	6	6	13			

(one instruction)

Block-Level Parallelism

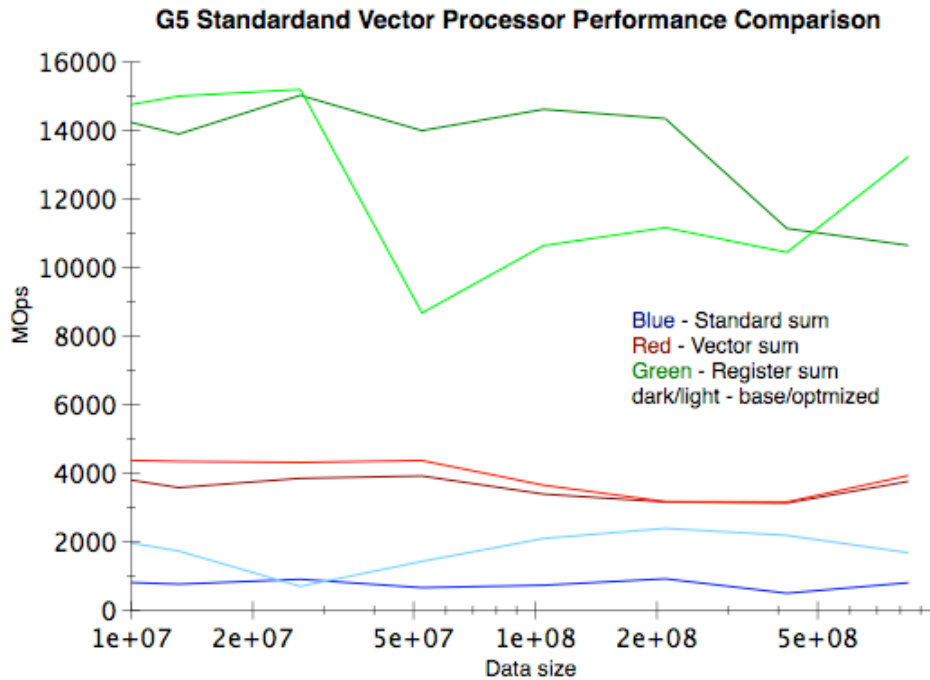
Idea: Exploit the independence of regions within the dot plot



Each block can be assigned to a different processor

Overlap prevents gaps by fully computing each possible window

Expectations



Basic Metric is ops:
base pair comparison/second

We have 2 data streams that perform
1.5 operations/load. There is also an
infrequent store operation when there is
a match.

← We should expect performance around 1.5 Gops

Green shows vector performance when data is all in registers
Red shows vector performance when data is read from memory
Blue shows performance of the standard processor

Results

	Base	SIMD 1	SIMD 2	Thread
Ideal	140	1163	1163	2193
NFS	88	370	400	-
NFS Touch	88	-	446	891
Local	-	500	731	-
Local Touch	90	-	881	1868

	Ideal Speedup	Real Speedup	Ideal/Real Throughput
SIMD	8.3x	9.7x	75%
Thread	15x	18.1x	77%
Thread (large data)	13.3	21.2	85%

- Base is a direct port of the DOTTER algorithm
- SIMD 1 is the SIMD algorithm using a sparse matrix data structure based on STL vectors
- SIMD 2 is the SIMD algorithm using a binary format and memory mapped output files
- Thread is the SIMD 2 algorithm on 2 Processors

Conclusions

- Processing large genomes using the dot plot is possible. The large comparisons here compared bacterial genomes with ~4 Mbp in about an hour on 2 processors
- Memory throughput is the bottleneck.

Visualization

- Render to PDF
- Algorithm 1
 - Display each dot
- Algorithm 2
 - Generate lines for each contiguous diagonal
 - For large datasets, this approach scales well (need more data, though :))