Protein Similarity Search with Subset Seeds on a Dedicated Reconfigurable Hardware

Pierre Peterlongo[1], Laurent Noé[2], Dominique Lavenier[1], Gilles Georges[1], Julien Jacques[1], Gregory Kucherov[2], and Mathieu Giraud[2]

[1] Symbiose, IRISA, INRIA, CNRS, Université Rennes 1
[2] Sequoia/Bioinfo, LIFL, INRIA, CNRS, Université Lille 1
http://www.irisa.fr/remix/arc.html

Abstract. With a sharp increase of available DNA and protein sequence data, new precise and fast similarity search methods are needed for large-scale genome and proteome comparisons. Modern seed-based techniques of similarity search (spaced seeds, multiple seeds, subset seeds) provide a better sensitivity/specificity ratio. We present an implementation of such a seed-based technique on a parallel specialized hardware embedding reconfigurable architecture (FPGA), where the FPGA is tightly connected to large capacity Flash memories. This parallel system allows large databases to be fully indexed and rapidly accessed. Compared to traditional approaches presented by the Blastp software, we obtain both a significant speed-up and better results. To the best of our knowledge, this is the first attempt to exploit efficient seed-based algorithms for parallelizing the sequence similarity search.

Keywords: sequence, similarity search, spaced seeds, subset seeds, indexing, FPGA, reconfigurable architecture, dedicated hardware.

1 Introduction

Sequence similarity search is one of the fundamental tasks in genomic research. Its main goal is to locate similar regions in DNA or protein sequences which correspond to biologically relevant *conserved* (or *homologous*) regions. A typical task, for example, is to query a genomic databank with a newly discovered DNA sequence. Observed similarities with other known genes witness their putative common biological function and direct further investigations.

With rapidly growing genomic databases, bioinformatics projects processing hundreds of gigabytes of data lead to computationally challenging tasks. Since searching for similarities between raw sequences is often the first step to more complex bioinformatics analysis, and since this process requires vast computational resources, there is a big interest in optimizing these computations.

Different approaches have been studied to reduce the computation time of sequence similarity search while keeping the same sensitivity as Blast, a commonly used software based on a seed-based heuristic [1] (see section 2.1). All these approaches exploit parallelism, though at different levels. A fine-grained

R. Wyrzykowski et al. (Eds.): PPAM 2007, LNCS 4967, pp. 1240–1248, 2008.

parallelism can be obtained through the use of SIMD instructions [2,3]. One immediate approach consists in splitting a genomic databank across a cluster of computers, like in the mpiBLAST implementation [4,5]. In that scheme, each processor performs an independent search on a part of the database. A final step merges the results. The efficiency of this coarse-grained parallelization is due to a small communication overhead between the involved computers.

Another approach is to parallelize the algorithm itself on a dedicated hardware (see section 2.2). We can exploit both a fine-grained parallelism (on a VLSI or FPGA) and a coarse-grained one (architecture with several boards). Such solutions can provide a lower cost and a better efficiency than a generic cluster. For example, a single dedicated hardware can be easier to administrate than a 64-node cluster with the same computing power.

This paper presents an implementation of a recently proposed seed-based heuristic, called *subset seeds*, on a parallel hardware designed for indexing large volumes of data such as genomic banks. Two levels of parallelism can be considered: a coarse-grained level and a fine-grained level. Here, only the first one will be discussed: it makes a subtle use of subset seeds in order to simultaneously run several partial searches on large indexes stored in a Flash memory. The fine-grained level is similar to [6] where a Blast for DNA search was proposed.

The rest of the paper is organized as follows. The next section introduces a background for sequence similarity search. Section 3 describes our parallel strategy based on subset seeds. Section 4 presents performance results obtained for a large-scale biological application and draws conclusions.

