GenIndex: an open source parallel program for enumerating and locating words in a genome

Huian Li, David Hart, Matthias Mueller, Ulf Markward, Craig Stewart

August 3, 2009
Contents

• Motivation
• Serial algorithm
• Parallel implementation
• Performance Analysis
• Conclusion
Motivation

Question from a Biology professor:

*Given a word length, is the computational task of scanning a DNA sequence and recording the positions of all possible words trivial?*

5’ TAGCCGTGGCGGAGCCTCTTGGCTTTGTTTATTC 3’
**Serial algorithm**

- **Straightforward implementation:** Binary coding for A, C, G, T. For example:
  - A: 00
  - C: 01
  - G: 10
  - T: 11

```
  5 * 10 * 15 * 20 *
T A G C C G T G G C G G A G C C T C T T G  ...
110010010110111010011010001001011101111110 ...
```
Serial algorithm

- Given a sequence and a word length $k$, in order to list all possible words, we scan the sequence once from left to right.

```
  5  10  15  20
  *  *  *  *
TAGCCTGGGGAGCCTCTTG...
110010010110111010011010001001011101111110...
TAGC11001001
AGC00100101
GCCG10010110
CTTG01111110...
```
Serial algorithm

5         10        15        20
*         *         *         *

TAGC

C

GTG

G

CG

G

AGC

C

TCT

T

G

T

A

G

C

C

G

T

G

G

C

G

G

A

G

C

C

T

C

T

T

G

...

1100100101110111010011010001001011101111110...

TAGC

11001001

AGCC

00100101

ENCODE("AGCC") =
ENCODE("TAGC") & MASK << 2 | ENCODE('C')

MASK = 4^{k-1} – 1 = 111111 (in case of k = 4)
Serial algorithm

- This essentially becomes a sorting problem, since each word is now converted into an integer.
- Each word is associated with its position information: \((\text{Encoded Word}, \text{Position})\)
- Sorting has to be stable so that for the same words, their positions will be in a certain order.
Serial algorithm

Implementation details:

• Words & positions are stored in a `long long` integer (8 bytes = 64 bits)
• Hash table with a linked list for each entry
• Space required for all words in given sequence is pre-allocated, instead of `malloc` one by one
• Mostly AND, OR and SHIFT-LEFT operations.
Word frequencies

![Frequency of all words of length 4 in a sequence of length 81819302](image1)

![Frequency of all words of length 4 in a sequence of length 81819302](image2)
Word distribution

![Distributions of all 4-long words in a 808-long sequence](image)
Motivation for parallel implementation

Another question from Biology professor:

*How about the human genome?*

Fact: Human genome includes about 3 billion DNA bases.
Parallel implementation: input

Large dataset input:

- Each process reads its own partition from the input file.
- Boundary area between neighboring processes has to be considered.
Parallel implementation: load balancing

Computation and load balancing:
1. Each process deals with its own piece of data
2. All processes perform global sorting
   - Straightforward implementation: binary tree merge sorting
   - Possible solution but could be problematic
   - Ideal solution leading to load balancing
Parallel implementation: load balancing

Straightforward implementation: binary tree merge sorting
Parallel implementation: load balancing

Possible solution but could be problematic:

- Straightforward solution: partition word range \([0, 4^k]\) equally, so each process has

\[
\left[ \frac{4^k \times i}{n}, \frac{4^k \times (i+1)}{n} \right), \text{where } i = 0, 1, \ldots, n-1
\]
Parallel implementation: load balancing

Implementation of the straightforward solution:

- Problem is that some words occur more often than others, leading to different memory requests for different processes.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAA: 5, 9</td>
<td>CAAA: 19</td>
</tr>
<tr>
<td>AAAC: 22, 37</td>
<td>CAAC: 12, 47</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ATTG: 101</td>
<td>CTTG: 201</td>
</tr>
<tr>
<td>ATTT: 80</td>
<td>CTTT: 26</td>
</tr>
<tr>
<td>GAAA: 4, 8</td>
<td>GTTG: 88</td>
</tr>
<tr>
<td>GAAC: 67, 72</td>
<td>GTTT: 53</td>
</tr>
<tr>
<td>TAAA: 35, 93</td>
<td>TTTG: 40, 87</td>
</tr>
<tr>
<td>...</td>
<td>TTTT: 15, 30</td>
</tr>
</tbody>
</table>
Parallel implementation: load balancing

Ideal solution leading to load balancing:
• partition the total number of words $L-k+1$ equally, so each process has $(L-k+1)/n$ words, where $L$ is the length of given sequence, $k$ is the given word length.

