14

Fine-Grained Parallel Genomic Sequence Comparison

Dominique Lavenier
ENS Cachan / IRISA Rennes
France

1. Introduction

Comparing DNA, RNA or protein sequences is a fundamental process in computational biology. The information deduced by processing genomic sequences remain the base of a large panel of bioinformatics activities such as genome assembly, gene annotation, phylogeny, prediction of 3D protein structures, meta-genomic analysis, etc.
For almost two decades, the amounts of data have steadily increased, nearly doubling every 16-18 months. Hence, from gene level analyses, bioinformatics researches have moved to full genome analysis, leading to extremely large quantities of data to process. Furthermore, recent progresses in biotechnology, such as the next generation sequencing technology able to generate billions of genomic sequences in a single day, still strengthen the needs for fast and efficient solutions.
Basically, genomic data, which are considered here, are DNA or protein sequences. A DNA sequence may be as simple as a single gene (a few thousands of nucleotides) or as complex as a full genome (three billions of nucleotides for the human genome). A protein sequence is shorter. It reflects the DNA to amino acids transcription of genes through the universal genetic code. Their lengths range from a few hundreds of amino acids to a few thousands of amino acids. The alphabet of a nucleotide sequence is composed of only 4 characters: A, C, G and T. The protein alphabet is larger and includes 20 amino acids. From a computational point of view, these data are seen as simple strings of characters.
These sequences are stored in genomic databases. SWISS-PROT and TrEMBL (Apweiler et al., 2004), for example, are two well-known protein sequence databases containing respectively 466739 and 7695149 entries (May 2009). From the DNA size, GenBank (release 171, Apr. 2009) contain more than 100 millions of sequences, representing more than 100 billions of nucleotides (Benson et al., 2008). New releases are made every two months to include new data coming from worldwide research institutes. With the exponential growth of these databases, performing computation on this mass of data is every day a more and more challenging task.
A lot of bioinformatics applications need to compare genomic sequences in their early processing steps. To illustrate our point, we briefly describe some of them in the next paragraphs. The goal is not to provide an exhaustive list, but to give, through some examples, an idea of the volume of data which are routinely processed.

Genome Assembly. Before getting the text of a genome, an initial phase is to *sequence* the long DNA molecule contained in each cell of every living organism. This is achieved by randomly breaking the DNA molecule into billions of short fragments which are re-assembled to compose the final text. Many algorithms have been proposed for reconstructing a genome from these short elements (Pop et al., 2002). However, the pre-processing is always the same: finding overlapping regions between them. This requires making intensive pair-wise comparisons to detect similarity between the beginning and the end of all fragments. In other words, assembling N fragments leads to $N^2/2$ pair-wise independent comparisons. Typically, for eukaryote organisms, N range from 10^7 to 10^8.

Database Scanning. A common task of the molecular biology is to assign a function to an unknown gene. To be functional, a protein must adopt a specific 3D shape related to its sequence of amino acids. The shape is important because it determines the function of the protein, and how it interacts with other molecules. It is assumed that two proteins with identical functions may have similar 3D structures, yielding to a similar sequence of amino acids. Even if this hypothesis is not always verified, a large number of algorithms were proposed to rapidly extract sequences (or portion of sequences) having a high similarity with a query sequence. But the scan of genomic databases is faced to the exponential growth of the data. To be able to query databases of billions of nucleotides within reasonable time (from seconds to minutes), the use of parallel systems is now the only solution.

Full Genome Comparison. Mid 2009, about 1000 genomes have been completely sequenced, and more than 4000 other genome sequencing projects are under progression (Liolios et al., 2008). By comparison, only 300 projects were referenced ten years ago. Actually, no decline in this activity is expected in the next few years. More and more genomes will come from many organisms: virus, bacterium, plants, fishes, vertebrates, etc. This avalanche of data opens the door to new ways of investigating the various genome structures. From a computational point of view, algorithms do not fundamentally differ from standard string comparison algorithms, except that the length of the sequences may seriously limit their use. Strings of hundreds of millions of characters need to be intensively processed to detect any kind of similarities. Compared to gene analysis, which can be satisfactory performed (in time) on a standard computer, genome analysis increases the complexity by several orders of magnitude.

Molecular Phylogeny. On Earth, there are millions of different living organisms. Morphological criteria and gene structure suggest that they are genetically related. Their genealogical relationships can be represented by a vast evolutionary tree. This assumption implies that different species arise from previous forms via descent, and that all organisms are connected by the passage of genes along the branches of the phylogenic tree. To build such a tree, identical (or near identical) genes present in all organisms are systematically compared. This aims to calculate a *distance* between all genes (larger the distance, older the relationship between genes). Based on these distances, trees can be constructed through different phylogenic methods. Again, the pre-processing step involves comparing precisely a large set of genomic sequences.

Next Generation Sequencing (NGS). For the last three years, the very fast improvements of sequencing machines have revolutionized the genomic research field (Shenure & Hanlee, 2008). The equivalent (in raw data) of the human genome can now be generated in a single day. Billions of nucleotides spread in millions of very short fragments (35 to 70 nucleotides) are thus available allowing a large spectrum of new large scale applications to be set up:

genome re-sequencing, meta-genomic analysis, molecular bar-coding, etc. Once again, the preliminary step often deals with intensive genomic sequence comparison.
Since the early 80's, many efforts were made to optimize the genomic sequence comparison problem, both on the software side with powerful heuristics, and on the hardware side with dedicated hardwired systems. Another important effort has also been done on the parallel side, ranging from pure parallel software implementations to highly specific parallel machines.
The goal of this chapter is to present the various strategies which are used to parallelize this essential bioinformatics task, and more specifically strategies using fine-grained parallelism. Section 2 formally introduces the problem and section 3 presents the main algorithms. The three next sections are devoted to three different technologies: VLSI and FPGA accelerators, SIMD instructions, and graphical processing units (GPU). The last section concludes the chapter.

