Parallel Prefiltering for Accelerating HHblits on the Convey HC-1

Michael Bromberger, Fabian Nowak

Chair for Computer Architecture and Parallel Processing
Karlsruhe Institute of Technology
Kaiserstraße 12
76131 Karlsruhe, Germany
bromberger@kit.edu
nowak@kit.edu

Abstract:
HHblits is a bioinformatics application for finding proteins with common ancestors. To achieve more sensitivity, the protein sequences of the query are not compared directly against the database protein sequences, but rather their Hidden Markov Models are compared. Thus, HHblits is very time-consuming and therefore needs to be accelerated. A multi-FPGA system such as the Convey HC-1 is a promising candidate to achieve acceleration. We present the design and implementation of a parallel coprocessor on the Convey HC-1 to accelerate HHblits after analyzing the application toward acceleration candidates. We achieve a speedup of $117.5\times$ against a sequential implementation for FPGA-suitable data sizes per kernel and negligible speedup for the entire uniprot20 protein database against an optimized SSE implementation.

1 Introduction

HMM-HMM-based lightning-fast iterative sequence search (HHblits [RBHS12]) is a bioinformatics application for finding sequences that have common ancestors. Such sequences are called homologous sequences. Homology search is a very time-consuming task due to the large amount of available data. It can be supported from algorithmic side by employing heuristics, such as comparing Hidden Markov Models (HMMs) [Edd98] instead of comparing the protein sequences directly. HMMs model multiple-sequence alignments or single sequences with regard to their structure. The Viterbi algorithm [Vit67] is used on an HMM to determine whether it can generate a given sequence. Exploiting special hardware can deliver substantial benefit. For example, the Viterbi algorithm can be accelerated by use of FPGAs [OSJM06]. Further, Farrar proposed an SSE implementation [Far07] of the Smith-Waterman algorithm [SW81] for direct sequence comparison.

The Convey HC-1 is an FPGA-extended heterogeneous high-performance system where a dual-core x86 host processor is coupled via Front-Side Bus (FSB) with a multi-FPGA coprocessor that possesses an aggregate memory bandwidth of up to 80 GB/s to its coprocessor memory. Some biological applications are already implemented on the Convey HC-

1. There is support for the BLAST algorithm [Con10] and also for the Smith-Waterman algorithm [VKC12]. Besides, an HC-1 design for creating deBruijn graphs supports assembly of sequences [ZB08]. We analyze the tool HHblits with regard to its structure, runtime and suitable candidates for acceleration and develop a highly parallel hardware architecture for the Convey HC-1 that provides massive speedup compared to sequential execution per kernel (117.5×) and performs slightly better than SSE even for the entire uniprot20 database (1.07×).

In Section 2, we describe the tool HHblits together with state of the art in accelerating sequence comparisons and then analyse the tool further. Upon the gained knowledge, we develop an architecture in Section 3 to accelerate HHblits with the Convey HC-1. We present the evaluation of our design in Section 4 and finally summarize our work in Section 5, also giving an outlook over future work.

2 The Application HHblits

The tool *HMM-HMM-based lightning-fast iterative sequence search* (HHblits) finds homologous sequences in a special large protein database [RBHS12]. Two protein sequences are homologous if both protein sequences have a common ancestor. Databases used for HHblits contain Hidden Markov Models (HMMs). HMMs cluster multiple sequences from available protein databases like uniprot when they are similar to each other. They can reflect this similarity through delete, insert and match states. For a consensus column of the clustered sequences, the match state models the probability for the occurrence of an amino acid at this position. Additionally, an HMM models transition probabilities from one state to another. Plan7 describes a structure for HMMs [Edd98]. Goal of HHblits is finding homologous sequences in a database for a given query sequence in order to draw inferences on its properties.

2.1 Application Flow and Structure

First, HHblits creates a query-HMM (cf. Figure 1, step 1) for the given query sequence. In HHblits, the query-HMM is approximated to a simplified sequence profile (step 2) for prefiltering the database in order to avoid later unprofitable CPU-intensive Viterbi searches on the processor (step 5 of HHblits), much like an accelerated version of HMMER [SB09], which is accomplished by either patterns or profiles. A profile is a matrix with probabilities for every character at each position in the sequence. By prefiltering the database using profiles, a speedup of 20× with almost full sensitivity is achievable in comparison to the unfiltered HMMER.

For the prefiltering (step 3), HHblits uses adapted versions of the accelerated Smith-Waterman SSE implementation [Far07]. The Smith-Waterman algorithm is a dynamic programming algorithm for calculating a local similarity score between two sequences [SW81].