2 Similarity Searches

2.1 Seed-Based Similarity Search

An alignment between two sequences is defined in terms of a scoring function minimizing possible substitutions, deletions and insertions needed to transform one sequence into the other. Given a set of scores assigned to those edit operations, dynamic programming (DP) equations compute the best local alignments between two sequences in quadratic time [7]. Some optimizations achieve a sub-quadratic complexity [8], but the computation time remains prohibitive for whole-genome comparisons.

Most of the time, true alignments contain small patterns, called *seeds*, that are shared by the two sequences in an exact way. These seeds are used to reduce DP computations to small neighborhoods of seed occurrences. For example, the Blast [1] single-hit strategy proceeds in 3 stages (Figure 1):

- Stage 1: searching for words of size k (*seeds*) that occur in both strings,
- Stage 2: extending each seed by allowing a limited number of substitutions, and keeping only those with a score greater than a given threshold,
- Stage 3: applying the full DP algorithm to successfully extended seeds.

About five years ago, it was understood that instead of contiguous k-words, it is more advantageous for Stage 1 to use so-called *spaced seeds* that correspond

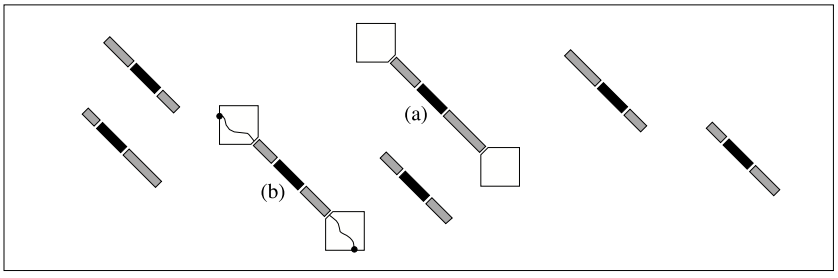

Fig. 1. Schematic view of the Blast 3-stage algorithm. Stage 1: identify exact seeds (black diagonals). Stage 2: compute seed extension allowing a small number of substitution errors (grey diagonals). Stage 3: perform a full DP computation (white squares) on remaining extended seeds. Here only seed (b) leads to an output alignment.

to gapped diagonals in the DP matrix. The idea of using spaced seeds for biological sequence comparison was first proposed in the PatternHunter software [9] and then used in a more elaborate form in the YASS software [10]. Theoretical design and usage of better seeds is an active field of research [11,12,13,14,15]. For protein search, Stage 1 of Blastp looks for words in databank sequences that are sufficiently close (in terms of the scoring function) to the query word. This strategy is captured by the general concept of *vector seeds* proposed in [13]. Recent works on seed-based protein search [16,17] apply some extended definition of this concept. An important advantage of all those seed models is the possibility to *design* appropriate seeds according to sensitivity/selectivity criteria and the class of target alignments. Moreover, instead of using a single seed, one can use *several* seeds simultaneously (so-called *multiple seeds*), to further improve the sensitivity/selectivity trade-off.

2.2 Dedicated Hardware for Similarity Search

There have been many attempts to efficiently implement the DP technique of sequence comparison in a specialized hardware. Dynamic programming equations can be projected on 2D or 1D systolic arrays [18,19,20]. A backtracking phase follows the score computation phase to build the alignment [21]. Special edition scores significantly reduce the hardware resources [19]. Between 1990 and 2006, more than twenty different architectures were proposed on VLSI or FPGA circuits [22] (see [23] or [24] for some recent works).

Hardware implementations for seed-based heuristics have been less studied. The first ASIC implementation of a seed-based heuristic was done in 1993 with the BioSCAN architecture [25]. Several FPGA implementations have been independently developed since 2003 (see [22] for a review). Some authors developed both a new algorithm (DASH) with better sensitivity than Blast as well as an FPGA implementation [26].

3 Parallel Implementation of Subset Seeds

To the best of our knowledge, no dedicated hardware has been proposed so far to efficiently implement modern seed-based sequence comparison methods. Moreover, some features of those methods are costly to implement at the software level, but can be easily implemented in hardware.

