Supplement for: Scaling read aligners to hundreds of threads on general-purpose processors

Ben Langmead\textsuperscript{1,2}, Christopher Wilks\textsuperscript{1,2}, Valentin Antonescu\textsuperscript{1}, and Rone Charles\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Johns Hopkins University
\textsuperscript{2}Center for Computational Biology, Johns Hopkins University

June 11, 2018
Supplementary Note 1  Software versions tested

On the Broadwell system, all software was built with GCC version 5.1.0 and the TBB library used was version 4.3. On the KNL system, all software was built with GCC 5.4.0 and the TBB library used was version 2017.0.

Bowtie

- For experiments using original parsing (O) and original TinyThread++ lock type:
  - Source code: baseline_paper_v2 tag from Bowtie repo:
    https://github.com/BenLangmead/bowtie
  - Compile-time preprocessor macros:
    
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
  - Alignment-time command-line parameters:
    -I 250 -X 800

- For experiments using original parsing (O) and TBB standard lock type:
  - Source code: same as for, TinyThread++ lock type above.
  - Compile-time preprocessor macros:
    
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    WITH_TBB=1
    NO_SPINLOCK=1
  - Alignment-time command-line parameters: same as for TinyThread++ lock type above.

- For experiments using original parsing (O) and TBB spin lock type:
  - Source code: same as for, TinyThread++ lock type above.
- Compile-time preprocessor macros:

```
WITH_THREAD_PROFILING=1
EXTRA_FLAGS="-DUSE_FINE_TIMER"
WITH_TBB=1
```

- Alignment-time command-line parameters: same as for TinyThread++ lock type above.

- For experiments using original parsing (O) and TBB queueing lock type:

  - Source code: same as for TinyThread++ lock type above.
  - Compile-time preprocessor macros:

```
WITH_THREAD_PROFILING=1
EXTRA_FLAGS="-DUSE_FINE_TIMER"
WITH_TBB=1
NO_SPINLOCK=1
WITH_QUEUELOCK=1
```

  - Alignment-time command-line parameters: same as for TinyThread++ lock type above.

- For deferred parsing (D) experiments:

  - Source code: parsing_paper_v2 tag from Bowtie repo: https://github.com/BenLangmead/bowtie

  - Compile-time preprocessor macros:

```
WITH_THREAD_PROFILING=1
EXTRA_FLAGS="-DUSE_FINE_TIMER"
WITH_TBB=1
NO_SPINLOCK=1
WITH_QUEUELOCK=1
```

  - Alignment-time command-line parameters:
For deferred batch parsing (B) experiments:

- Source code: same as for deferred parsing (D) above.
- Compile-time preprocessor macros: same as for deferred parsing (D) above.
- Alignment-time command-line parameters:
  
  
  -I 250 -X 800
  --reads-per-batch 32

For deferred batch parsing with separate input/output thread (B*) experiments:

- Source code: queue tag from Bowtie repo:
- Compile-time preprocessor macros: same as for batch parsing (B) above.
- Alignment-time command-line parameters:

  -I 250 -X 800
  --reads-per-batch 32

For MP baseline experiments:

- Source code: same as for deferred batch parsing (B) above.
- Compile-time preprocessor macros: same as for deferred batch parsing (B) above.
- Alignment-time command-line parameters:

  -I 250 -X 800
  --reads-per-batch 32
  --mm

For blocked-FASTQ experiments:

- Source code: blocked_paper_v2 tag from Bowtie repo:
  
  https://github.com/BenLangmead/bowtie
- Compile-time preprocessor macros: same as for deferred parsing (D) above.
- Alignment-time command-line parameters:
  
  ```
  -I 250 -X 800
  --block-bytes 12288
  --reads-per-block 70 for unpaired, or:
  --reads-per-block 44 for paired-end
  ```

- For blocked-FASTQ experiments with striped output:
  
  - Source code: `blocked_striped_paper_v2` tag from Bowtie repo:
    
    `https://github.com/BenLangmead/bowtie`
  
  - Compile-time preprocessor macros: same as for deferred parsing (D) above.
  