Implementation:
1. After each process scanned its own piece, we know that:
$$\sum_{x=0}^{4^k-1} f(W_x, P_i) = \frac{L-k+1}{n} , \text{ where } i = 0, 1, \ldots, n-1$$

2. We divide the word range $[0, 4^k)$ into many small divisions with total divisions of $d$ (where $d \gg n$):
$$\sum_{j=0}^{d-1} \sum_{x=\frac{4^k}{d} \times j}^{\frac{4^k}{d} \times (j+1)-1} f(W_x, P_i) = \frac{L-k+1}{n} , \text{ where } i = 0, 1, \ldots, n-1$$
Parallel implementation: load balancing

Implementation:

3. The number of words in each small division as below:

\[ T(i, j) = \sum_{x=4^k \times d \times j}^{4^k \times (j+1)-1} f(W_x, P_i) \], where \( i = 0, 1, \ldots, n-1 \) and \( j = 0, 1, \ldots, d-1 \)

4. The total number of words in each small division across all processes will be:

\[ T(j) = \sum_{i=0}^{n-1} \sum_{x=4^k \times d \times j}^{4^k \times (j+1)-1} f(W_x, P_i) \], where \( j = 0, 1, \ldots, d-1 \)
Parallel implementation: load balancing

Implementation:

5. Find an array of boundary $B$, such that:

$$\sum_{j=B(m)}^{B(m+1)} T(j) = \frac{L - k + 1}{n}$$

where $B(0) = 0$,

$B(0 < m < n) = \text{any number in } (0, d-1)$,

and $B(n) = d-1$

6. Process $P_i$ should have all words in:

$[B(i), B(i+1)],\text{ where } i = 0, 1, \ldots, n-1.$
Parallel implementation: load balancing
Parallel implementation: output

Output:

- Each process creates its own output file
- If necessary, all files can concatenate into one single file, while keeping the order

<table>
<thead>
<tr>
<th>AAAA: 5, 9, 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAC: 22, 37</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

...  

...  

...  

<table>
<thead>
<tr>
<th>TTTG: 15, 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTTT: 40, 87</td>
</tr>
</tbody>
</table>
Testbed
Testbed specification

- Consists of 768 IBM JS21 blades
- On each blade:
  - 2 dual-core PowerPC CPUs @ 2.5GHz
  - 8 GB memory
  - SUSE Linux Enterprise Server 9 (ppc)
- Interconnect: Myrinet
- Parallel environment: MPI
Performance analysis

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Number of processes</th>
<th>D. melanogaster</th>
<th>H. sapiens</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$k=6$</td>
<td>$k=25$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>95</td>
<td>23203</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>53</td>
<td>5770</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>29</td>
<td>1500</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>15</td>
<td>386</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>9</td>
<td>107</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>512</td>
<td>512</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

Timings of running against two datasets on BigRed using 1 PPN (SECONDS)
## Performance analysis

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Number of processes</th>
<th>D. melanogaster</th>
<th>H. sapiens</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>k=6</td>
<td>k=25</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>54</td>
<td>5958</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>31</td>
<td>1545</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>17</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>11</td>
<td>115</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>128</td>
<td>256</td>
<td>25</td>
<td>16</td>
</tr>
<tr>
<td>256</td>
<td>512</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>512</td>
<td>1024</td>
<td>34</td>
<td>37</td>
</tr>
</tbody>
</table>

Timings of running against two datasets on BigRed using 2 PPN (SECONDS)
## Performance analysis

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Number of processes</th>
<th><em>D. melanogaster</em></th>
<th></th>
<th><em>H. sapiens</em></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><em>k=6</em></td>
<td><em>k=25</em></td>
<td><em>k=6</em></td>
<td><em>k=25</em></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>37</td>
<td>1738</td>
<td>163</td>
<td>6649</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>23</td>
<td>453</td>
<td>121</td>
<td>1632</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>15</td>
<td>130</td>
<td>96</td>
<td>634</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>10</td>
<td>46</td>
<td>120</td>
<td>233</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>9</td>
<td>23</td>
<td>96</td>
<td>634</td>
</tr>
<tr>
<td>32</td>
<td>128</td>
<td>11</td>
<td>17</td>
<td>120</td>
<td>233</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>20</td>
<td>25</td>
<td>96</td>
<td>634</td>
</tr>
<tr>
<td>128</td>
<td>512</td>
<td>39</td>
<td>42</td>
<td>120</td>
<td>233</td>
</tr>
<tr>
<td>256</td>
<td>1024</td>
<td>79</td>
<td>90</td>
<td>180</td>
<td>187</td>
</tr>
<tr>
<td>512</td>
<td>2048</td>
<td>134</td>
<td>131</td>
<td>270</td>
<td>281</td>
</tr>
</tbody>
</table>

Timings of running against two datasets on BigRed using 4 PPN (SECONDS)
Performance analysis

Scalability in terms of node numbers

Scalability of enumerating 6-mers in *H. sapiens*
Performance analysis

Scalability in terms of node numbers

Scalability of enumerating 25-mers in *H. sapiens*
Conclusion

• Addressed questions from the biology professor 😊
• Complicate solution aroused from memory restriction.
• It can handle words of length up to 30.
• It can find often-repeated words, rarely-occurred, or even non-occurred words.
• It scales relatively well on large cluster machines.
• We recently developed a Java version for small DNA sequences, which was “our future work”. It can zoom in or zoom out to view distribution and frequencies interactively.
The End

Thank you