3.1 Subset Seeds for Protein Searches

The detection of an occurrence of a seed (a *hit*) in Stage 1 is done by first constructing an index for all keys corresponding to the seed. Very general seed models, such as vector seeds [13], lead to more expressive hit definitions but also to more complex and less cache-efficient implementations of this process. More specifically, whereas traditional seeds imply accessing one entry of the index for each query key (direct indexing), vector seeds require to store, for each key p, its *neighborhood*, i.e. a set of all keys that reach a given score threshold when compared to p. Therefore, this leads, for each key, to multiple accesses the main index at non-contiguous positions, inducing a larger latency.

For example, the 3-letter seed ### implies that for each 3-letter key (word) occurring at a query position, only one (identical) key should be looked up in the index. The single-hit Blastp strategy (seed $\overline{\#\#\#}^{\geq 11}$) uses the same index, but should look up, for each query key, for all possible keys that score at least 11 when compared with the query key. This strategy gives a theoretically expected number of 26 index look-ups for the Blosum-62 background distribution of amino acids.

In this work, we use the *subset seeds* model, first proposed in [27] for DNA similarity search. Subset seeds are more expressive than spaced seeds but less expressive than vector seeds. The main idea of subset seeds is that they use elements (seed letters) that distinguish between different types of mismatches. The main advantage of this model is that it provides a powerful seed definition and at the same time preserves the possibility of direct indexing.

Consider the alphabet of amino acids $\Sigma = \{C, F, Y, W, M, L, I, V, G, P, A, T, S, H, Q, E, R, K, D, N\}$. A *subset seed* is defined as a word $s_1 s_2 \dots s_m$ such that:

- each seed letter s_i denotes a partition of the alphabet Σ, grouping amino acids that can be exchanged at this position,
- a subset seed $s_1 s_2 \dots s_m$ matches an alignment fragment $(x_1, y_1)(x_2, y_2) \dots (x_m, y_m) \in (\Sigma^2)^m$ if, for each position i, amino acids x_i and y_i belong to the same set in the partition s_i.

Figure 2 provides an example of seed letters and a subset seed. The design of seed letters, i.e. partitions of the set of amino acids, will be subject of a separate publication. Once seed letters are fixed, we use the approach proposed in [27] to estimate the performance of a given seed. Theoretical estimates for sensitivity and selectivity are computed on Bernoulli background and foreground models taken from the Blosum-62 matrix model (Blocks database version 5) using the original program of [28]. Seeds achieving the best sensitivity/selectivity ratios are selected for practical evaluation (see section 4).

$$\begin{cases} b_0 = \{CFYWMLIVGPATSNHQEDRK\} \\ b_1 = \{C, FYW, MLIV, G, P, ATS, HQERK, DN\} \\ b_2 = \{C, FYW, ML, IV, G, P, A, TS, H, QE, RK, DN\} \\ b_3 = \{C, F, Y, W, M, L, I, V, G, P, A, T, S, H, Q, E, R, K, D, N\} \end{cases}$$

Fig. 2. Example of seed letters ranging from a *don't care symbol* (b_0, the whole set of amino acids) to a *match symbol* (b_3, the partition into singletons). With this alphabet, the subset seed $s = b_1 b_3 b_2$ matches the alignment fragment $(H, K)(L, L)(F, W)$.

3.2 Hardware Search Filter

As shown on Figure 3, the hardware prototype architecture, called ReMIX, is composed of several 64 GB Flash memory boards, each linked to a FPGA component. An implementation of a seed-based heuristic with fixed seeds was presented in [6]. The key point is that, in the index, each position of each seed key is stored *together with its neighborhood* (Figure 4), allowing both Stage 1 and Stage 2 to be computed without additional memory accesses. As we use a *multiple seed* (a *set* of several subset seeds), each seed requires a separate index of the database, and each index is stored on a separate memory board. In runtime, each seed is thus separately processed on distinct couples (flash memory board / FPGA), motivating the coarse-grained parallelism.