2. The genomic sequence comparison problem

Basically, comparing two genomic sequences is equivalent to find similarities between these two elements. Similarities are symbolized by *alignments* which are the objects that biologists are able to interpret. An alignment is composed of two strings where most characters of both strings match together. For instance, consider the following alignment:

```
A G T G G T C T T A - A C G T T A C A T G T T
| | | : | | | : | |   | | | |     | | : | | |
A G T T G T C A T A T A C G T - - C A A G T T
```

The symbol **|** represents a match between two characters. The symbol **:** represents a mismatch. No symbol indicates a deletion or an insertion. In that case, this operation is referred as a gap. Given two sequences, the game is to find regions which maximize the number of consecutive matches and which represent significant biological similarities. To decide if an alignment is significant or not, a score is associated. If the score exceeds a statistically predefined threshold value, it is then considered as valid.
The score is computed as the sum of three elementary costs:

- Cost of a match
- Cost of a mismatch
- Cost of a gap

If we assign +1 for a match, -1 for a mismatch and -3 for a gap, the score of the above alignment is equal to: 17 x matches + 3 x mismatches + 3 gaps = (17×1) +(3×-1)+(3×-3) = 5. This simple scoring scheme is used for DNA sequences. The values of the match, mismatch and gap costs are given by the user and depend of the applications. To better match the biological reality, the gap cost is often calculated using an affine function giving a highest cost for the first gap and a lower cost for the following ones. Taking again the example, and setting the open gap cost to -3 and the extension gap cost to -1, the value of the score will increase to 7: the cost of the first gap stay the same, but the cost of the second gap rise to -4. For protein comparison, the match and mismatch cost is included in a single operation

called substitution given by a *substitution matrix* reflecting the mutation rate between the 20 amino acids.
Depending of the applications, different types of alignment may be considered. Figure 1 depicts the three main variations commonly used in molecular biology: global alignment, local alignment and semi-global alignment. Historically, global alignments were first studied. Global alignments try to find the best match between all characters of two sequences of similar size. They are typically used for phylogeny studies: the score of the alignment between two genes indicates their degrees of proximity.
On the other hand, local alignments aim to detect similarities of any length. Given two sequences, the comparison process aims only to detect part of the sequences having significant similarity. The difficulty is that the position and the length of the alignments are unknown, leading to explore a vast search space. Finding local similarities represents the major needs in bioinformatics. The scan of large databases is the best example. Biologists don't only want to know if there are similar items in the database, they also want to detect if their queries shares some common functionalities with other elements. As proteins (or genes) are often assemblies of different functional domains, extracting only local similarities bring pertinent biological information.

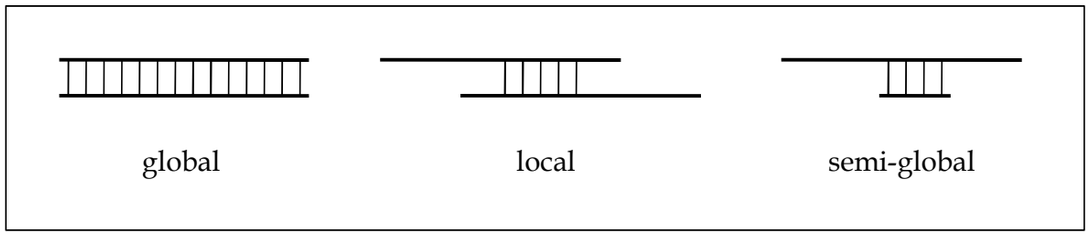

Fig. 1. Schematic representation of the three types of alignments commonly used in molecular biology

The semi-global alignments match all the characters of a small sequence over a large one. The Next Generation Sequencing (NGS) approach which generates a very large number of very short fragments is one of the main activities requiring this treatment. The goal is to map small DNA sequences on full genomes allowing only a restricted number of errors.
Having defined the comparison sequence problem as the search of alignments between two sequences, and having described the main features of an alignment, the next section focuses on the algorithmic side of the problem.

3. The main algorithms

For the last 25 years, due to the tremendous increase of the genomic field, and the growing demand for processing larger and larger amounts of data, many algorithms were proposed to search alignments. The goal, here, is not to review in detail all of them. We will only focus on the two main families which have been widely adopted by the scientific genomic community and which have been implemented on a large panel of parallel structures. The first algorithm introduced in 1970 by Needleman & Wunsch (Needleman & Wunsch, 1970) and revisited in 1981 and 1982 respectively by Smith and Waterman (Smith & Waterman, 1981) and Gotoh (Gotoh, 1982) are based on dynamic programming. They are optimal in the

way that they find the best alignments (local or global) between two sequences. But their quadratic complexity – $O(n^2)$ – make them unsuitable for processing large quantity of data. However, for some applications, such as phylogeny or search of weak similarities, there are essential, thereby justifying all the efforts among the last three decades to provide efficient parallel solutions.

By the end of the 80's, however, an important algorithmic breakthrough has emerged, based on a powerful heuristic providing extremely good results. This heuristic drastically reduces the search space by focusing on interesting points, called hits, between two sequences. Using this technique, the execution time could be decreased by nearly two orders of magnitude. Two programs have been immediately proposed to the scientific community, FASTA in 1988 (Pearson & Lipman, 1988) and BLAST in 1990 (Altschul et al., 1990). The later, through many improvements, is now the reference in the bioinformatics community (Altschul et al., 1997). It is maintained by the NCBI (National Center for Biotechnology Information) as an open-source software including parallel implementations.

3.1 Dynamic programming algorithm

The dynamic programming algorithm compares two strings of characters by computing a distance which represents the minimal cost to transform one segment into another one. As stated earlier, two elementary operations are used: the substitution and the gap operations. By using a list of such operations any segment may be transformed into any other segment. It is then possible to take the smallest number of operations required to change one segment to another as the measure of distance between them.