  - Alignment-time command-line parameters:
    
    ```
    -I 250 -X 800
    --block-bytes 12288
    --reads-per-block 70 for unpaired, or:
    --reads-per-block 44 for paired-end
    ```

**Bowtie 2**

- For experiments using original parsing (O) and original TinyThread++ lock type:
  
  - Source code: `baseline_paper_v1` tag from Bowtie 2 repo:
    
    `https://github.com/BenLangmead/bowtie2`
  
  - Compile-time preprocessor macros:
    
    ```
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    ```
  
  - Alignment-time command-line parameters:
    
    ```
    -I 250 -X 800
    ```

- For experiments using original parsing (O) and *TBB standard* lock type:
– Source code: same as for TinyThread++ lock type above.

– Compile-time preprocessor macros:

\[
\begin{align*}
\text{WITH\_THREAD\_PROFILING} &= 1 \\
\text{EXTRA\_FLAGS} &= "-DUSE\_FINE\_TIMER" \\
\text{WITH\_TBB} &= 1 \\
\text{NO\_SPINLOCK} &= 1
\end{align*}
\]

– Alignment-time command-line parameters: same as for TinyThread++ lock type above.

• For experiments using original parsing (O) and TBB spin lock type:

– Source code: same as for TinyThread++ lock type above.

– Compile-time preprocessor macros:

\[
\begin{align*}
\text{WITH\_THREAD\_PROFILING} &= 1 \\
\text{EXTRA\_FLAGS} &= "-DUSE\_FINE\_TIMER" \\
\text{WITH\_TBB} &= 1
\end{align*}
\]

– Alignment-time command-line parameters: same as for TinyThread++ lock type above.

• For experiments using original parsing (O) and TBB queueing lock type:

– Source code: same as for TinyThread++ lock type above.

– Compile-time preprocessor macros:

\[
\begin{align*}
\text{WITH\_THREAD\_PROFILING} &= 1 \\
\text{EXTRA\_FLAGS} &= "-DUSE\_FINE\_TIMER" \\
\text{WITH\_TBB} &= 1 \\
\text{NO\_SPINLOCK} &= 1 \\
\text{WITH\_QUEUELOCK} &= 1
\end{align*}
\]

– Alignment-time command-line parameters: same as for TinyThread++ lock type above.
• For deferred parsing (D) experiments:
  
  – Source code: parsing_paper_v2 tag from Bowtie 2 repo:
    
    https://github.com/BenLangmead/bowtie2
  
  – Compile-time preprocessor macros:
    
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    WITH_TBB=1
    NO_SPINLOCK=1
    WITH_QUEUELOCK=1
  
  – Alignment-time command-line parameters:
    
    -I 250 -X 800
    --reads-per-batch 1

• For deferred batch parsing (B) experiments:
  
  – Source code: same as for deferred parsing (D) above.
  
  – Compile-time preprocessor macros: same as for deferred parsing (D) above.
  
  – Alignment-time command-line parameters:
    
    -I 250 -X 800
    --reads-per-batch 32

• For MP baseline experiments:
  
  – Source code: same as for deferred batch parsing (B) above.
  
  – Compile-time preprocessor macros: same as for deferred batch parsing (B) above.
  
  – Alignment-time command-line parameters:
    
    -I 250 -X 800
    --reads-per-batch 32
    --mm
• For blocked-FASTQ experiments:
  
  – Source code: blocked_paper_v2 tag from Bowtie 2 repo:
    
    https://github.com/BenLangmead/bowtie2
  
  – Compile-time preprocessor macros: same as for deferred parsing (D) above.
  
  – Alignment-time command-line parameters:
    
    -I 250 -X 800
    --block-bytes 12288
    --reads-per-block 70 for unpaired, or:
    --reads-per-block 44 for paired-end

• For blocked-FASTQ experiments with striped output:
  
  – Source code: blocked_striped_paper_v2 tag from Bowtie 2 repo:
    
    https://github.com/BenLangmead/bowtie2
  
  – Compile-time preprocessor macros: same as for deferred parsing (D) above.
  
