**Although the SW algorithm is one of the most fundamental algorithms for local sequence alignment, it is inefficient in terms of memory use and execution time, making it impractical to use for a variety of applications. While heuristic approaches are frequently used to minimise execution time, they do so at the expense of the algorithm's accuracy and sensitivity.**

**As a result, approaches to reducing execution time and memory utilisation involve a variety of architectures and computing platforms.**

**One approach is to use an Intel Xeon Phi coprocessor with a MIC (Many Integrated Core architecture). It has 57 to 72 cores as well as 512-bit vector units.**

**The processor series is also equipped for parallel programming models such as MPI, and while they share some features with GPUs, they can run as a standalone computing system. A heterogeneous system with a workload divided between a CPU with two Xeon cores and a Xeon Phi coprocessor card improves performance.**

**SWAPHI is the name given to the algorithm developed specifically for the aforementioned system (Smith-Waterman Protein Database Search on Xeon Phi Coprocessors). Fine-grained parallelism and coarse-grained parallelism are involved. The former employs a 512-bit vector, whereas the latter employs multiple cores of the Xeon coprocessor card.**

**It entails creating a unique mapping between Xeon Phi cores and host threads, and the threads offload tasks like memory management and core alignment. Threads wait for one another to coordinate and finish tasks. This approach, however, is extremely energy intensive, and energy-efficient solutions are required when utilising such powerful technologies.**

**Field Programmable Gate Arrays (FPGAs) are another option for implementing the SW algorithm because they provide a high performance secure environment. To optimise sequence alignment on FGPAs, various techniques can be used. One such solution is dynamic programming, which divides a complex problem into smaller sub-problems.**

**The initialization process involves converting input sequences to elements that can be processed and communicating via Ethernet. A side matrix is also generated to reduce system workload and is useful for longer sequences. Backtracking can be used for shorter sequences. The matrix calculation employs Smith-Waterman processing elements based on FGPA (SWPE).**

**Using different tactics for deploying SWPEs, such as linear and lattice techniques, will have an impact on the algorithm's performance. The architecture of the FPGA is also important in terms of speed and efficiency. The Xilinx ZYNQ-7000 is one such FPGA that can be used to provide a multifold increase in efficiency by parallelizing the SW algorithm.**

**The proposed architecture also employs a reduced bit size for each base pair in order to increase speed, as well as divide and extend. Another FPGA-based architecture employs an optimised comparator for matching the sequences, which compares two bits at a time and aims to reduce the number of components required for the system.**

**One such method accomplishes this by combining the algorithm's score calculation and backtracking parts to generate the path while calculating the cell values in the matrix.**

**When evaluating implementations solely using OpenMP, it is discovered that if each element is determined independently as a separate task, increasing the number of threads does not result in a significant speedup because each thread has a number of small tasks, but the communication One recent implementation, CloudSW, uses Apache Spark, an open standard for computing clusters, in conjunction with SIMD instructions to implement the SIMD paradigm. It entails map-reduce tasks that are sped up with SIMD instructions. Following the mapping process, a priority queue is used to obtain K similar alignments.**

**During backtracking, the k results are reduced to give the best sequence alignment. This method is extremely scalable, particularly because it allows for an increase in the number of nodes without affecting performance. It improves performance for multiple nodes by a factor of ten.**

**However, for a single node, this technique does not result in a significant speedup when compared to other parallel OpenMP and MPI approaches.**

1