More formally, let $X = (x_1, x_2, \ldots x_n)$ and $Y = (y_1, y_2, \ldots y_m)$ two sequences to be compared. Let $d(x,y)$ the substitution cost to change x into y and g the gap cost. The Needleman & Wunsch algorithm is given by the following recursion:

$$D(i,j) = max \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ D(i-1,j) - g \\ D(i,j-1) - g \end{cases} \quad (1)$$

with the following initialization:

- $D(0,0) = 0$;
- $D(i,0) = H(i\text{-}1,0) - i \times g$ for $i>0$
- $D(0,i) = H(0,i\text{-}10) - i \times g$ for $i>0$

$D(i,j)$ represents the maximum similarity of the two segments ending at x_i and y_j. Thus, $D(n,m)$ gives the score representing the similarity between the strings X and Y. Higher the score, better the similarity. From the $D(n,m)$ point, a trace-back procedure can be applied to recover the alignment, as shown figure 2. In that case, all the values $D(i,j)$ must be stored in a 2D table. The trace-back procedure consists in reconstructing the optimal path from the last two characters (bottom right) to the first two characters (up left).

		A	T	T	T	G	A	C	G	T	A	T	C
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24
A	-2	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21
T	-4	-1	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
T	-6	-3	0	3	1	-1	-3	-5	-7	-9	-11	-13	-15
G	-8	-5	-2	1	2	2	0	-2	-4	-6	-8	-10	-12
A	-10	-7	-4	-1	0	1	3	1	-1	-3	-5	-7	-9
C	-12	-9	-6	-3	-2	-1	1	4	2	0	-2	-4	-6
T	-14	-11	-8	-5	-2	-3	-1	2	3	3	1	-1	-3
G	-16	-13	-10	-7	-4	-1	-3	0	3	2	2	0	-2
T	-18	-15	-12	-9	-6	-3	-2	-2	1	4	2	3	1
A	-20	-17	-14	-11	-8	-5	-2	-3	-1	2	5	3	2
T	-22	-19	-16	-13	-10	-7	-4	-3	-3	0	3	6	4
C	-24	-21	-18	-15	-12	-9	-6	-3	-4	-2	1	4	7

Fig. 2**.** Execution of the Needleman & Wunsch algorithm between two DNA sequences. The cost of a match is set to +1, the cost of a mismatch to -1 and the cost of a gap to -2. Once the similarity score is computed, a trace-back procedure permits to recover the global alignment by reconstructing the optimal path.

Remember that the Needleman & Wunsch algorithm computes a global alignment between two sequences. To find shorter similarities, or local alignments, the Smith & Waterman algorithm introduces a slight modification to the former recursion:

$$D(i,j) = max \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ D(i-1,j) - g \\ D(i,j-1) - g \\ 0 \end{cases} \quad (2)$$

with the following initialization: *D(0,0) = D(i,0) = D(0,i) = 0*

A threshold value, sets to 0, prevents the score to become negative. The effect is that if, somewhere on the 2D table, a local maximum occurs, it can reflect some local similarity. Figure 3 illustrates this situation. The word ATTGA is present in both sequences and is detected by the highest score inside the 2D table.

		C	G	T	T	G	A	A	T	T	G	A	A
	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	1	1	0	0	0	1	1
T	0	0	0	1	1	0	0	0	2	1	0	0	0
T	0	0	0	1	2	0	0	0	1	3	1	0	0
G	0	0	1	0	0	3	1	0	0	1	4	2	0
A	0	0	0	0	0	1	4	2	0	0	2	5	3
C	0	1	0	0	0	0	2	3	1	0	0	3	4
T	0	0	0	1	1	0	0	1	4	2	0	1	2
G	0	0	1	0	0	2	0	0	2	3	3	1	0
T	0	0	0	2	1	0	1	0	1	3	2	2	0
A	0	0	0	0	1	0	1	2	0	1	2	3	3
T	0	0	0	1	1	0	0	0	3	1	0	1	2
C	0	1	0	0	0	0	0	0	1	2	0	0	0

Fig. 3. Execution of the Smith & Waterman algorithm between two DNA sequences. A match is set to +1, a mismatch to -1 and a gap to -2. A trace-back procedure, starting from the highest score, permits to recover the best local alignment.

To better reflect the biological reality, Gotoh improved both algorithms by modifying the cost of N consecutive gaps. The fist gap has an open value g_{open} while the following ones have an extended gap cost g_{ext}. The recursion is modified as follows:

$$D(i,j) = max \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ V(i,j) \\ H(i,j) \\ 0 \end{cases} \quad (3)$$

$$V(i,j) = \max \begin{cases} D(i-1,j) + g_{open} \\ V(i-1,j) + g_{ext} \end{cases}$$

$$H(i,j) = \max \begin{cases} D(i,j-1) + g_{open} \\ H(i,j-1) + g_{ext} \end{cases}$$

These new equations can be applied both for searching local or global alignments. The complexity for comparing two sequences is the same and is in O(nm), where n and m

represent the length of the two genomic sequences. Note that to get only the similarity score between two sequences, it is not necessary to keep the complete 2D table in memory.

3.2 Heuristic optimization

The dynamic programming algorithm systematically explores a search space equals to $n \times m$. For genomic data mining applications which process billions of sequences, this approach cannot practically be used due to its very high computational complexity. To bypass this constraint, many heuristic algorithms have been developed having in mind to target only regions of interest. These zones can be seen as short regions (sub-sequences) in both sequences with good probabilities of match. The quality and the speed of the algorithms highly depend of the ability to detect these regions.

In the FASTA and the BLAST packages, the idea is the following: Generally, the two strings of an alignment share, at least, one identical word of W characters. These words, called seeds, generate hits between the sequences. From these hits an alignment can thus be reconstructed by extending the search on the left and right hand sides. The size of the seeds has a great influence on the search sensitivity: small seeds have a high probability to belong to all the alignments detected by programming dynamic methods. On the other hand, large seeds often miss weak similarity alignments because such alignments do not include at least one similar word of W consecutive characters. Similarly, small seeds will increase the computation time while large seeds will tend to limit it, just because of the direct relationship between the size of the seed and the number of generated hits: larger the seeds, smaller the number of hits, and smaller the time spent in computing extensions. Users are then faced to a difficult tradeoff: fast and approximate results or slow and sensitive results.

