HPC Lab 3 – Smith-Waterman Algorithm

Ilia Lecha (hpc1217) and Igor Trujnara (hpc1218)
ilia.lecha@alum.esci.upf.edu, igor.trujnara@alum.esci.upf.edu

May 2023, BDBI 6th term

Overview of problem lines

Much of the initial work is done exclusively by the root process, and does not differ in any way from the original sequential program. Once that part is done, the root uses MPI_Bcast to send several integers (sequence lengths, block size) and the substitution matrix to the other processes. Then, every process calculates the number of rows to process using the number of processes and its own rank (a very standard procedure), and the root scatters the first sequence among all processes (itself included). Finally, it broadcasts the entirety of the second sequence to everyone.

The next part is the trickiest of all. Each process performs strip mining using a triple nested loop. The structure of the outer loop is quite straightforward – it iterates over block starts (that is, starts at 0 and increments by block size until the end of seq2). Inside the loop, the process receives the row above (if it's not the first), performs the normal Smith-Waterman nested loop on the current block, and sends the last row onward if appropriate. The boundaries for the algorithm need to be selected with care – the outer loop must iterate from row 1 (first non-ghost) to nrows, and the inner must traverse the block, however keeping in mind the possibility of slack (final block shorter than BS). The same precaution can be used for the sends and receives.

After the loop, there is one finishing touch left: the root uses a max-reduction to find the maximum score in the matrix, and calculates the global position of that score. The traceback part of the algorithm has been skipped entirely.

Timing

With 3560 characters of input per sequence, the sequential algorithm requires about 140 milliseconds to process the matrix, and a negligible time to process the input. The parallel algorithm takes about 130 milliseconds to process the matrix (as the calculation is shorter, but there is communication overhead), plus a very impactful 500 milliseconds to initialize all processes. This indicates that the input would need to be significantly larger to justify using this kind of parallelization. Our data is too rough to provide a precise estimate of the required size, but we estimate that it would need to be at least 2 orders of magnitude larger (i.e. in the order of 10^5).