The first step in the prefilter (step 3.1) is to calculate a locally ungapped similarity score

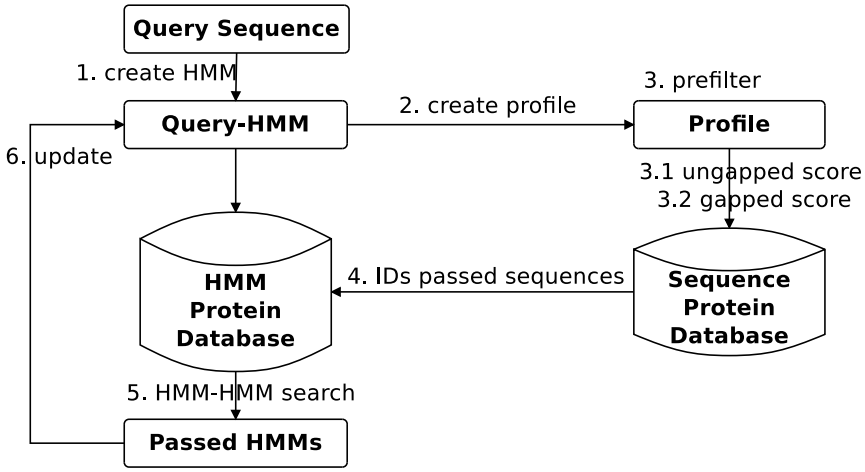

Figure 1: HMM-HMM-based lightning-fast iterative sequence search

of the query profile against each entry in the clustered database. These entries consist of an HMM and, in addition, an approximated sequence to this HMM. In the second prefilter step (step 3.2), a locally gapped score is calculated for all sequences with a previously calculated ungapped score higher than a threshold. For prefiltering, an FPGA can compute a simplified Viterbi algorithm, which also restricts the range for finding optimal Viterbi scores, thereby achieving an overall speedup of up to 182× against the original HMMER [EMdJ12].

For all HMMs whose corresponding sequence has passed (step 4) both steps of the prefilter, the application calculates a score applying the Viterbi algorithm (step 5). Given two HMMs instead of a sequence and an HMM, their modified Viterbi algorithm [Sö05] calculates the series of states that has the maximum score. If this calculated score is higher than a given threshold then this HMM is homologous to the query-HMM. Some approaches use MPI to accelerate HMMER by parallelizing the Viterbi calculations, e.g. MPI-HMMER [WQC06], and HSP-HMMER achieves due to better load balancing a speedup of 100× over MPI-HMMER [RHZ09]. GPGPUs pose another technique for accelerating the Viterbi algorithm for HMMER. The ClawHMMER approach [HHH05] implements a streaming architecture on a commercial graphics card to calculate several Viterbi comparisons in a parallel manner, being limited by programming model and available bandwidth only. A pipelined implementation for the compute-intensive Viterbi algorithm on an FPGA [OSJM06] achieved a speedup of 220× over unaccelerated HMMER. But this implementation only supports the Viterbi search for a simplified Plan7 HMM, whereas in a fully supported Plan7 HMM, only one element of the result matrix can be calculated at once. Therefore, another approach is to calculate several Viterbi searches on an FPGA in parallel resulting in a speedup of 182× against unaccelerated HMMER [OYS08].

Finally in step 6, all HMMs whose score passed the threshold in step 5 are realigned in order to update the query-HMM. Then, for the next iteration the query-HMM is updated

based on the information from all homologous HMMs and the query-HMM itself.

2.2 Runtime behavior

With a given query sequence and a clustered protein database called uniprot20, we analyzed the runtime profile of HHblits on the Intel Xeon 5138 in the Convey HC-1 using gprof. The results of this profiling are shown in Table 1. HHblits spends about 62.58 % of the entire computation time for one iteration in the function *ungapped_sse_score*, and also most of the time for two iterations. Because of these results we investigated approaches for accelerating this function. A closer look at the function revealed that only 7.178 GB of data were processed per second, although the Xeon processor runs at 2.13 GHz and hence with SSE could in theory process 16 bytes * 2.13 GHz = 34.08 GB/s. Therefore, the bottleneck lies in transferring data to the processor (to be processed in the SSE unit). The limiting factor for transferring data should be the available bandwidth of the Front-Side Bus (FSB) interface of 8.512 GB/s. Our approach is hence accelerating the function *ungapped_sse_score* by exploiting the higher memory bandwidth available for the coprocessor on the Convey HC-1 The coprocessor has roughly ten times more bandwidth than the FSB interface, which is distributed over 16 DIMMs that are accessed via eight Memory Controllers (MCs) with two ports each running at 150 MHz.