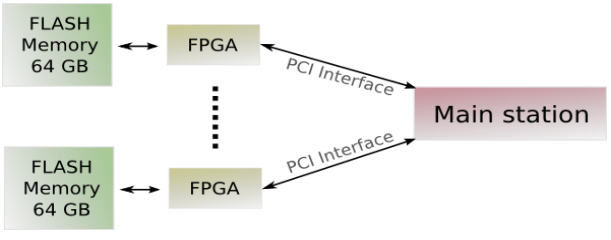

Fig. 3. Principle of the ReMIX architecture. Flash memory boards, linked to a FPGA filter, are linked to a host computer. In this experience, four boards are used.

More specifically, Algorithm 1 below shows how to find local alignments between a query and a database, closely following the heuristic described in section 2.1.

During a preprocessing phase, the databank is indexed off-line with respect to the specified seed (Figure 4). The index points to the positions of keys in the database matched by the seed s (for the Stage 1) and their neighborhoods. This index is stored in the Flash memory. The Flash technology allows all those data to be quickly accessed at the runtime. The latency of 20 μs for a random access can be hidden by a large number of successive calls. Moreover, as shown on the right part of Figure 4, for each query position, only one index look-up is needed, reducing the total latency, and thus the total filtering time.

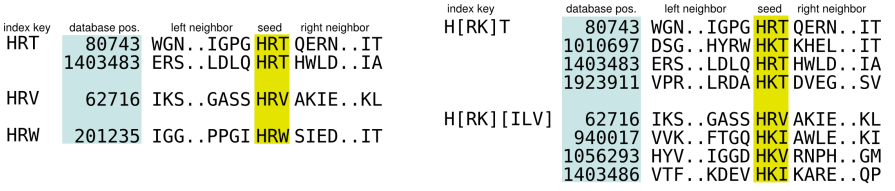

Fig. 4. Each index line, on the ReMIX architecture, contains the position of the seed key (database pos.) and its neighborhood (20 amino acids on each side). On the left, the index used on traditional ### and $\overline{\#\#\#}^{\geq 11}$ seeds gives all the database positions of occurrences of a given key. On the right, the index used with subset seeds uses less keys. There is more data for each key: the full database size remains the same.

Algorithm 1. Querying a database indexed with one seed

Input: database, seed s, query
1: index the database (with respect to s)
2: store the index into Flash memory
3: **for** each key of the query **do**
4: using the index, focus on similar key occurrences in the database (Stage 1)
5: **for** each key occurrence in the database **do**
6: using the FPGA, filter out the neighborhoods of the occurrence (Stage 2)
7: **end for**
8: **end for**
9: on host computer, perform DP computations on filtered sets of positions (Stage 3)
Output: local alignments of the query against the database

Those neighborhoods are processed on the FPGA together with the neighborhoods of the query (Stage 2). Each FPGA filter can compute approximatively 160 ungapped alignments simultaneously in 50 clock cycles. As a clock cycle is around 25 nanoseconds, up to 128 millions ungapped alignments per board can be computed each second.

Finally, the host computes the final set of alignments from the remaining set of filtered positions given by the FPGA (Stage 3). With our approach, using both dedicated hardware and subset seeds, Stage 3 is not a limiting factor, even computed in software on the host.

4 Performance Results and Conclusions

In our application, the database was extracted from the hard-masked human genome (UCSC Release hg18) translated according to the six possible reading frames. The query was a set of seven archea and bacteria proteomes deriving from a study on mitochondrial diseases. The goal of this study was to detect potential insertions of mitochondrial genes in the human genome. We selected three sets consisting respectively of 1, 2, and 4 subset seeds among the sets with

Table 1. Comparison between different seeds. The fixed seed ### is given as a reference. The number n is the number of seeds of the set: the computation is distributed over n boards. Experimental values for sensitivity (third column) are obtained through comparison with Smith-Waterman alignments on human chromosomes 1 – 11. For the other columns, we focused on the chromosome 1 (85×10^6 amino acids). All subset seeds used here were chosen to have a better sensitivity than $\overline{\#\#\#}^{\geq 11}$. On this data, the $\overline{\#\#\#}^{\geq 11}$ seed looks up the index 17 more times than any of the subset seeds (fourth column). The number of returned database positions (fifth column) estimates the selectivity, as most of them are false positive that will be filtered out in Stage 2. When several seeds and boards are used ($n > 1$), we show the results for the slowest one. In comparison, the usual single-hit Blastp implementation would take more than 3400 minutes on this dataset with a 3 GHz PC (data not shown).