  – Alignment-time command-line parameters:
    
    -I 250 -X 800
    --block-bytes 12288
    --reads-per-block 70 for unpaired, or:
    --reads-per-block 44 for paired-end

HISAT

• For experiments using original parsing (O) and original TinyThread++ lock type:
  
  – Source code: baseline_paper_v1 tag from the following fork of the HISAT repo:
    
    https://github.com/BenLangmead/hisat
  
  – Compile-time preprocessor macros:
    
    WITH_THREAD_PROFILING=1
EXTRA_FLAGS="-DUSE_FINE_TIMER"

- Alignment-time command-line parameters:
  -I 250 -X 800  
  --no-spliced-alignment  
  --no-temp-splicesite

- For experiments using original parsing (O) and TBB standard lock type:
  - Source code: same as for TinyThread++ lock type above.
  - Compile-time preprocessor macros:
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    WITH_TBB=1
    NO_SPINLOCK=1
  - Alignment-time command-line parameters: same as for TinyThread++ lock type above.

- For experiments using original parsing (O) and TBB spin lock type:
  - Source code: same as for TinyThread++ lock type above.
  - Compile-time preprocessor macros:
    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    WITH_TBB=1
  - Alignment-time command-line parameters: same as for TinyThread++ lock type above.

- For experiments using original parsing (O) and TBB queueing lock type:
  - Source code: same as for TinyThread++ lock type above.
  - Compile-time preprocessor macros:
    WITH_THREAD_PROFILING=1
EXTRA_FLAGS="-DUSE_FINE_TIMER"
WITH_TBB=1
NO_SPINLOCK=1
WITH_QUEUELOCK=1

– Alignment-time command-line parameters: same as for TinyThread++ lock type above.

• For deferred parsing (D) experiments:

  – Source code: parsing_paper_v2 tag from the following fork of the HISAT repo:

    https://github.com/BenLangmead/hisat

  – Compile-time preprocessor macros:

    WITH_THREAD_PROFILING=1
    EXTRA_FLAGS="-DUSE_FINE_TIMER"
    WITH_TBB=1
    NO_SPINLOCK=1
    WITH_QUEUELOCK=1

  – Alignment-time command-line parameters:

    -I 250 -X 800
    --reads-per-batch 1
    --no-spliced-alignment
    --no-temp-splicesite

• For deferred batch parsing (B) experiments:

  – Source code: same as for deferred parsing (D) above.

  – Compile-time preprocessor macros: same as for deferred parsing (D) above.

  – Alignment-time command-line parameters:

    -I 250 -X 800
--reads-per-batch 32
--no-spliced-alignment
--no-temp-splicesite

• For MP baseline experiments:
  – Source code: same as for deferred batch parsing (B) above.
  – Compile-time preprocessor macros: same as for deferred batch parsing (B) above.
  – Alignment-time command-line parameters:
    -I 250 -X 800
    --reads-per-batch 32
    --no-spliced-alignment
    --no-temp-splicesite
    --mm

• For blocked-FASTQ experiments:
  – Source code: blocked_paper_v2 tag from the following fork of the HISAT repo:
    https://github.com/BenLangmead/hisat
  – Compile-time preprocessor macros: same as for deferred parsing (D) above.
  – Alignment-time command-line parameters:
    -I 250 -X 800
    --no-spliced-alignment
    --no-temp-splicesite
    --block-bytes 12288
    --reads-per-block 70 for unpaired, or:
    --reads-per-block 44 for paired-end

• For blocked-FASTQ experiments with striped output:
– Source code: blocked_striped_paper_v2 tag from the following fork of the HISAT repo:

https://github.com/BenLangmead/hisat

– Compile-time preprocessor macros: same as for deferred parsing (D) above.

– Alignment-time command-line parameters:

- I 250 -X 800
  --no-spliced-alignment
  --no-temp-splicesite
  --block-bytes 12288
  --reads-per-block 70 for unpaired, or:
  --reads-per-block 44 for paired-end

BWA-MEM

• For all experiments:

– Source code: chunksz_oflow branch of the following fork of the bwa repo:

https://github.com/BenLangmead/bwa

– Compile-time preprocessor macros: none

– Alignment-time command-line parameters. Note that BWA-MEM estimates the fragment length distribution from a sample of the data.
Supplementary Note 2  Read set download links

- human\_100\_300M
  - http://www.cs.jhu.edu/~langmea/resources/mix100\_1.fq.gz
  - http://www.cs.jhu.edu/~langmea/resources/mix100\_2.fq.gz