Function	**1 Iteration**			**2 Iterations**			**4 Iterations**		
	Time		Calls	Time		Calls	Time		Calls
ungapped_sse.	62.58%	49.77s	3129234	52.52%	99.69s	6258468	11.08%	200.83s	12516936
CalculatePost.	12.10%	9.62s	2155	7.09%	13.46s	3017	1.18%	21.32s	4741
Hit::Viterbi	6.37%	5.07s	601	19.63%	37.27s	3778	64.79%	1174.64s	83760
swStripedByte	3.72%	2.96s	77921	3.88%	7.37s	191721	1.10%	19.90s	452762
Other	15.23%			16.88%			21.85%		

Table 1: Profiling HHblits over one, two and four iterations on an Intel Xeon 5138

2.3 Function ungapped_sse_score

To get a clue of how we can design an architecture for the function *ungapped_sse_score*, we had a closer look at its implementation. This function calculates a locally ungapped score for the query profile and a sequence from the database. To calculate this score, a simplified version of the Smith-Waterman algorithm is used. Each element in the result matrix is only calculated depending on the left diagonal neighbor in the matrix. SSE instructions are used to calculate the elements of the result matrix. Hence, the function works on 16-byte blocks. A nested loop processes these 16-byte blocks. The outer loop iterates over all elements of the sequence. In this outer loop, for each element in the sequence an address is calculated to access a distinct line in the profile. The inner loop iterates over all 16-byte blocks of a column in the result matrix. For each block, the previously calculated left

diagonal neighbor 16-byte block is added to the corresponding block in the profile. Then an offset is subtracted from each score in the block. The 16 maximum scores are kept in a 16-byte variable. Using a striped representation of the profile, internal dependencies between elements are removed [Far07]. Thereby, only the first block in a line depends on the last block of the previous line. At the end of the function, the maximum byte value in the 16-byte variable is calculated. This is the calculated locally ungapped score.

A user-programmable FPGA on the coprocessor of the Convey HC-1 can access 16 times 8 bytes per cycle from the coprocessor memory, potentially processing 8 times more data per cycle. Moreover, four such FPGAs are available. Thereby, 512-byte blocks could be processed per cycle. For determining the maximum, special reduction circuitry can be integrated.

3 Calculating the Ungapped Score on an FPGA

The Convey HC-1 coprocessor consists of four application engines (AEs) (user FPGAs) that can be reconfigured by a user. A hardware developer creates an FPGA design by use of the Convey Personality Development Kit (PDK), specifying the functionality in a hardware description language such as VHDL. The interfaces to the Convey environment, such as memory controllers and further control processors, are all provided within the PDK. The Xilinx tools are then used to create bitstreams for the FPGAs. Normally, the same bitstream is configured on each AE. With the four AEs, we can exploit coarse-grained task-level parallelism and calculate scores for four sequences with the same query profile in parallel.

Instead of using the SSE implementation for calculating the ungapped score within HH-blits, we employ the FPGA-based coprocessor. First, data needs to be written to coprocessor memory. Then, the coprocessor can be started via specific Convey-supplied function calls that also care for configuring the FPGAs. Through these calls, the coprocessor also gets all required parameters for processing. These parameters are the address of the profile, the address of the sequence, the length of the sequence and an offset for calculating the score. Before processing the data, the FPGAs must begin fetching data from coprocessor memory and store them in FPGA-internal memories.

To exploit all the available bandwidth, we use *Binary Interleave Mode* for the Convey memory architecture. We therefore had to extend the original profile to align lines on 512-byte boundaries because in binary interleave mode, only one of the 8 Memory Controllers (MCs) has access to a certain 64-byte region of the coprocessor memory.

3.1 FPGA Design for the Convey HC-1

Our AE design for calculating the locally ungapped score exploits fined-grained data-level parallelism. Up to 128 entries of the result matrix are calculated in parallel per AE. All internal units are initialized by a controller and each unit is controlled by the data flow.