Using this technique, the search of alignments is generally split into a few distinct steps. For example, the BLAST program works as follows:

- Step 1: find hits of W character words
- Step 2: perform ungap extension
- Step 3: perform gap extension

Figure 4 illustrates the process. The first step marks the regions in the 2D space where similar words of W characters are found. These regions are called hits. The second step starts a restricted search on the hit neighborhoods. The complexity of the search is intentionally limited by considering only substitution operations. At this stage, gaps are not allowed. This step aims to investigate if a significant similarity exists near the hit before launching a full alignment computation. An intermediate score is thus calculated. If it exceeds a predefined threshold value, then the third step is run. The last step, only triggered by step 2, performs a dynamic programming on both side of the hit (see Figure 4). Again, a score is calculated. If this new score becomes greater than a statistically significant threshold value, an alignment is generated.

Algorithms based on seed heuristics have been widely adopted by biologists because of their great speed improvements compared to programming dynamic approaches. Furthermore, their sensitivity can be efficiently tuned to match the requirements of many bioinformatics applications just by setting a simple parameter: the size of the seed. Today, these families of algorithms are daily used by thousands of researchers. They represent a large part of the processing time of many bioinformatics centers. Their parallelization on

clusters, super-computers or grids has been one of the responses to increase the interactivity with end-users for rapidly processing huge masses of genomic data.

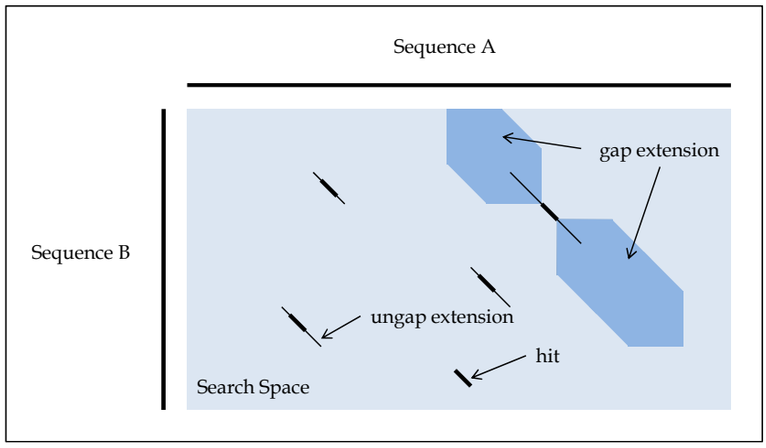

Fig. 4. 3-step BLAST strategy to detect similarity: (1) hit location; (2) ungap extension; (3) gap extension.

However, this type of parallelization is not the only issue. A lot of research works have been done to parallelize the genomic sequence algorithms on other hardware platforms. The next three sections present three different alternatives which exploit the fine-grained potential parallelism of the algorithms

4. VLSI and FPGA accelerators

Historically, the hardware acceleration of the string comparison problem is related to the parallelization of the dynamic programming algorithm on systolic arrays. The immediate implementation consists in hardwiring the recursion of equation (1) on a 2D systolic array as depicted Figure 5. According to the data dependencies, a cell D(i,j) receives data from its three top left neighbouring cells D(i-1,j-1), D(i,j-1), D(i-1,j), computes a similarity score and propagates it to its three bottom right cells D(i+1,j+1), D(i+1,j), D(i,j+1).
If the size of both sequences is n, then, due to the data dependencies, a similarity score is computed in $2n-1$ cycles, providing a speedup of $n^2/2n-1 \approx n/2$. The efficiency of this implementation is far from the optimum, since n^2 cells provide only a speedup of $n/2$. It can be noted that during the computation, only one anti diagonal of cells is active at each cycle. It is thus possible to emulate one column (or one line) on a single cell. The resulting architecture is a linear systolic array of n cells. Details of this kind of architectures can be found in (Lavenier & Giraud, 2005). In that configuration, the number of cycles to compute a similarity score between two sequences of size n stays the same, but the efficiency is much better: a speedup of $n/2$ is obtained with n cells.
To compare one sequence of size n with P sequences of size m with an n-cell array, $n+P\times m-1$ cycles are required. The speedup is thus given by:

$$\frac{n \times P \times m}{n + P \times m - 1} \approx n \qquad with\ P \times m \gg n$$

In that case, a speedup of *n* is obtained with a systolic array of *n* cells. This optimal situation occurs, for example, in phylogeny studies where thousands of sequences must be compared together. The systolic array is initialized with one sequence and all the other sequences pass sequentially through the array. This operation is iterated for all sequences.

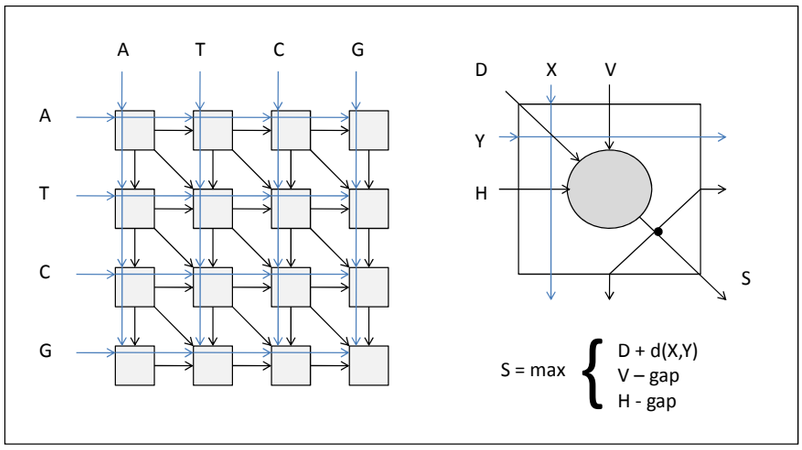

Fig. 5. Implementation of the programming dynamic algorithm on a 2D systolic array. Each cell performs a maximum of three terms. The similarity score is obtained on the bottom right cell in *2n-1* cycles (*n* is the length of the sequences).