Seed model	n	sensitivity	index calls ($\times10^6$)	positions returned ($\times10^9$)	filtering time (min:sec)
Fixed seed ###	1	92.87%	5.4	24	24:13
Blastp seed $\overline{\#\#\#}^{\geq 11}$	1	99.09%	92.9	246	257:50
Subset seed n°1	1	99.11%	5.4	231	216:25
Subset seeds n°2	2	99.14%	5.4	max: 109	max: 103:51
Subset seeds n°3	4	99.13%	5.4	max: 69	max: 65:35

the best sensitivity/specificity ratio and with a global sensitivity comparable to the Blastp seed $\overline{\#\#\#}^{\geq 11}$.

Using one or several boards, we performed tests parallelizing algorithm 1, except for Stage 3 that has been computed by the host computer on the merged results from all the boards. The computation time of Stage 3, lower than Stage 1 and 2, is hidden by successive calls of queries.

Time and data results are shown in Table 1. On average, FPGA filter took approximatively 67 nanoseconds for processing one index line, representing around 15 million of index entries filtered each second.

Even with traditional Blastp seeds, one ReMIX board provides a 13× speed-up over a conventional software implementation. A convenient speed-up is obtained by joining several PCI boards inside a host PC [6]. Moreover, the use of the subset seeds gives an additional speed-up due to reduced access to the memory. Here, the best results (in proportion to the number of boards) are achieved with the set of 2 subset seeds, giving a 24% speed-up over the implementation of Blastp seeds.

Thus a simple host computer equipped with 4 ReMIX boards with those subset seeds provides a $4 \times 13 \times 1.24 > 64\times$ speed-up. Thus this host is equivalent to a 64-node cluster processing traditional Blastp seeds.

As the cost of the FPGA circuit and the Flash memory declines, this solution becomes more interesting than the cluster. One may argue that the hardware design enabling the fine-grained parallelism required an additional time of development. However, the use of subset seeds is fully transparent for the architecture, as the index in the memory does only see integer keys. A better algorithmic

design of seeds provides an additional speed-up with *the very same fine-grained operators* on the same FPGA architecture.

One possible extension is to index only a part of the databases positions rather than all of them. This would reduce the index size and speed up the search. The loss in sensitivity could be limited with specially designed seeds. The efficiency of such an approach remains to be studied. Another open question is if a similar approach could be used to speed up DNA similarity searches.

Acknowledgments. Support for this work was provided by INRIA through the grant ARC Flash "Seed Optimisation and Indexing of Genomic Databases".