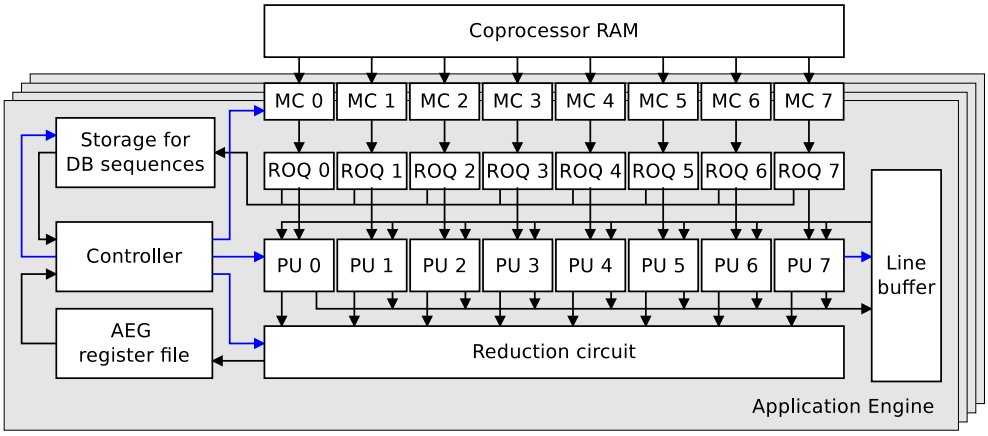

Figure 2: Parallel architecture with 8 processing units (PUs) for evaluating the ungapped score between a profile and a sequence on one FPGA

The AE design depicted in Figure 2 calculates the maximum locally ungapped score in a pipeline. Before this calculation of the score begins, the whole sequence is loaded over 8 respectively 16 ports of the memory controllers via the Read Order Queues (ROQs) to the internal storage for a sequence of the database. These queues are required to reorder the incoming data according to their request order. With the loaded sequence, we calculate for each element in this sequence the address to request the appropriate line of the profile. So the controller starts to stream all needed lines of the profile into the ROQs.

When each ROQ gets data from the profile line, the 8 respectively 16 processing units (PUs) start to calculate the entries of the result matrix, which will be described in the next paragraph. The multi-port line buffer needs to store both the previous and the current line of the result matrix because the previous line is required for calculating the current line. Whenever the processing units produce valid data, the reduction circuit begins searching the local maximum of the freshly calculated 64 respectively 128 elements of the matrix in a pipelined fashion in the reduction circuit, employing a binary-tree-like approach. In the first step, two calculated entries are compared at a time and the greater one is forwarded to the next stage. Same is done in this next stage. Finally, only one value is left. In the last stage, this value is compared against the global maximum. In case the local maximum is greater, this value becomes the next global maximum. So the reduction circuit pipeline has 7 respectively 8 pipeline stages. When all entries of the result matrix have been calculated, the global maximum value is returned to the host system via the AEG register file.

Figure 3 shows how each PU calculates 8 entries of the result matrix in parallel. First, for each byte from the profile data, the corresponding byte of the previously calculated line is added. Then an offset is subtracted from this result. Both calculations are done with saturated arithmetic together in one cycle. The resulting 8 byte values are forwarded to the reduction unit and stored in the line buffer.

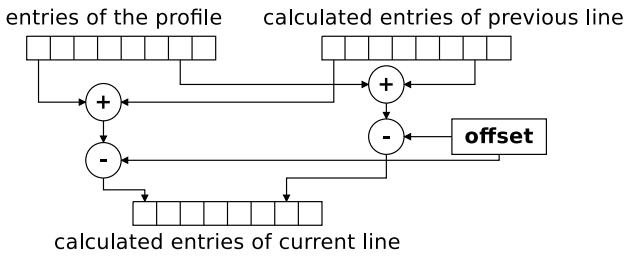

Figure 3: Parallel structure of one processing unit (PU)

After we designed and implemented the architecture, we looked at the consumption of the resources on the FPGA of an application engine. As can be seen from the results given in Table 2, we have implemented a design that does not need many of the resources available on the FPGA. In addition, the coprocessor is clocked with only 150 MHz, so there is much room available for further extensions that will still meet timing.