Many systolic implementations have been studied and prototypes have demonstrated the efficiency of the systolic approach. Historically, dynamic programming algorithms were first accelerated with ASIC solutions, such as P-NAC (Lopresti, 1987), BioSCAN (White et al., 1991), Kestrel (Dashe et al., 1997), Samba (Guerdoux & Lavenier, 1997) or Swasad (Han & Parameswaran, 2002) accelerators. The performances of these parallel machines were impressive due to the high number of small processing units running in parallel. However, they suffered from:

- The high cost induced by the design of specific chips and the relatively small market niche where these accelerators were intended.
- The competition with software enhancements, such as seed heuristics, making them not so interested in terms of speed for a wide range of bioinformatics applications.

With the fast evolution of the FPGA technology, the successors of these machines naturally moved to reconfigurable hardware. Basically, their parallel structures didn't change but they could adapt their configuration according to the nature of the data to process (DNA, protein), or according to the type of alignments required by the applications (global alignment, local alignment, with gap, without gap, etc). Pioneer works were realized on the Splash and Splash-2 FPGA systolic machines in the beginning of the 90's (Hoang, 1993). Since this date, a lot of variants have been published in the literature, making this specific domain extremely active to product efficient reconfigurable accelerators (Yamaguchi et al., 2002) (Puttegowda et al., 2003) (Yu et al., 2003) (Dydel et al., 2004) (Pfeiffer et al., 2005) (Li et al., 2007).

It is also interesting to note that commercial products based on these parallel architectures are now available. For example, the DeCypher engine from TimeLogic[1] or the Cube from CLCbio[2] are two FPGA accelerators dedicated to bioinformatics applications, and especially tailored for genomic sequence comparisons. Other generic systems, like the SGI RASC-100 reconfigurable platform, for example, are not specifically devoted to this domain, but permit to implement extremely fast systolic operators (Nguyen et al., 2009).

5. SIMD instructions

The use of SIMD instructions available in each microprocessor for video and image processing purpose is also a very interesting way to parallelize genomic sequence comparison, and especially the dynamic programming algorithm. It can be efficiently speedup by considering groups of cells which can be computed concurrently on the 2D matrix. As stated earlier, the propagation of the computation follows the anti diagonal of the matrix. Cells belonging to a same anti diagonal can thus be processed independently. This can be done with SIMD instructions able to perform K instructions in parallel, as shown figure 6.

A first implementation of the Smith & Waterman algorithm was proposed by Woznia in 1997 (Wozniac, 1997) with the Visual Instruction Set (VIS) available on the SUN ULTRA SPARC processor. It follows the parallel scheme of figure 6. VIS instructions are executed in a specially enhanced floating point unit (FPU) and use its 64-bit registers. Instructions operate on two 32-bit, or four 16-bit integer data packed in a 64-bit double word. In this pioneer implementation, four cells of the matrix are executed in parallel by VIS instructions, storing the running score on 16-bit integers. A speedup of two was obtained.

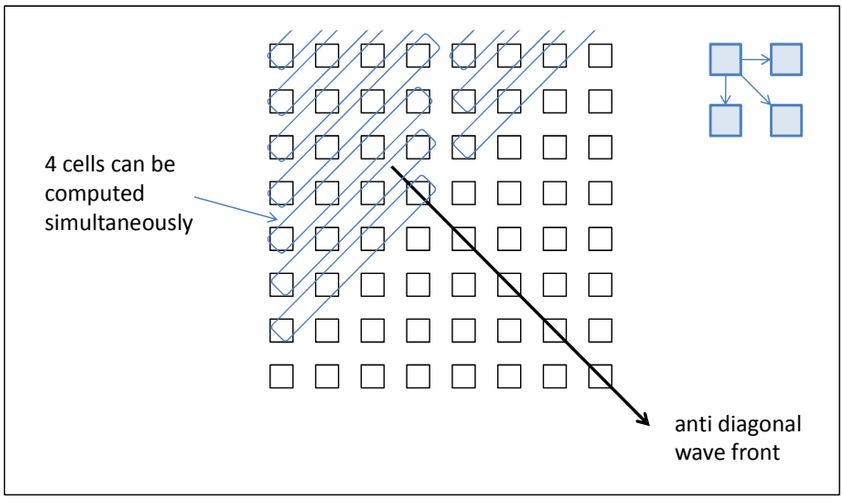

Fig. 6. Diagonal parallelization. Due to the data dependencies of the dynamic programming algorithm, only cells belonging to the same anti diagonal can be simultaneously processed. SIMD instructions can process K cells in parallel.

[1] www.timelogic.com
[2] www.clcbio.com

In (Rognes & Seeberg, 2000). the SSEARCH program (a quasi standard implementation of Smith-Waterman for comparing one query with many sequences from a database) was parallelized using the Intel SSE instructions (Streaming SIMD Extension). Eight cells are processed in parallel, each of them manipulating only 8-bit integer values. To increase the precision, unsigned integers are used and a bias mechanism is added to avoid negative values coming from the matrix substitution costs. Speedup of 6 is measured compared to the purely sequential version of SSEARCH.

The speedup improvement, compared to the Wozniac implementation, is due to (1) the superior number of cells computed in parallel, (2) to a clever preprocessing of the query consisting in building a structure called a *profile* and (3) to a programming optimization allowing the cells to be processed in a vertical way as shown figure 7.

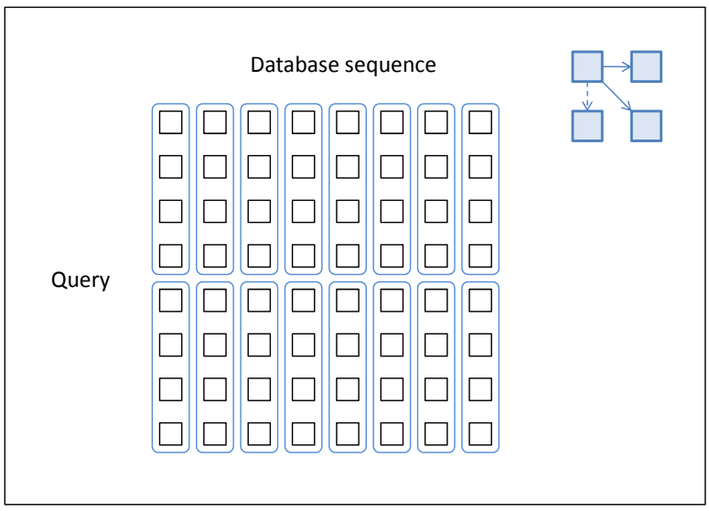

Fig. 7. Vertical parallelization. Under certain assumptions, the horizontal or vertical dependencies can be temporary omitted, leading to the possibility to compute several horizontal or vertical cells in parallel. Here, the vertical dependency is suppressed.