- human\_50\_300M
  - http://www.cs.jhu.edu/~langmea/resources/mix50\_1.fq.gz
  - http://www.cs.jhu.edu/~langmea/resources/mix50\_2.fq.gz

- human\_100\_block\_300M
  - http://www.cs.jhu.edu/~langmea/resources/mix100\_block\_1.fq.gz
  - http://www.cs.jhu.edu/~langmea/resources/mix100\_block\_2.fq.gz

- human\_50\_block\_300M
  - http://www.cs.jhu.edu/~langmea/resources/mix50\_block\_1.fq.gz
  - http://www.cs.jhu.edu/~langmea/resources/mix50\_block\_2.fq.gz
Supplementary Note 3  BWA-MEM chunk size fix

We noticed an issue in the BWA-MEM source code that would create issues for experiments using more than 214 threads (-t 215 or greater). Specifically, a signed 32-bit integer is used to represent the “chunk size,” i.e. the number of input bases to include in a single chunk. That number is 10 million by default, but it is multiplied by the thread count to calculate the final chunk size. This multiplication results in integer overflow for thread counts greater than 214, leading to behavior that dramatically slowed down read alignment. We fixed this issue and submitted a pull request to the BWA-MEM author. The pull request is located at https://github.com/lh3/bwa/pull/155. The work was originally done in the chunksz_oflow branch of our BWA fork: https://github.com/BenLangmead/bwa/tree/chunksz_oflow.
<table>
<thead>
<tr>
<th></th>
<th>Broadwell Unpaired</th>
<th>Broadwell Paired</th>
<th>KNL Unpaired</th>
<th>KNL Paired</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowtie</td>
<td>1,000,000</td>
<td>110,000</td>
<td>450,000</td>
<td>37,500</td>
</tr>
<tr>
<td>Bowtie 2</td>
<td>200,000</td>
<td>85,000</td>
<td>65,000</td>
<td>16,000</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td>200,000</td>
<td>85,000</td>
<td>65,000</td>
<td>16,000</td>
</tr>
<tr>
<td>HISAT</td>
<td>1,200,000</td>
<td>550,000</td>
<td>400,000</td>
<td>250,000</td>
</tr>
</tbody>
</table>