Resource	**8 PUs**		**16 PUs**	
Slices	21406 /	41%	25701 /	49%
Registers	52478 /	25%	68002 /	32%
LUTs	53133 /	25%	70432 /	33%
BRAMS	58 /	20%	74 /	25%
Max. freq.	201.014 MHz		198.196 MHz	

Table 2: Resource usage of our Convey HC-1 FPGA design

4 Evaluation

First we measured the average kernel execution time for calculating one ungapped score for different implementations (see Table 3). For our implementation, we had to adapt the original source code of the tool HHblits. We adapted the profile in the source code to 512-byte boundaries. Our focus is on accelerating the adapted version, but we also compare our results with the original implementation. We measured the computation time over 30 runs for a sequential and an SSE version calculating the ungapped score.

sequential		**SSE**		**Convey HC-1**	
original	adapted	original	adapted	1 AE	4 AEs
512.7 μs	607.2 μs	16.1 μs	18.9 μs	31.8 μs	11.8 μs

Table 3: Average kernel execution times over entire uniprot20 database

We achieved a speedup of $512.7/31.8 = 16.1\times$ with one AE including 16 processing units against the sequential implementation using the original profile, while the design with 8 PUs performed only a little worse. So in later steps, we will use the 16-PU design to calculate the scores. Also did we achieve a speedup over the SSE implementation on one core of $16.1/11.8 = 1.4\times$ using all four application engines.

After that we looked at how the computation time depends on the length of the sequence. So we equally distributed the sequences according to their length and calculated the average time for each sequence scope (see Figure 4). Our design using 16 processing units is faster than the SSE implementation for sequences with a length over 3499 bytes. As can be seen from Table 4, for greater sequences the performance of the FPGA design is much better ($371/311.9 = 1.2$) than that of the SSE version. The best results are achievable when using all four application engines of the coprocessor ($371/91.3 = 4.1$). Moreover, the overhead incurred on the Convey HC-1 when executing in parallel is very low as 4 AEs speed up computation by $311.9/91.3 = 3.4$, hence an efficiency of $3.4/4 = 0.85$, although parallel execution is done in bunches of four sequences each and hence depends on the longest sequence.

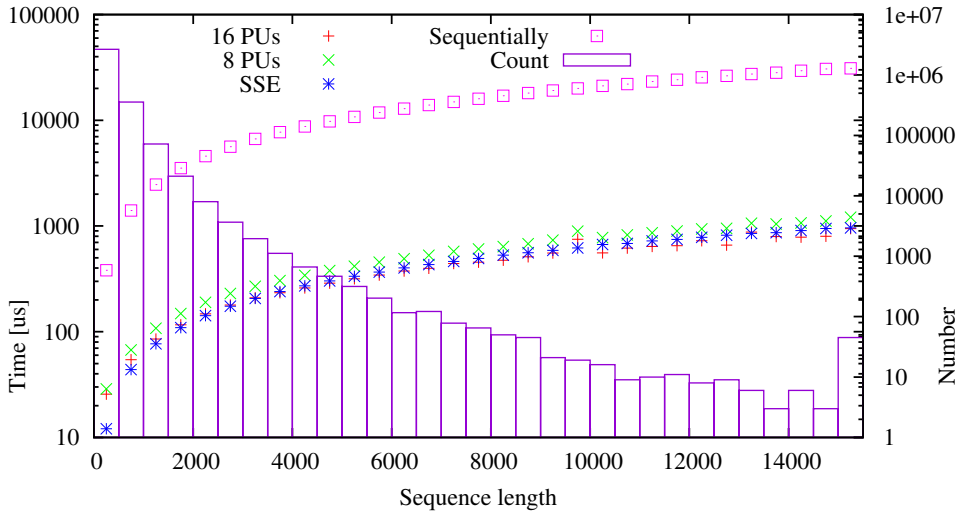

Figure 4: Average kernel execution times over varying sequence lengths with adapted profiles

sequential	**SSE**	**Convey HC-1**	
		1 AE	4 AEs
10731.9 μs	371 μs	311.9 μs	91.3 μs

Table 4: Average kernel execution times for sequences greater than 3499 bytes from the database uniprot20

We also investigated which method is useful for copying sequence data and profile data to coprocessor memory. We determined that for small data it is useful to copy data with the C library function *memcpy*. For large data, the special Convey-provided function *cny_cp_memcpy* is more useful. After having applied this knowledge, we measured computation time of the entire application HHblits for the whole database and the database without short sequences (see Table 5). For the latter, we used *cny_cp_memcpy*.

	sequential	**SSE**	**Convey HC-1**	
			1 AE	4 AEs
orig. database	2343.02 s	97.0 s	155.8 s	91.0 s
long-sequence database	64.7 s	22.4 s	22.0 s	21.7 s

Table 5: Computation time of HHblits compiled with cnyCC and activated -O3 flag

We compiled HHblits with the compiler cnyCC and with activated -O3 flag. But as we could not compile SSE instructions with cnyCC, we compiled these instructions with g++ and linked them against the other object files. So we achieve a speedup with four application engines of $1.03\times$ - $1.07\times$ against the SSE implementation and of $3.0\times$ - $25.5\times$ against the sequential version of HHblits.