The optimization of the Smith & Waterman algorithm implemented in the Rognes & Seeberg version is based on the observation that in equation (3) V and H are often close to zero and, hence, most of the time, do not participate to the calculation of D. If for K consecutive vertical cells, the V values do not exceed a threshold value, then the vertical dependency can be suppressed, saving many computations. It is possible to check simultaneously if any of the K cells are above a threshold value. If so, the computation of the D values can be very fast. If not, the K scores are computed sequentially.
Farrar (Farrar, 2007) goes one step further by striping the query sequence into T fragments where $T = n/K$ (n is the length of the query and K the size of the SIMD vector). As in the previous implementation, the V values are also neglected to reduce data dependencies. The combination of these two techniques provides better data accesses to the SSE registers and greatly optimizes the SIMD parallelization. After the full computation of the 2D matrix, a lazy evaluation of V is done. Depending of the D scores in some points of the matrix, V values are updated and D scores are recalculated accordingly. This method is very efficient

for sequences with a low level of similarity. The D scores remain low and a very small fraction of the matrix needs to be updated. This situation typically happens in the case of database scanning where only a few sequences have significant similarity among millions of others. Speedup between 2 to 8 is reported compared to the previous SIMD implementations. Performance variations come from the fact that the Rognes & Seeberg implementation is very sensitive to the gap and substitution costs while the Farrar's implementation remains stable.

The Farrar implementation has still been improved in the SWSP3 package (Szalkowski, 2008). Modifications of the code are minors but they significantly reduce the cache footprint especially when long sequences are processed. Furthermore, the lazy V evaluation loop was restructured by transforming it into two nested loops with specific index ranges to hint the compiler at execution counts.

Finally, successive software improvements of the Smith & Waterman algorithm and their clever implementations using SIMD instructions have drastically reduced the performance gap with the seed heuristic algorithms which cannot directly benefit from these SIMD optimizations due to their irregular nature. However, in PLAST, a parallel BLAST-like version for comparing two large databases, SIMD instructions are efficiently used to speedup the computation of the ungap step which represents an important fraction of the execution time (Nguyen and Lavenier, 2008). Identical hits of both databases are grouped together to construct two lists of short sequences. Each sequence of one list is then compared with all sequences of the other list. At this step, gaps are not allowed, easing the computation of the scores to bit fit onto SSE instructions manipulating 16 × 8-bit integers. The parallelization of this part of the algorithm with SSE instructions makes PLAST three to ten times faster than BLAST.

The next generation of microprocessors will increase the SIMD instructions capabilities. New instructions will be provided with larger SIMD registers. For instance, the new Intel set of SSE instructions, called AVX (Firasta et al., 2008), will extend the SIMD integer registers to 256 and/or 512 bits. The genomic sequence comparison will directly benefit from these future improvements.

6. Graphical Processing Units (GPU)

GPGPU stands for General-Purpose computation on Graphics Processing Units. Graphics Processing Units (GPUs) are high-performance many-core processors that can be used to accelerate a wide range of applications[3]. Bioinformatics applications and especially the genomic sequence comparison problem did not escape from deep investigations to evaluate the potential gain these low-cost hardware accelerators can offer.

The last generation of GPU houses hundred of small processing units than can be easily programmed with high-level language, such as CUDA proposed by NVIDIA[4] or OpenCL (Open Computing **L**anguage) which is the future standard proposed by the Khronos Group[5]. In such a language, the GPU is viewed as a compute device suitable for massive parallel data application. It can randomly access its own data memory and can run a very

[3] www.gpu.org
[4] www.nvidia.com
[5] www.khronos.org/opencl/

high numbers of tasks in parallel. These tasks, called threads, are grouped in blocks and perform the same algorithms in a SIMD mode. Threads of the same block share data through a complex memory hierarchy and can be synchronized through specific synchronization points.

Again, the dynamic programming algorithm is a good candidate to for GPU because of its high regularity. Different parallelization techniques have been tested. The first relies on the independence of the computation which can be performed on the anti diagonal of the matrix (cf. previous sections). In that case, a thread is assigned to the computation of one anti diagonal. If n is the length of the sequences to be compared, then there is the possibility to run simultaneously up to n threads performing the recursion of equation (3). This approach has been implemented in (Liu et al., 2007). Speedup from 3 to 10 have been measured compared to the SSEARCH program, depending of the length of the sequences. Long sequences favor the use of GPU accelerators.

The implementation of (Manavski & Valle, 2008) is quite different and targets the scan of databases. The genomic bank is first sorted by the length of the sequences. Then each thread is assigned with a complete comparison between the query and one sequence of the database. As the threads are executed in a SIMD mode, it is important to have the same volume of computation per thread. This is why the sequences are sorted: blocks of sequences of identical size are processed together. Blocks of 64 threads are executed simultaneously, leading to a speedup of 30 compared to SSEARCH (not optimized with SSE instructions). The same style of implementation is done in (Ligowski & Rudnicki, 2009), but with a more efficient use of the global memory bandwidth, providing still better performance.

Another GPU implementation, called CUDASW++, and based on the same parallelization scheme as described above, compares its own performance with one of best multithreaded heuristic implementation (BLAST). A standard Linux workstation (3 GHz dual core processor) equipped with the latest NVIDIA board (GTX 295) including two GPU chips provides much better performance: an average speedup of 10 was reported. In that configuration, the adjunction of a low-cost accelerator outperforms the best seed-based heuristic software while increasing the quality of the results.