Supplementary Table 1: Number of reads per thread for all the experiments. These were chosen to ensure that all runs take about a minute or longer.
<table>
<thead>
<tr>
<th>Aligner</th>
<th>Paired</th>
<th>Unpaired</th>
<th>Broadwell (112 threads)</th>
<th>Skyline (96 threads)</th>
<th>KNL (272 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Whole human</td>
<td>Whole human</td>
<td>Whole human</td>
<td>Whole human</td>
<td>Whole human</td>
</tr>
<tr>
<td></td>
<td>Th Krd/s (mins)</td>
<td>Th Krd/s (mins)</td>
<td>Th Krd/s (mins)</td>
<td>Th Krd/s (mins)</td>
<td>Th Krd/s (mins)</td>
</tr>
<tr>
<td><strong>Bowtie</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O-parsing</td>
<td>96 129.83</td>
<td>154.05</td>
<td>24 465.01</td>
<td>43.01</td>
<td>88 98.69</td>
</tr>
<tr>
<td>TinyThread++ spin</td>
<td>96 129.83</td>
<td>154.05</td>
<td>24 465.01</td>
<td>43.01</td>
<td>88 98.69</td>
</tr>
<tr>
<td>D-parsing</td>
<td>96 145.17</td>
<td>137.77</td>
<td>96 1,643.77</td>
<td>12.23</td>
<td>112 116.49</td>
</tr>
<tr>
<td><strong>B-parsing</strong></td>
<td>96 148.28</td>
<td>134.88</td>
<td>96 1,080.64</td>
<td>11.17</td>
<td>112 116.13</td>
</tr>
<tr>
<td><strong>L-parsing</strong></td>
<td>96 141.63</td>
<td>141.21</td>
<td>96 1,705.03</td>
<td>11.73</td>
<td>108 115.01</td>
</tr>
<tr>
<td><strong>L-parsing, 2 outputs</strong></td>
<td>96 141.46</td>
<td>141.38</td>
<td>96 1,765.78</td>
<td>11.33</td>
<td>112 116.45</td>
</tr>
<tr>
<td><strong>L-parsing, 8 outputs</strong></td>
<td>96 141.86</td>
<td>140.98</td>
<td>96 1,990.44</td>
<td>11.17</td>
<td>112 116.13</td>
</tr>
<tr>
<td><strong>MP baseline</strong></td>
<td>96 150.01</td>
<td>153.84</td>
<td>96 1,433.97</td>
<td>13.95</td>
<td>112 108.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bowtie 2</strong></td>
<td>88 65.88</td>
<td>203.57</td>
<td>64 168.18</td>
<td>118.92</td>
<td>80 51.55</td>
</tr>
<tr>
<td>O-parsing</td>
<td>96 129.83</td>
<td>154.05</td>
<td>24 465.01</td>
<td>43.01</td>
<td>88 98.69</td>
</tr>
<tr>
<td>B-parsing</td>
<td>96 73.25</td>
<td>273.05</td>
<td>96 193.83</td>
<td>103.18</td>
<td>104 89.41</td>
</tr>
<tr>
<td><strong>BWA-MEM</strong></td>
<td>96 71.70</td>
<td>278.94</td>
<td>96 198.99</td>
<td>100.51</td>
<td>100 58.51</td>
</tr>
<tr>
<td><strong>HISAT</strong></td>
<td>96 71.94</td>
<td>278.02</td>
<td>96 198.99</td>
<td>100.51</td>
<td>100 58.51</td>
</tr>
<tr>
<td>O-parsing</td>
<td>96 71.94</td>
<td>278.02</td>
<td>96 198.99</td>
<td>100.51</td>
<td>100 58.51</td>
</tr>
<tr>
<td>D-parsing</td>
<td>96 71.82</td>
<td>278.48</td>
<td>96 199.17</td>
<td>100.42</td>
<td>112 59.06</td>
</tr>
<tr>
<td><strong>L-parsing</strong></td>
<td>96 71.70</td>
<td>278.94</td>
<td>96 198.99</td>
<td>100.51</td>
<td>100 58.51</td>
</tr>
<tr>
<td><strong>L-parsing, 2 outputs</strong></td>
<td>96 71.82</td>
<td>278.48</td>
<td>96 199.17</td>
<td>100.42</td>
<td>112 59.06</td>
</tr>
<tr>
<td><strong>MP baseline</strong></td>
<td>96 67.85</td>
<td>294.79</td>
<td>96 179.77</td>
<td>111.26</td>
<td>112 57.11</td>
</tr>
<tr>
<td><strong>BWA-MEM</strong></td>
<td>92 60.98</td>
<td>327.99</td>
<td>88 163.77</td>
<td>122.13</td>
<td>104 49.92</td>
</tr>
<tr>
<td><strong>HISAT</strong></td>
<td>16 137.72</td>
<td>145.22</td>
<td>16 297.79</td>
<td>67.16</td>
<td>16 89.80</td>
</tr>
<tr>
<td>O-parsing</td>
<td>16 137.82</td>
<td>144.07</td>
<td>16 308.40</td>
<td>64.85</td>
<td>16 94.10</td>
</tr>
<tr>
<td>D-parsing</td>
<td>48 344.21</td>
<td>58.10</td>
<td>48 648.73</td>
<td>30.83</td>
<td>108 144.26</td>
</tr>
<tr>
<td><strong>B-parsing</strong></td>
<td>56 543.45</td>
<td>56.80</td>
<td>56 1,289.28</td>
<td>18.36</td>
<td>64 153.24</td>
</tr>
<tr>
<td><strong>L-parsing</strong></td>
<td>96 786.31</td>
<td>25.44</td>
<td>96 1,579.81</td>
<td>12.66</td>
<td>104 563.34</td>
</tr>
<tr>
<td><strong>L-parsing, 2 outputs</strong></td>
<td>96 796.54</td>
<td>25.11</td>
<td>96 1,625.66</td>
<td>12.30</td>
<td>104 569.43</td>
</tr>
<tr>
<td><strong>L-parsing, 8 outputs</strong></td>
<td>96 800.10</td>
<td>25.00</td>
<td>96 1,670.97</td>
<td>11.97</td>
<td>104 590.66</td>
</tr>
<tr>
<td><strong>L-parsing, 16 outputs</strong></td>
<td>96 802.14</td>
<td>24.93</td>
<td>96 1,677.32</td>
<td>11.92</td>
<td>104 566.02</td>
</tr>
</tbody>
</table>

