We thank our reviewers for their suggestions and comments. Following gives our answers to reviewers’ comments alongside subsequent changes made to paper highlighted in blue(reviewer1), red(reviewer2), and orange(reviewer3) colors.

**Reviewer-1**

**a-Describe contribution of paper compared to others, include related work section.**

We revised our contributions in Introduction with a distinction from state-of-the-art (lines 92,106). We included related work section from computational approach perspective and compared our approach to those methods (line 360).

**Reviewer-2**

**a-Introduction: authors make a case for transitioning from mouse to human model. However, they didn’t use human dataset.**

Our intention was to highlight scale of the problem if we were to move to human data and use this as a motivation for our optimization efforts. We revised Introduction accordingly (line 102) to clarify.

**b-How does parallelization strategy change across datasets.**

We included discussion on moving from mouse to human in “Conclusion”(line 1124).

**c-Why did authors only use 8-GPUs? It appears that performance improves (for larger n) with increasing GPU count.**

We were allowed to run jobs with maximum of 8 GPUs on the HPC system.

**d-Execution time does not saturate for n=10 as authors claim. It scales (almost perfectly) from 4 to 8 GPUs.**

We greatly appreciate our reviewer for catching this issue. We apologize for having overlooked the trend. We revised our analysis on multi-GPU in Section-8.3 (lines 1002,1011). For n-length 0-7, single GPU is sufficient to handle workload. For n-length 8, we observe saturation at 4-GPUs. For n-length 9-10, execution time reduces linearly beyond 4-GPUs.

**e-Section-6, authors mention that maximum number of threads-per-SM is constrained due to register usage. Can this be addressed?**

We thank our reviewer for suggesting this optimization opportunity. Moving some registers to shared memory is possible to increase number of ThreadBlocks/SM. Optimal decision is moving 9 of 48 registers, which leads to 4,742 bytes (134byte+4byte\*9\*128) shared memory and 156 bytes ($39\*4$) register usage per block. This way, we can launch 13 ThreadBlocks/SM instead of 10, and anticipate 1.3x speed-up assuming bank-conflict free implementation and same latency as register access for shared memory. We implemented this version, functionally verified. We launched experiments for all n-lengths and GPU configurations on May 22. As of May 24 afternoon there were 134 jobs in queue and all our jobs were still waiting. We are sorry that we were not able to include results for the improved version. We believe that as we focus on the impact of bit-wise and multi-GPU contributions, the conclusions on the benefits of these two have not changed. Ability to launch 3 more blocks would shift the saturation point for n-length of 8 to 2 or 3 GPUs.

**f-Based on analysis in last paragraph of Section-7.3, third factor seems to have most impact. Will it be possible to break down impact among three factors?**

It is difficult to break down impact of three factors as we need to track all possible sequences generated by each thread for analyzing the impact of nDn’. We believe reduction is the least significant factor among three. We included a discussion in Section-8.3(line 1072).

**g-Paragraphs 4 and 5 in Introduction provide too many details and hamper understanding**.

We agree. Detailed discussions were braking the flow. We shortened those paragraphs, which allowed us focus on contributions(line 155).

**h-Typos, grammatical errors, ill constructed sentences**

We sincerely appreciate our reviewer’s help. We addressed all and carefully checked again. “Padded-bits” is correct since those refer to bits in last byte of each sequence whose length isn’t divisible by 8.

**i-Explanation for loop-5 in Algorithm1 is missing. What do N and T signify?**

Thank you for catching this. We revised Algorithm1 and included explanation in Section-2.2(line 324).

**j-Authors provide algorithm description in Section-4.2 but not algorithm/pseudo-code.**

We included pseudo-code(Algorithm2) for GPU kernel in Section-5.2(line 523).

**k-Section-7.1 is longer than necessary.**

We agree. 32 threads-per-block discussion is redundant. We shortened Section-8.1, focused on other thread-block configurations.

**Reviewer-3**

**a-Table1 is unnecessary for technical audience.**

We replaced table with 4n in the text.

**b-Authors should include conversion time in run-time comparison.**

Thank you for your suggestion. We updated Table4 with conversion overhead (line 927-951) of 29 seconds.

**c-Algorithm2, (j-2/2) looks wrong.**

Thank you for catching this. We revised as (j-2)/2 (now Algortihm3)

**d- Why equation-2 has -1 and +1.**

We use them for cases when number of GPUs is not power of 2(line 734).

**e-Unit of memory in Table4.**

We included “byte” in line 887 (now Table 3).

**f-Explanation for loop-5 in Algorithm1.**

Please refer to Reviewer-2.i.

**g-Section-4.2 missing algorithm/pseudo-code.**

Please refer to Reviewer-2.j.