In the GPU version of PLAST (cf. previous section), the ungap alignment step for detecting local similarity near the hits are deported on GPU. Two lists (List1 and List2) of short sequences are sent to the GPU in order to make an all-by-all comparison. The parallelization is an adaptation of the matrix multiplication algorithm proposed in the CUDA documentation (Cuda, 2007). Matrices of numbers are simply replaced by blocks of strings. More precisely, suppose that block B1[N1, L] and block B2[N2,L] correspond respectively to List1 and List2, with L the length of the sequences and N1 (N2) the number of sequences in List 1 (List2). A third block SC[N1,N2] stores the scores of all the computation between block B1 and block B2.

The global treatment is done by partitioning the computation into block of threads computing only a sub block of SC, called SC_{sub}. Each thread within the block processes one element of SC_{sub} dimensioned as a 16.x16 square matrix. This size has been chosen to optimize the memory accesses, allowing the GPU internal fast memory to store short sequences which can simultaneously be shared by 256 threads. At the end, the host processor gets back an N1xN2 matrix of scores from which significant ones need to be

extracted. Figure 8 illustrates the parallelization scheme of a 16 x 16 string comparison, corresponding to a sub bloc SC_{sub}.

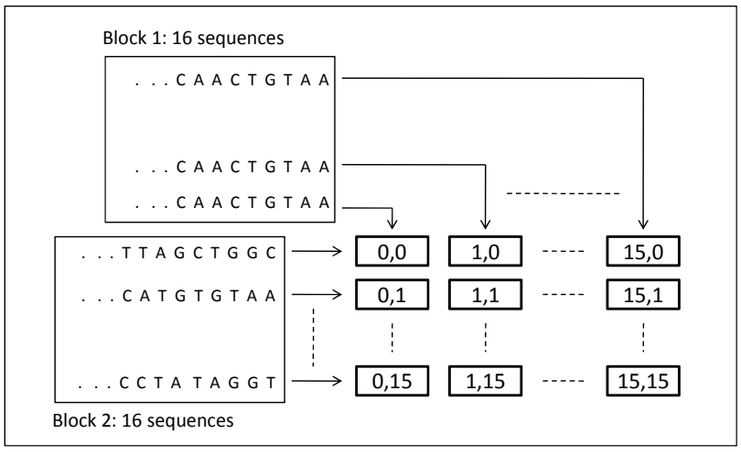

Fig. 8. Principle of the parallelization of an all-by-all string comparison on GPU. A thread (i,j) performs the comparison between the i^{th} and the j^{th} sequences.

Compared to an optimized sequential algorithm an average speedup of 10 is measured for performing this computation on recent NVIDIA graphic boards (GTX 280).

7. Conclusion

This chapter presented three approaches to parallelize the genomic sequence comparison problem: (1) systolic parallelization with VLSI or FPGA accelerators, (2) SIMD parallelization with microprocessor SSE instruction sets, and (3) streaming parallelization with GPU boards. These types of parallelization, referred as fine-grained parallelization, exploit the internal parallelism of the algorithms.

Another possibility is the data-level parallelism. This is actually the approach which is mostly exploited in many bioinformatics applications. A sequence, or a group of sequences, is generally compared with millions of other sequences. There is thus a natural way to split the computation on parallel machines, starting from multicores to clusters or grid platforms. The implementation is immediate: the database is dispatched among the available processing units, and each node works independently on its own subset of data. This approach is very efficient and fit well with the structures of the bioinformatics centres which are mainly composed of clusters of multiprocessors. Besides, MPI versions of the most popular bioinformatics software are now available.

These two alternatives, however, are not antagonist and can be combined to provide higher performance. A few nodes of a general purpose cluster can be equipped with hardware accelerators such as FPGA or GPU boards. When intensive comparisons are required, the system automatically assigns these nodes for this specific process, freeing the rest of the machines for other tasks. As a matter of fact, the scan of genomic databases may represent up to 60%-70% of the execution time of a bioinformatics server. As seen in this chapter, the heart of the algorithms mostly manipulates small integers and, consequently, exploits a relatively small fraction of the microprocessor computational power. Fitting these

algorithms into dedicated platforms is much more efficient both in terms of cost and electric power consumption.
With the next generation sequencing technology, the amounts of data to process become a real challenge. Comparing billions of genomic sequences is not the ultimate goal; it is just a necessary step before more complex data analysis in order to filter, organize or classify raw data coming from the fast sequencing machines. In order for this step to not become a serious bottleneck, comparison algorithms must exploit any forms of parallelism available in the next generation of microprocessors. The structures of the genomic sequence comparison algorithms probably need to be revisited to better fit tomorrow architectures such as manycores architectures enhanced with powerful SIMD instruction sets.

8. References

Altschul, S.; Gish, W.; Miller ,W.; Myers, E. & Lipman, D. (1990). Basic local alignment search tool, *J. Mol. Biol.,* vol. 215, no. 3, pp. 403–410

Altschul, S.; Madden, T.; Schäffer, A.; Zhang, J.; Zhang, Z.; Miller, W. & Lipman, D. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *Nucleic Acids Res*, vol. 25, pp. 3389-3402

Apweiler, R. et al. (2004). UniProt: the Universal Protein knowledgebase, *Nucleic Acids Res.*, vol. 32, database issue, pp. 115-119

Benson, D. et al. (2008). GenBank, *Nucleic Acids Res.*, vol. 36, database issue, pp. 25-30

Cuda. (2007). NVIDUA CUDA: Compute Unified Device Architecture, Programming guide, Version 1.0

Dahle, D.; Hirschberg, J.; Karplus, K.; Keller, H.; Rice, E.; Speck, D.; Williams, D. & Hughey, R. (1997). Kestrel: Design of an 8-bit SIMD Parallel Processor, *Proceedings of the 17th Conference on Advanced Research in VLSI (ARVLSI '97),* September 15 – 16, pp. 145-163, Ann Arbor, Michigan

Dydel, S. & Piotr, B. (2004). Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices**,** *14 th International conference on field-programmable logic and applications*, Antwerp , Belgique, pp. 23-32

Farrar, M. (2007). Striped Smith–Waterman speeds database searches six times over other SIMD implementations, *Bioinformatics*, vol. 23, no. 2, pp. 156-161

Firasta, N.; Buxton, M.; Jimbo, P.; Nasri, K. & Kuo, S. (2008). Intel AVX: New frontiers in performance improvements and Energy Efficiency. *Intel White paper*

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular biology*, vol. 162, no. 3, pp. 705-708

Gpu .(2009). http://gpgpu.org

Green, P. (1996). SWAT Optimization, www.phrap.org/phredphrap/swat.html

Guerdoux-Jamet, P. & Lavenier, D. (1997). SAMBA: hardware accelerator for biological sequence comparison, *Bioinformatics*, vol. 13, no. 6, pp. 609-615.