Each row reports: maximal peak throughput in thousands of reads per second ("Krd/s"), number of threads that achieved the peak ("Th"), and the number of wall-clock minutes required to align a 40-fold coverage dataset assuming 100 nt reads (or 100 x 100 pairs for paired-end scenarios) and a 3-billion bp genome ("Whole human"). For each combination of aligner, paired-end status and test system, the best and second-best throughputs are highlighted red and orange respectively. B-parsing and BWA-MEM rows are highlighted as those represent the default modes for those tools.
Supplementary Table 3: Impact of block padding on FASTQ file size. File sizes are in gigabytes.

<table>
<thead>
<tr>
<th>Input set</th>
<th>Uncompressed</th>
<th>Gzipped</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FQ (GB)</td>
<td>Blocked FQ (GB)</td>
</tr>
<tr>
<td>human_100_300M end 1</td>
<td>76.86</td>
<td>83.78</td>
</tr>
<tr>
<td>human_100_300M both ends</td>
<td>153.72</td>
<td>167.56</td>
</tr>
<tr>
<td>human_50_300M end 1</td>
<td>46.26</td>
<td>52.66</td>
</tr>
<tr>
<td>human_50_300M both ends</td>
<td>92.52</td>
<td>105.32</td>
</tr>
</tbody>
</table>
Supplementary Figure 1: Comparison of 4 lock types and multiprocessing baseline. Reads are paired-end. Results are shown for three aligners (rows) and two systems (columns). Jobs that ran for over 20 minutes are omitted. Squares indicate the point on each line yielding maximal total alignment throughput. These are summarized in Table 2 in the main text.
Supplementary Figure 2: Comparison of 3 parsing strategies and multiprocessing baseline. Reads are paired-end. Jobs that ran for over 20 minutes are omitted. Squares indicate the point on each line yielding maximal total alignment throughput and these points are summarized in Table 3 in the main text.
Supplementary Figure 3: HISAT running time versus number of striped output files on KNL. Input and output filesystems were varied; “SSD” is a local solid-state drive, “Lustre1” is a directory on a Lustre system set to use 1 stripe, “Lustre16” is a directory on a Lustre system set to use 16 stripes. “Null” means output is directed to /dev/null. Left plot shows full results, while right plot zooms in on the 8- and 16-file results. The first 16M pairs from the human 100 block 300M input set were aligned. The results indicate that running time depends much more on output filesystem than on input filesystem. Increasing the number of output files reduces running time up to 16 files. This is likely because striping reduces contention for the output lock(s). It is the output striping that achieves the gain; increasing the number of stripes using Lustre’s own striping mechanism seems only to reduce throughput. Even at 16 output-file stripes, there is a notable difference in running time depending on output file system, indicating that output can be a bottleneck for fast and scalable aligners.
Supplementary Figure 4: Paired-end-alignment comparison of B-parsing, L-parsing, L-parsing with output striped across 16 files and the MP baseline. BWA-MEM is also evaluated and compared to the Bowtie 2 configurations. Jobs that ran for over 20 minutes are omitted. Squares indicate the run for each configuration yielding greatest overall alignment throughput, also summarized in Table 4 in the main text.
Supplementary Figure 5: BWA-MEM’s peak resident set size (memory footprint) grows more quickly than Bowtie 2’s on KNL. This is likely due to BWA-MEM’s large batch size and the fact that the batch size scales linearly with the number of threads.
Supplementary Figure 6: Comparison of B-parsing, output-striped L-parsing, the MP baseline, and a version of B-parsing that uses separate threads for input and output parsing, called B*-parsing here. Input and output threads exchange records with the worker threads via queues. The “Moody Camel” multi-producer multi-consumer queue is used. Only unpaired alignment using Bowtie is assessed.