5 Conclusions and Future Work

We presented a coprocessor design for supporting the application HHblits on FPGAs. After carefully analyzing the application, we developed a design for the multi-FPGA system Convey HC-1 that seems very appropriate due to its high memory bandwidth for the FPGA-based coprocessor. For longer sequences, our design performs well with a speedup of $4.1\times$ against SSE per kernel execution and of even $117.5\times$ against a sequential implementation. We already achieved some negligible speedup using all four application engines for the entire application and whole uniprot20 database. According to Amdahl's law, only a speedup of $2.289\times$ is achievable for HHblits on the Convey HC-1 when accelerating the computation of the ungapped score that consumes about 62.58% of the application time and when assuming a bandwidth-limited speedup of 10 for this part. We didn't yield this speedup yet because the used protein database includes lots of shorter sequences that do not allow fully exploiting the available memory bandwidth on the coprocessor. Accordingly, our focus is now to accelerate comparison with such shorter sequences. Our implementation saved resources on the FPGA, so we can additionally outsource other parts of HHblits, such as the second step of prefiltering. We will investigate which data ordering can achieve the best throughput on the Convey HC-1. Also will we work on further minimizing data transfer.

References

[Con10] Convey Computer Corporation. Hybrid-Core Computing for High Throughput Bioinformatics, 2010. White Paper.

[Edd98] Sean R. Eddy. Profile hidden Markov models. *Journal of Bioinformatics (Oxford University Press)*, 14:755–763, 1998.

[EMdJ12] Eusse, J.F., Moreano, N., de Melo, A. C., and Jacobi, R. P. A protein sequence analysis hardware accelerator based on divergences. *International Journal of Reconfigurable Computing (Hindawi Publishing Corp.)*, pages 4:4–4:4, Jan. 2012.

[Far07] M. Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Journal of Bioinformatics (Oxford University Press)*, 23(2):156–161, Jan. 2007.

[HHH05] Horn, D.R., Houston, M., and Hanrahan, P. ClawHMMER: A Streaming HMMer-Search Implementation. In *Supercomputing 2005: Proceedings of the ACM/IEEE SC 2005 Conference*, page 11. IEEE Computer Society, Nov. 2005.

[OSJM06] Oliver, T. F., Schmidt, B., Jakop, Y., and Maskell, D. L. Accelerating the viterbi algorithm for profile hidden markov models using reconfigurable hardware. In *Proceedings of the 6th international conference on Computational Science - Volume Part I*, ICCS '06, pages 522–529. Springer-Verlag, 2006.

[OYS08] Oliver, Tim, Yeow, Leow Yuan, and Schmidt, Bertil. Integrating FPGA acceleration into HMMer. *Journal of Parallel Computing (Elsevier Science Publishers B. V.), ISSN: 0167-8191*, 34(11):681–691, November 2008.

[RBHS12] Remmert, M., Biegert, A., Hauser, A., and Söding, J. HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. *Journal of Nature methods (Nature Publishing Group)*, 9(2):173–175, Feb. 2012.

[RHZ09] Rekapalli, B., Halloy, Ch., and Zhulin, I. B. HSP-HMMER: A tool for protein domain identification on a large scale. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 766–770, New York, NY, USA, 2009. ACM.

[SB09] Sun, Y. and Buhler, J. Designing Patterns and Profiles for Faster HMM Search. *Journal of IEEE/ACM Transactions Computational Biology and Bioinformatics (IEEE Computer Society Press)*, 6(2):232–243, Apr. 2009.

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology (Elsevier)*, 147(1):195 – 197, 1981.

[Sö05] Johannes Söding. Protein homology detection by HMM-HMM comparison. *Bioinformatics*, 21(7):951–960, 2005.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[VKC12] Vacek, G., Kelly, M., and Collins, K. Screening DNA Sequences Against a Protein Database Using a Hybrid-Core Implementation of Smith-Waterman, 2012. White Paper.

[WQC06] John Paul Walters, Bashar Qudah, and Vipin Chaudhary. Accelerating the HMMER Sequence Analysis Suite Using Conventional Processors. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 01*, AINA '06, pages 289–294, Washington, DC, USA, 2006. IEEE Computer Society.

[ZB08] Zerbino, D. R. and Birney, E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Journal of Genome Research (Cold Spring Harbor Laboratory Press)*, 18(5):821–9, 2008.