References

1. Altschul, S., Gish, W., Miller, W., Myers, W., Lipman, D.: Basic local alignment search tool. Journal of Molecular Biology 215(3), 403–410 (1990)
2. Rognes, T.: ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. Nucleic Acids Research 29(7), 1647–1652 (2001)
3. Farrar, M.: Striped smith–waterman speeds database searches six times over other simd implementations. Bioinformatics 23(2), 156–161 (2007)
4. Darling, A., Carey, L., Feng, W.: The design, implementation, and evaluation of mpiBLAST. In: ClusterWorld Conference and Expo (CWCE 2003) (2003)
5. Thorsen, O., Smith, B., Sosa, C.P., Jiang, K., Lin, H., Peters, A., Fen, W.: Parallel genomic sequence-search on a massively parallel system. In: Int. Conference on Computing Frontiers (CF 2007), pp. 59–68 (2007)
6. Lavenier, D., Xinchun, L., Georges, G.: Seed-based genomic sequence comparison using a FPGA/FLASH accelerator. In: Field Programmable Technology (FPT 2006), pp. 41–48 (2006)
7. Smith, T., Waterman, M.: Identification of common molecular subsequences. Journal of Molecular Biology 147, 195–197 (1981)
8. Crochemore, M., Landau, G., Ziv-Ukelson, M.: A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In: Symposium On Discrete Algorithms (SODA 2002), pp. 679–688 (2002)
9. Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)
10. Noé, L., Kucherov, G.: YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acids Research 33, W540–W543 (2005)
11. Csürös, M., Ma, B.: Rapid homology search with two-stage extension and daughter seeds. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 104–114. Springer, Heidelberg (2005)
12. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. Journal of Computer and System Sciences 70(3), 342–363 (2005)
13. Brejová, B., Brown, D., Vinar, T.: Vector seeds: An extension to spaced seeds. Journal of Computer and System Sciences 70(3), 364–380 (2005)
14. Li, M., Ma, M., Zhang, L.: Superiority and complexity of the spaced seeds. In: Symp. on Discrete Algorithms (SODA 2006), pp. 444–453 (2006)
15. Mak, D., Gelfand, Y., Benson, G.: Indel seeds for homology search. Bioinformatics 22(14), e341–e349 (2006)
16. Kisman, D., Li, M., Ma, B., Li, W.: tPatternhunter: gapped, fast and sensitive translated homology search. Bioinformatics 21(4), 542–544 (2005)

17. Brown, D.: Optimizing multiple seeds for protein homology search. IEEE Transactions on Computational Biology and Bioinformatics 2(1), 29–38 (2005)
18. Kung, H.T., Leiserson, C.: Algorithms for VLSI processors arrays. Addison-Wesley, Reading (1980)
19. Lipton, R., Lopresti, D.: In: Fuchs, H. (ed.) A systolic array for rapid string comparison, pp. 363–376. Computer Science Press, Rockville, MD (2004)
20. Chow, E., Hunkapiller, T., Peterson, J., Waterman, M.S.: Biological information signal processor. In: International Conference on Application Specific Array Processors (ASAP 1991), pp. 144–160 (1991)
21. Hoang, D.: Searching genetic databases on splash 2. In: IEEE Workshop on FPGAs for Custom Computing Machines (FCCM 1993), Napa, California, pp. 185–191 (1993)
22. Lavenier, D., Giraud, M.: Bioinformatics Applications. In: Gokhale, M.B., Graham, P.S. (eds.) Reconfigurable Computing, Springer, Heidelberg (2005)
23. Dydel, S., Bala, P.: Large scale protein sequence alignment using FPGA reprogrammable logic devices. In: Becker, J., Platzner, M., Vernalde, S. (eds.) FPL 2004. LNCS, vol. 3203, pp. 23–32. Springer, Heidelberg (2004)
24. Court, T.V., Herbordt, M.C.: Families of fpga-based accelerators for approximate string matching. Microprocessors and Microsystems 31(2), 135–145 (2007)
25. Singh, R.K., Tell, S.G., White, C.T., Hoffman, D., Chi, V.L., Erickson, B.W.: A scalable systolic multiprocessor system for analysis of biological sequences. In: Borrielo, G., Ebeling, C. (eds.) Symposium on Research on Integrated Systems, pp. 168–182 (1993)
26. Knowles, G., Gardner-Stephen, P.: A new hardware architecture for genomic and proteomic sequence alignment. In: IEEE Computational Systems Bioinformatics Conference (CSBC 2004) (2004)
27. Kucherov, G., Noé, L., Roytberg, M.: A unifying framework for seed sensitivity and its application to subset seeds. J. Bioinf. Comp. Biology 4(2), 553–569 (2006)
28. Henikoff, S., Henikoff, J.: Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA 89, 10915–10919 (1992)