Han, T. & Parameswaran, S. (2002). Swasad: An Asic Design For High Speed Dna Sequence Matching, *Proceedings of the 2002 Conference on Asia South Pacific Design automation/VLSI Design*, January 07–11, Bangalore, India

Hoang, D. (1993). Searching genetic databases on SPLASH2, *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 185-191, Napa, California

Lavenier, D. & Giraud, M. (2005). Bioinformatcs Applications, in *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays,* Gokhale, M. & P.S. Graham P. editor, chapter 9, Springer, ISBN 0-387-26105-2

Liolios, K. et al. (2008). The Genomes On Line Database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata, *Nucl. Acids Res*., vol. 36, database issue, pp. 475-479

Li, IT.; Shum, W. & Truong, K. (2007). 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics*. vol. 8, no. 185

Ligowski, L. & Rudnicki, W. (2009). An efficient implementation of the Smith-Waterman algorithm on GPU using CUDA for massively parallel scanning of sequence databases, *HiComb 2009: Eighth IEEE International Workshop on High Performance Computational Biology*, Rome, Italy

Liu, Y.; Huang, W.; Johnson, j; & Vaidya, S. (2006). GPU Accelerated Smith-Waterman, *General Purpose Computation on Graphics Hardware (GPGPU): Methods, Algorithms and Applications, LNCS,* vol. 3994, pp. 188-195, ISSN 0302-9743

Liu, W.; Schmidt, B.; Voss, G.; Muller-Wittig, W., (2007). Streaming Algorithms for Biological Sequence Alignment on GPUs, *Parallel and Distributed Systems, IEEE Transactions on Parallel and Distributed Systems* , vol. 18, no. 9, pp. 1270-1281

Liu, Y.; Maskell, D. & Schmidt, B. (2009). CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units, *BMC Research Notes*, vol. 2 no. 73

Lopresti, D. (1987). P-NAC: A Systolic Array for Comparing Nucleic Acid Sequences. *Computer*, vol. 20, no. 7, pp. 98-99

Manavski A. & Valle, G. (2008). CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment, *BMC Bioinformatics*, vol. 9, no. 10.

Needleman , S. & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol Biol.,* vol. 48, no. 3, pp. 443–53

Nguyen, V. ; Cornu, A. & Lavenier, D. (2009). Implementing Protein Seed-Based Comparison Algorithm on the SGI RASC-100 platform, *16th Reconfigurable Architectures Workshop*, May 25-26, Rome, Italy

Nguyen, V. & Lavenier, D. (2008) Fine-grained parallelization of similarity search between protein sequences, *INRIA Report,* (RR-6513)

Pearson, W. & Lipman, D. (1988) Improved tools for biological sequence comparison. *Proc. National Academy of Science*, vol. 85, no. 8, pp. 2444–2448

Pfeiffer G,; Kreft H. & Schimmler, M. (2005) Hardware Enhanced Biosequence Alignment, *International Conference on METMBS'05*, Monte Carlo Resort, Las Vegas, Nevada, USA

Pop,M.; Salzberg, S. & Shumway, M. (2002). Genome Sequence Assembly:Algorithms and Issues, *Computer*, vol. 35, no. 7, pp. 47-54

Puttegowda, K.; Worek, W.; Pappas, N.; Dandapani, A.; Athanas, P. & Dickerman, A. (2003). A Run-Time Reconfigurable System for Gene-Sequence Searching, *Proceedings of the 16th international Conference on VLSI Design*, January 04 - 08, New Delhi, India

Rognes, T., Seeberg, E. (2000). Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors, *Bioinformatics*, vol. 16, no. 8, pp. 699–706

Shendure, J. & Hanlee, J. (2008). Next-generation DNA sequencing, *Nature Biotechnology*, vol. 26, no. 10, pp.1135-1145

Smith, T. & Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology,* vol. 147, no .1, pp. 195–197

Szalkowski A.; Ledergerber, C.; Krähenbühl, P. & Dessimoz C. (2008). SWP3 – fast multi-threaded vectorized Smith-Waterman for IBM Cell/BE and x86/SSE2, *BMC Research notes*, vol. 1, no. 107

White, C.; Singh, R.; Reintjes, P.; Lampe, J.; Erickson, B.; Dettloff, W.; Chi, V. & Altschul, S. (1991). BioSCAN: A VLSI-Based System for Biosequence Analysis, *IEEE International Conference on Computer Design: VLSI in Computer & Processors*, pp. 504-509, October 14 – 16, Cambridge, Massachusetts, USA

Wozniak, A. (1997). Using video-oriented instructions to speed up sequence comparison, *Comput Appl Biosci.,* vol.13, no. 2, pp. 145–50.

Yamaguchi, Y.; Marumaya, T. & Konagaya, A. (2002). High speed homology search with FPGAs, *Pacific Symposium on Biocomputing*, pp. 271–282, Lihue, Hawaii

Yu C.; Kwon K. ; Lee K. & Leong P. (2003). A Smith-Waterman systolic cell, *13 th International conference on field-programmable logic and applications,* Lisbon , Portugal

Parallel and Distributed Computing

Edited by Alberto Ros

ISBN 978-953-307-057-5

Hard cover, 290 pages

Publisher InTech

Published online 01, January, 2010

Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen