Design and Implementation of Parallel Algorithms for Gene-Finding*

James Puthukattukaran
Suresh Chalasani
Electrical & Comp. Engr. Dept.
Univ. of Wisconsin-Madison
Madison, WI 53706-1691

Periannan Senapathy
Genome International Corporation
579 D'Onofrio Drive
Madison, WI 53719

Abstract

Finding genes unequivocally in DNA sequences is one of the key goals of the Human Genome project. The human genome is a 3 billion character long DNA sequence and is estimated to contain about 100,000 genes. It has been shown by several biologists that genes in a DNA sequence satisfy certain special properties. In this paper, we use a combination of these properties to design a serial algorithm for gene-finding. To speed up the process of finding genes in long DNA sequences (of the order of $\geq$ 100,000 *characters), we design a parallel algorithm for gene-finding. We have implemented the parallel gene-finding algorithm on the CM-5 multicomputer as well as on a network of HP Apollo workstations under PVM. Experimental results indicate that our algorithms predict genes with reasonable accuracy.*

1 Introduction

The discovery of DNA and genes and their relationship to medicine has revolutionized genetics in the past few decades. One of the main goals of the Human Genome Project, which is a federally funded Grand Challenges Project, is to unequivocally find genes in the entire human genome; there are an estimated number of 100,000 genes in the 3-billion character long human DNA sequence [10, 1]. Finding genes is an important task, since it facilitates novel approaches to cure several diseases (for example, cancer) by correcting the defective genes.

Computational techniques have become increasingly important in genetics, especially to map and sequence the genomes of different organisms [15, 14]. Determining biologically significant patterns in raw DNA sequences will almost be an impossible task for a human being.

In this paper, we present parallel algorithms for finding genes in DNA sequences. Our algorithms make use of a few statistical rules that the DNA sequences follow. We also present experimental results from our implementation of these algorithms on a network of HP Apollo workstations running under PVM and compare it to an implementation on the CM-5 parallel computer.

The genome is often called the blueprint for the species. Roughly speaking, the genome is a concatenation of genes; each gene contains the plans for one or more proteins; and proteins are the building blocks of the body.

*This research has been supported in part by a grant from the Graduate School of UW-Madison and the NSF grants CCR-9308966 & ECS-9216308.

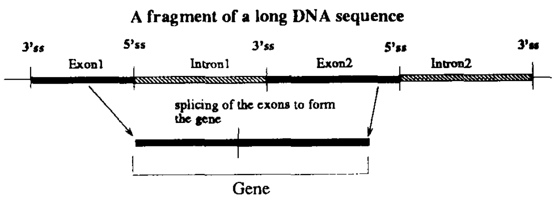
A fragment of a long DNA sequence
3'ss
5'ss
3'ss
5'ss
3'ss
Exon1
Intron1
Exon2
Intron2
splicing of the exons to form the gene
Gene

Figure 1: An overview of the DNA and gene.

A few definitions. The DNA sequence is made up of four nucleotides (or base-pairs or bases) called Adenine (A), Cytosine (C), Guanine (G), and Thyamine (T). Bases C and T are *pyrimidines*; A and G are *purines*. The DNA sequence for each organism can be very long; for example, in human beings, the DNA sequence is estimated to be three-billion nucleotides (or, equivalently, characters) long. The DNA has a double-helix structure and is double-stranded; further, one strand is the complement of the other [17]. The bases A and T on one strand match G and C on the complementary strand.

A *codon* is a three-character sequence of nucleotides A, C, T, and G. Thus $4^3 = 64$ different codons are possible out of which three specific codons — TAG, TGA, and TAA — are known as *stop-codons*. Each DNA sequence can be viewed as consisting of coding regions called *exons* and non-coding regions called *introns* (see Figure 1). Exons are the portions of DNA that are finally translated, according to some genetic code, into proteins. An intron just separates two consecutive exons. Finding genes in a given DNA sequence is equivalent to finding the exact location of exons (or the coding regions). The boundary between an exon and intron is known as the $5'$ splice site ($5'$-ss) and that between an intron and an exon is known as a $3'$ splice site ($3'$-ss) (see Figure 1).

Problem definition. The problem of finding genes in a given DNA sequence is equivalent to finding the exact location of exons in the sequence. Our approach to solve the gene-finding problem uses a combination of rules that exons in a DNA sequence seem to obey. Two important facets of a gene-finding algorithm are *speed* and *accuracy*. Sequential gene-finding algorithms can be slow when applied on DNA sequences that are a few hundred thousand characters long. To speed up the gene-finding algorithms, we design parallel algorithms and present implementation results on the CM-5 parallel computer as well as on a distributed system running under the PVM environment. The second important aspect of gene-finding is accuracy. Accuracy is often measured as

$$\frac{\#\text{ exons found by the algorithm}}{\text{Total }\#\text{ exons present in the DNA sequence}}.$$

Missing exons, which are the exons that cannot be found by the algorithm, are measured by accuracy. However, accuracy does not give an indication of the number of *false exons*; false exons are portions of introns that are incorrectly identified as exons by the algorithm.

Literature survey. Computational methods to identify splice sites ($5'$-ss and $3'$-ss) in DNA sequences were developed in the mid 1980s by Staden [19] and Senapathy [18]; they also suggested methods to use splice-site information in detecting genes. More recently, Lapedes *et al.* applied neural network techniques to the gene-finding problem [7]. Fields *et al.* have incorporated statistical sequence asymmetries to improve gene-finding algorithms [9]. Uberbacher and Mural have developed a neural-network based coding-recognition module (CRM) for recognizing genes [24]. However, none of the above methods are able to unequivocally identify eukaryotic genes [1]. For example, the CRM located 80% of coding exons of 100 or more bases. It also identified about 18% false exons. The problem is more complicated because approximately 30% of exons in genes are shorter than 100 nucleotides [23], which are more difficult to locate [24, 7]. The method of Uberbacher and Mural was successful only 30% of the time in identifying such short exons. Thus the overall accuracy of the best gene-finding algorithms is $70\% \times 80\% + 30\% \times 30\% = 65\%$. In addition, every gene-finding method reports several false exons.

Difference between our approach and existing work. Our approach, unlike existing techniques for gene-finding, combines a variety of features based on splice sites, branch sites, open reading frames, codon bias and RNY periodicity. The motivation to use a combination of various features is as follows: A false exon may satisfy one or a few individual features, where as only the genuine exons will have all the features. For instance, a false exon may have highly accurate splice sites and branch points; but it may not have a high codon bias and RNY periodicity. Implementation of parallel gene-finding algorithms on a distributed system and on the CM-5 is another significant contribution of this paper.

This paper is organized as follows. Section 2 will describe the various techniques used in finding genes in a DNA sequence. We will also discuss some of the implementation issues here. Section 3 discusses the results from our serial and parallel implementations of the gene-finding algorithms. Section 4 concludes this paper.

2 Gene Finding Concepts

The basic idea in finding exons (and thus finding genes) is as follows. The sequence of nucleotides in and around exons exhibit certain special characteristics. Any subsequence of the DNA sequence that exhibits these characteristics will be classified as an exon.

In this section, we will describe various rules used in identifying exon regions in a DNA sequence. We discuss these rules primarily from a computer science perspective, and do not elaborate on their biological significance. Figure 5 describes the sequence in which these rules are applied. One of the novelties of our method is the fact that we combine various techniques using different weights to get a high accuracy. The individual features and parameters of genes may occur randomly in a DNA sequence; however a combination of these features occur only in genuine exons and that is the basis for using a combination of techniques.

2.1 Splice sites

Exons can be identified if we find the 3' and 5' splice sites. To find exons, one should find the $3'$-ss and $5'$-ss for each exon. We use the method developed by Shapiro and Senapathy [18] and Senapathy *et al.* [4] to find the most likely splice sites in a given DNA sequence. We illustrate this method for 5' splice sites.

Table 1: Statistical Table for $5'$-ss for Primates

Location	A	C	G	T	CN
0	28	40	17	14	C
1	59	14	13	14	A
2	8	5	81	6	G
3	0	0	100	0	G
4	0	0	0	100	T
5	54	2	42	2	AG
6	74	8	11	8	A
7	5	6	85	4	G
8	16	18	21	45	T

Table 1 is a statistical table that was formed from experiments made on various genes of the primate species. A $5'$-ss sequence for the primates has a length of 9 characters. Row i of Table 1 gives the percentage of each base at location i of the $5'$-ss, for $0 \leq i \leq 8$. For example, character A appears with 28% frequency in position 0 of a $5'$-ss for primates. In row i, the final entry (under column CN) is the character that is most likely to be found in location i of a $5'$-ss. Thus, the final column of Table 1 forms the *consensus* sequence for a $5'$-ss (for primates). The following rule was used in arriving at the consensus sequence. If the highest percentage computed in row i for a particular character equals or exceeds 40, choose the corresponding character in that row; if there is more than one such character, there is a slightly more complex rule described in [4]. A similar table is constructed for the $3'$-ss except that the length is 15 characters rather than 9 characters used for $5'$-ss.

We use the statistical table to find the $5'$ splice sites in a given DNA sequence as follows. We take a window of 9 characters at a time and match it against the statistical table given above. Let $x_0 \ldots x_8$ be the current window of 9 characters. Let v_i be the value for character x_i in row i of the statistical table. Similarly, let h_i and l_i be the highest and the least values in row i of the statistical table. Define $total = \sum_{i=0}^{i=8} v_i$, $maxt = \sum_{i=0}^{i=8} h_i$, and $mint = \sum_{i=0}^{i=8} l_i$. In other words, total, $maxt$ and $mint$ represent the total score for the current window, maximum possible score for any window and the minimum possible score for any window, respectively. For example, $maxt$ in Table 1 equals $40+59+81+100+100+54+74+85+45 = 638$. Similarly, $mint$ equals 62. We obtain a score for the $5'$-ss in the current window using the following formula.

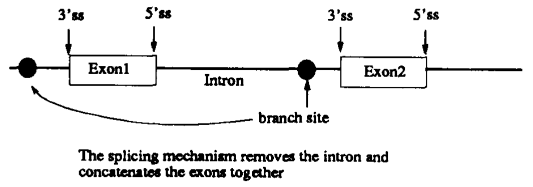
3'ss
5'ss
3'ss
5'ss
Exon1
Intron
Exon2
branch site
The splicing mechanism removes the intron and concatenates the exons together

Figure 2: The branchsite position for a DNA sequence.

$$score = 100 * \frac{(total - mint)}{(maxt - mint)}$$

For example, for a window AAGTGAGTA,

$$\text{total} = 28 + 59 + 81 + 0 + 0 + 54 + 11 + 4 + 16 = 253.$$

Thus the *score* for the window AAGTGAGTA is 43.9.

The above procedure is repeated and scores are computed for all the windows in the given DNA sequence. The windows that have the highest scores are considered good candidates for $5'$-ss. The computation of scores for $3'$-ss is slightly more complex and is discussed in [18, 4].

Once we find the $3'$-ss and $5'$-ss with highest scores, a $3'$-ss and its closest $5'$-ss form a *candidate exon*. Not all candidate exons are genuine exons. The next few methods described in this section help in differentiating genuine exons from false exons.

2.2 Branch Sites

Another signal sequence that helps the gene finding mechanism is called a *branch site* or *branch point signal*. The branch site lies within the intron usually between 30 and 50 characters before the $3'$-ss corresponding to an exon (see Figure 2). The biological function of a branch site is to signal the gene-finding mechanism on how to join the exons. Methods similar to those discussed above for the $5'$-ss also exist for computing the scores of branch sites. One such method was developed by Harris and Senapathy [11], which we use in our algorithms. Keller and Noon [13] also present a different method to identify branch sites.

After finding candidate exons in the given DNA sequence, we also find all the branch sites in the DNA sequence. For each candidate exon, we determine whether there is a branch site before its $3'$-ss; the presence of a branch sites before the $3'$-ss increases the probability of the candidate exon being a real exon.

2.3 Open Reading Frame (ORF) Technique

A sequence of characters can be an exon in any of the three *reading frames*. Reading frame 1 (or RF1) is the original sequence itself. Reading frames 2 and 3 are obtained from RF1 by removing the the first and the first two characters, respectively, from RF1 (see Figure 3).

A reading frame is said to be *open* if it does not contain any stop codons. For example, RF2 contains the stop codon TAA for the sequence ATCGTAATGTTACTA as shown in Figure 3; hence, RF2 is not an open reading frame, while other reading frames are open. We use the

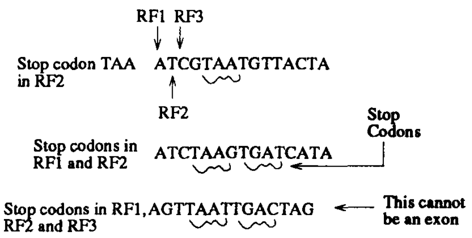

Figure 3: Example for the Open Reading Frame method.

following biological rule to prune the list of candidate exons : *A candidate exon can be a real exon only if at least one of its reading frames is open.*

In Figure 3, the third sequence has stop codons in all three reading frames. Hence, it cannot be an exon and will be removed from the list of candidate exons.

2.4 RNY Periodicity Technique

Recall that bases C and T are termed pyrimidines, and A and G are known as purines. Let us denote a purine with R and a pyrimidine with Y; further, let N denote any of the bases A, C, T, G. It has been statistically observed that the codons found in exons have a purine in the first position, any base in the second position, and a pyrimidine in the third position. That is, a codon found in an exon is most likely to have the form RNY, while a codon found in the intron regions will not have any special structure associated with them.

Thus, given a candidate exon, we count the fraction of codons of the RNY form in that exon as follows.

$$\text{RNY factor} = \frac{\#\text{ codons of the form RNY}}{\text{Total }\#\text{ codons in the candidate exon}}$$

The higher the RNY factor for a given candidate exon, the greater is the probability for it to be a real exon.

2.5 Codon Bias Technique

There are 64 codons out of which three are stop codons. Proteins, which are the final products of gene-expression, are made of 20 amino acids[1]. Any codon, other than a stop codon, can be an amino acid. Thus, there are 61 codons which are mapped into 20 amino acids. Since there are more codons than amino acids, multiple codons code for the same amino acid. However, the frequency with which a specific codon codes for an amino acid differs widely between exons and introns.

As an example, let us assume that codons $\mathcal{C}_i$, $\mathcal{C}_j$ and $\mathcal{C}_k$ code for amino-acid $\mathcal{A}_l$. Codon $\mathcal{C}_i$ is preferred by amino acid $\mathcal{A}_l$ over codons $\mathcal{C}_j$ and $\mathcal{C}_k$ in exons, while no such preference exists in introns. This phenomenon according to which exons tend to show bias to a specific codon to code a particular amino acid is known as *codon bias* (or *codon preference*).

Staden and McLachlan [20] and Staden [5] used the codon bias method to identify protein coding regions. We use a method similar to that discussed in [20] to distinguish exons from introns. Let

$$S = a_1b_1c_1a_2b_2c_2 \ldots\ldots a_nb_nc_na_{n+1}b_{n+1}c_{n+1}$$

[1]An amino acid is the basic building block for a protein.

be a given candidate exon sequence. Let f_{abc} be the frequency of the codon *abc*. Let q_i be the product of the frequencies of all codons in reading frame *i*. We compute q_i using the formulae given below.

$$\text{Frame 1:} \quad q_1 = f_{a_1b_1c_1} f_{a_2b_2c_2} \cdots f_{a_nb_nc_n} \quad (1)$$

$$\text{Frame 2:} \quad q_2 = f_{b_1c_1a_2} f_{b_2c_2a_3} \cdots f_{b_nc_na_{n+1}} \quad (2)$$

$$\text{Frame 3:} \quad q_3 = f_{c_1a_2b_2} f_{c_2a_3b_3} \cdots f_{c_na_{n+1}b_{n+1}} \quad (3)$$

Let p_i be the probability that frame *i* is the correct frame in which the given sequence S will be read by the gene-splicing process. We compute p_i using

$$p_i = q_i/(q_1 + q_2 + q_3), \qquad i = 1, 2, 3.$$

Next the codon bias value is computed using the equation

$$\text{codon-bias-value} = \max\{p_1, p_2, p_3\}. \quad (4)$$

We declare that the sequence S exhibits codon bias if the codon-bias-value exceeds a user-defined threshold.

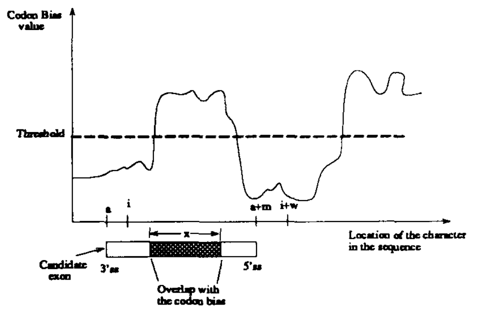
Codon Bias value
Threshold
a
i
a+m
i+w
x
Location of the character in the sequence
Candidate exon
3'ss
5'ss
Overlap with the codon bias

Figure 4: Forming the codon bias list and computing scores for the candidate exons.

Creating the codon bias list. Figure 4 shows the codon bias for a sample DNA sequence. The x-axis shows the locations of the characters in the DNA sequence and the y-axis shows the codon bias value at each location. We compute the codon bias value for a character in the DNA at location *i* by taking a window of $w+1$ characters from location *i* to location $i + w$ (see Figure 4) and using the Equation 4 in Section 2.5. The codon bias value is computed for all the characters in the DNA. Typical window sizes used are around 65 characters [20, 5].

Computing the codon bias score for each candidate exon. Once the codon bias list is constructed and the codon bias values for each location in the DNA obtained, we use this information to compute the codon bias scores for the exons in the candidate exon list. As shown in Figure 4, the shaded region of the candidate exon is said to have a codon bias. If the location of the $3'$-ss is a and that of the $5'$-ss is $a+m$, then the exon length is $m+1$. If the length of the codon bias region that overlaps with the exon is x (length of the shaded region), then the codon bias score s_{cb} of the candidate exon is

$$s_{cb} = \frac{x}{m+1}.$$

That is, s_{cb} is a quantitative measure of the portion of the candidate exon that has codon bias.

2.6 Uneven Positional Base Frequencies (UPBF) Method

The method that we describe in this section is attributed to Fickett [8]. Exons (or coding sequences) tend to show unequal use of the four bases in the three positions of codons. From biological experiments, it has been shown that this inequality in the usage of bases is more pronounced in coding than in non-coding regions.

To apply the UPBF method, let us consider the given DNA sequence S. From S, we calculate N_{ij}, the number of times base i occurs in position j of a codon, for each base i and each position j. Since there are four bases A, C, T and G and three positions in each codon, 12 different combinations exist for N_{ij}. We then calculate the expected value E_i for each base in each position of a codon using the following formula

$$E_i = (N_{i1} + N_{i2} + N_{i3})/3, \qquad i \in \{A, C, T, G\}$$

Now we measure the divergence D of the usage of bases in sequence S using the formula given below.

$$D = \sum_{i,j} |E_i - N_{ij}|, \qquad i \in \{A, C, T, G\}, \quad j \in \{1, 2, 3\} \quad (5)$$

That is, we measure the absolute differences between observed and expected positional base frequencies. The divergence D gives us a measure of the amount of variation there is in the usage of the four bases in the three locations of the codon. The higher this divergence, the greater the probability for a sequence to be an exon. The implementation of the UPBF list is done exactly in the same manner as the codon bias list [12].

2.7 Main Routine

Figure 5 gives a flowchart for the entire procedure. On each candidate exon, the techniques based on codon bias, branch sites, RNY periodicity and UPBF are applied and the corresponding scores are obtained. We can compute the overall score for each candidate exon using

$$s_o = w_{ss} * s_{ss} + w_{bs} * s_{bs} + w_{cb} * s_{cb} + w_{rny} * s_{rny} + w_{upbf} * s_{upbf}.$$

where $s_o, s_{ss}, s_{bs}, s_{cb}, s_{rny}, s_{upbf}$ are the overall, splice site, branch site, codon bias, RNY and UPBF scores respectively; $w_o, w_{ss}, w_{bs}, w_{cb}, w_{rny}, w_{upbf}$ are the weights for overall, splice site, branch site, codon bias, RNY and UPBF tecniques respectively. We then sort the candidate exons based on their overall scores and select the top few exons as the exons found by the algorithm. A discussion on how to choose weights for individual methods will be given in Section 3.

Analysis of the serial algorithm. Let n be the length of the DNA sequence. The overall complexity is dominated by the step in which splice sites and branch sites are found; this step involves sorting of splice sites and branch sites based on their scores and thus, requires $O(n \log n)$ time. Each of the remaining methods requires scanning the input DNA sequence at most once, and hence can be completed in $O(n)$ time.

2.8 Parallel Implementation

The basic outline of the parallel implementation is given below.

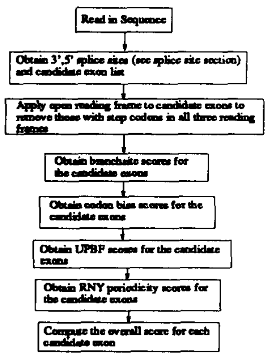
Read in Sequence
Obtain 3',5' splice sites (see splice site section) and candidate exon list
Apply open reading frame to candidate exons to remove those with stop codons in all three reading frames
Obtain branchsite scores for the candidate exons
Obtain codon bias scores for the candidate exons
Obtain UPBF scores for the candidate exons
Obtain RNY periodicity scores for the candidate exons
Compute the overall score for each candidate exon

Figure 5: The flowchart explaining the serial algorithm for gene finding.

- Divide the DNA sequence among the processors.
- Each processor then creates a candidate exon list and is responsible for the $3'$-ss location that lies in the DNA subsequence assigned to it.
- Each processor applies the steps in Figure 5 on its list of candidate exons.

The only difference between the parallel and serial versions arises from the communication requirements of the parallel version. The step in which splice sites are found dominates the computation as well as the communication time of the parallel algorithm. Hence, we describe only the communication complexity in finding splice sites.

2.8.1 Splice sites

To get the splice sites with the highest scores, we need to first sort the splice sites in each processor by their scores. The sorting takes $O(\frac{n}{p}\log\frac{n}{p})$. The number of best candidates required is specified by *LIMIT*. The best *LIMIT* candidates from each processor are merged together into one processor. This merging step takes $O(\log p * LIMIT)$. This array of the best candidates is then broadcasted to all the processors in $O(p * LIMIT)$ time.

Once every processor has computed its candidate exon list, it is possible that, for a given $3'$-ss region, the matching $5'$-ss lies on another processor. The communication pattern used in obtaining the missing part of the candidate exon from other processors forms an *acyclic* graph. This is because each processor needs to send sequences to only those processors whose number is lower than its number. The first processor only receives subsequences while the final processor only sends. The number of messages sent under this communication pattern is $O(p^2)$, since, in the worst case, each processor sends to every processor whose number is lower than itself. Computation in the parallel algorithm is dominated by the term $O(\frac{n}{p}\log\frac{n}{p})$.

3 Implementation Results

In this section, we first give a brief summary of the Thinking Machines CM-5 and the PVM software running on a network of 7 HP Apollo Workstations. We then give results on the performance of the parallel implementations of our algorithm on the Thinking Machines' CM-5 using the CMMD version 3.2 message passing library and the PVM distributed system.

3.1 CM-5 Summary

The CM-5 is the latest massively parallel computer (MPP) developed by Thinking Machines Corporation. The CM-5 is capable of offering a peak performance of up to 1 teraflops. It uses the Single Program Multiple Data (SPMD) programming model. It has three networks: a data network, a control network and a diagnostic network. The data network provides high performance point-to-point data communication between processors and has a *fat-tree* structure [16]. The control network provides cooperative operations, including broadcast, synchronization and scans.

The CM-5 uses CMMD as its message-passing library [21, 22]. The current version is CMMD 3.2 and runs under the CMOST operating system. The current version of the operating system is CMOST Version 7.2 beta 1.1-P4.

3.2 PVM Summary

PVM (Parallel Virtual Machine) is a software package developed at the Oak Ridge National Laboratory [6]; it allows the use of a set of heterogenous computers as a single computational resource (just like a MIMD parallel computer). The computers networked together could be workstations, vector machines, and multiprocessors interconnected by one or more networks. We have PVM installed on 7 HP Apollo Workstations in the Computer-Aided Engineering Laboratory of UW-Madison.

The PVM consists of two parts: the *pvmd* daemon and the library of PVM routines. Application programs must be linked with the library to use PVM. The PVM libraries contain all system calls for communication and synchronization. The programming model used by our program is the SPMD paradigm (Single Program Multiple Data), which is similar to that used on the CM-5.

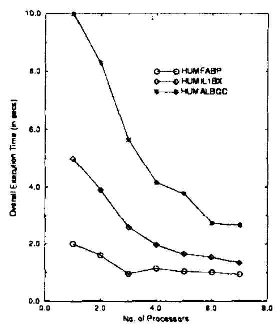

Figure 6: Overall execution time versus number of processors for small sequences on PVM.

3.3 Results from Parallel Implementations

Table 2: The Description of the Benchmark Sequences Used (Human Genes)

Sequence Name	Description	No. of base pairs
HUMFABP	Intestinal binding acid protein gene	5204
HUMIL1BX	Interleukin 1 beta gene	9721
HUMALBGC	Serum albumin gene	19,002
HUMAFP	Aalpha-fetoprotein gene	22,166
HUMNEUROF	Oligodendrocyte myelin glycoprotein gene	100,849

Table 2 gives a brief listing of all the sequences that we used in our experiments. These sequences were obtained from the Genbank, which is a public domain software that contains a database of all the known genes and their locations in the DNA [3]. Each sequence in the database gives information about the locations of the known exons and introns, the number of base pairs and other relevant information. The exons in these sequences were known; this fact helped us in measuring the degree of accuracy with which our implementations find genes. We took several DNA sequences whose length varied from 5000 to above 100,000 characters.

Accuracy of gene-finding for benchmark sequences. Table 3 reports the accuracy of our gene finding algorithm. It should be noted that the parallel version implemented on both the PVM and the CM-5 give the same accuracy because they use the same algorithms. As seen from the table, most of the sequences we used show an accuracy of close to 65%. This accuracy is better than the accuracy that we obtained if only a particular method was used (instead of a combination of various methods) to find exons. The HUMNEUROF sequence is one of the longest sequences that we ran and since there are only 23 exons in such a long sequence makes it difficult to get high accuracies. We observed that, for those exons that had not matched in the HUMNEUROF sequence, most of them had either the $3'$-ss matched to the wrong $5'$-ss or vice-versa. Ultimately, the accuracy of gene-finding can be improved only if new biological rules related to exons are discovered.

Table 3: Exons Found by Our Algorithm for a Few Benchmark DNA Sequences

Sequence Name	No. of exons	Exons found	Accuracy (in percent)
HUMFABP	4	2	50%
HUMIL1BX	6	4	67%
HUMALBGC	14	9	64%
HUMAFP	14	9	64%
HUMNEUROF	23	10	43%

Choosing Weights. The accuracies reported in Table 3 were for a specific combination of weights. The set of weights we chose for the sequences given in Table 3 were those that gave us the highest accuracy.

However, we found that modifying the weights could lead to the uncovering of new exons at the expense of masking some other exons found with the old set of weights. The accuracy can vary widely for different weight combinations. For example, for the sequence HUMALGBC, a set of weights $w_{ss} = 0.6, w_{cb} = 0.2, w_{bs} = 0.1, w_{rny} = 0.1$ gives an accuracy of 57%; for the same sequence, exons are found with an accuracy of 64% with weights $w_{ss} = 0.5, w_{cb} = 0.2$, $w_{bs} = 0.2$, and $w_{rny} = 0.1$. At this time, an optimal combination of weights which yields best accuracies can only be found by thorough experimentation.

Comparison of implementations on PVM & CM-5. In our implementation, we chose to run DNA sequences of different lengths for processor sizes ranging from 1 to 7 on the CM-5. We ran similar experiments on the distributed system of workstations running under PVM by varying the number of workstations networked together (from 1 to 7 as well). We however did run the gene-finding program for upto 32 processors on the CM-5 in order to see the performance of the algorithm for larger number of processors. We were limited to a set of seven workstations that were available to run PVM.

Figures 6 and 7 show the overall execution times for short and long sequences that were run on PVM. The PVM measurements were taken when we had exclusive access to the entire network of workstations (no other users were on the system). We ran the same experiments on the CM-5 and got graphs with similar shapes (except different execution times as discussed later).

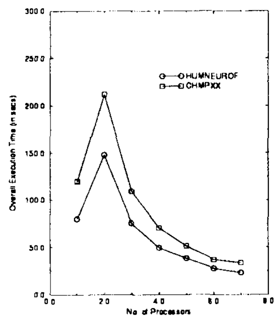

Figure 7: Overall execution time versus number of processors for large sequences on PVM.

In Figure 7, we see that when going from 1-processor to a 2-processor case, there is an increase in the overall execution time. This is because there is absolutely no communication overhead in the serial case, while in the 2-processor case the communication overhead is high due to large data sizes. However, as we use more than 2 processors, the execution time starts to decrease because the message sizes start decreasing drastically. On the CM-5, the communication cost is significantly higher than the computation cost on the CM-5 [2]. The decrease in the message sizes with larger number of processors makes computation dominate communication, and this results in lower execution times as the number of processors increases. Similar behavior is seen when we run the same algorithm on the CM-5. This trend is not observed for smaller sequences as shown in Figure 6. We observed that the overall execution times on PVM are lower than those obtained from the CM-5; the reasons are given below.

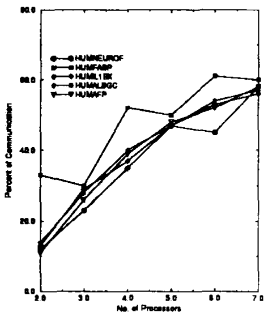

Figure 8: Percent of time spent in communication for DNA sequences run on PVM.

Comparison of computation time between PVM and CM-5. Figure 9 gives a comparison of the time spent only in computation on the CM-5 and the PVM. This comparison is done for the sequence HUMALBGC. As mentioned earlier, the PVM gives lower execution times than CM-5. We ran a few experiments to compare the processing power of a CM-5 node to that of an HP Apollo workstation. We found that the HP workstation was about *five times faster* than a single CM-5 node; hence the reason for the lower execution times on PVM than that on the CM-5. In Figure 9, we notice that, as we increase processors, this gap between the computation times starts decreasing. This is because the amount of computation per node starts to decrease as the number of processors increases, thus reducing the effect of computation on the overall execution time.

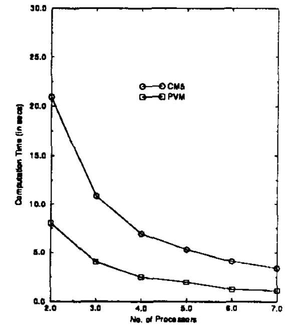

Figure 9: Comparison of computation times for the CM-5 and PVM for the sequence HUMALBGC (19002 characters).

Comparison of communication time between PVM and CM-5. Figure 8 shows the percent of time spent in communication by the algorithm when run on PVM. We obtained this as follows:

$$\text{Percent of Communication} = \frac{\text{Communication Time}}{\text{Overall Execution Time}}$$

As expected, for a smaller number of workstations we see that computation dominates and thus a smaller amount of time is spent in communication. For a larger number of workstations, the number of messages exchanged between workstations increases and, as a result, we see an increase in the percent of time spent in communication. As we go to 7 processors, we see that PVM spends around 50% of its time in communication. Compared to PVM, the percentage of time spent in communication for the CM-5 is negligible.

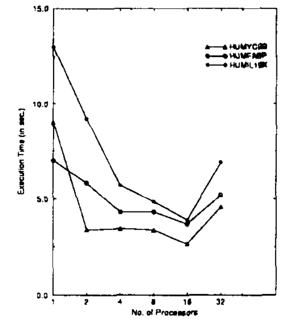

Figure 10: Overall execution time versus number of processors for small sequences for 32 nodes on CM-5.

Performance with > 7 processors on the CM-5. Figures 10 and 11 show the overall execution times obtained by the parallel version of the program when run on the CM-5 as the number of processors varies from 1 to 32. For short sequences, with more than 16 processors, the overall execution time increases (see Figure 10). This is due to the fact that there is not enough computation to mask the overhead incurred by communication when we use a larger number of processors. However, if we have larger data sizes, this trend is not observed and lower execution times are achieved even when the number of processors exceeds 16 (see Figure 11).

For large data sizes and for more processors, there is a dramatic reduction in the overall execution time. With today's technology, DNA sequences are produced at an extremely high rate in short periods of time. Therefore, long DNA sequences (of the order of 10^6 characters) are more likely to be sequenced in laboratories than short DNA sequences [15]. For large DNA sequences, Figure 11 exhibits low overall execution times. Thus, Figure 10 does not represent the common case. The human genome which is about 3 billion characters will be completely sequenced rather soon and finding all the genes in this entire genome is definitely a formidable task for a supercomputer or for a network of workstations.

4 Conclusions

Parallel computers are required to solve problems in computational genetics because of the following reasons:

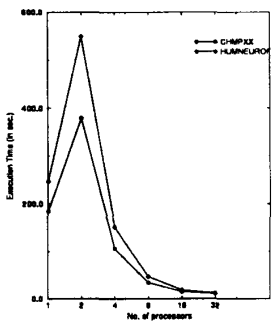

Figure 11: Overall execution time versus number of processors for large sequences for 32 nodes on CM-5.

- Data sizes in the DNA sequences are very large, and
- The rate at which data is being sequenced is high thus leading to a rapid influx of raw DNA sequences.

In this paper, we have developed a parallel algorithm for finding genes in DNA sequences. This algorithm uses of a combination of biological rules that the exons in the DNA sequence obey statistically. Various investigators have used individual techniques such as splice site, codon bias, branch site and RNY periodicity to find exons. In this paper, we used a combination of these various techniques to get accuracies better than those that can be obtained by one technique alone. We have experimented with different weights given to these techniques and have found that different combinations of these weights give rise to different accuracies in gene-finding. We achieve accuracies in the range of 65% for sequences of varying lengths. Ultimately, the accuracy can be improved only if new biological rules about exons are discovered. Our implementation results show that the performance of the parallel gene-finding algorithms implemented on a distributed system using PVM is comparable to that implemented on the CM-5 supercomputer. We are investigating the possible use of more biological rules to improve the accuracy of gene-finding.

References

[1] *Cited in the Special Issue on* "The Human Genome Project". *Los Alamos Science*, 20:162–163, 1992.

[2] G. Bell. Ultracomputers: A teraflop before its time. *Comm. of the ACM*, 35(8):27–47, Aug. 1992.

[3] M. L. Cinkosky, J. W. Fickett, P. Gilna, and C. Burks. Electronic data publishing and genbank. *Science*, 252:1273–1277, May 1991.

[4] R. F. Doolittle, editor. *Methods in Enzymology*, volume 183, pages 252–278. Academic Press, 1990.

[5] R. F. Doolittle, editor. *Methods in Enzymology*, volume 183, pages 163–180. Academic Press, 1990.

[6] J. D. et al. *A Users's Guide to PVM Parallel Virtual Machine.* Oak Ridge Nat'l Lab, July 1991.

[7] R. Farber, A. Lapedes, and K. Sirotkin. Determination of eukaryotic protein coding regions using neural networks and information theory. *J. Mol. Biol.*, 226:5305–5318, 1992.

[8] J. W. Fickett. Recognition of protein coding regions in dna sequences. *Nucleic Acids Research*, 10(17):5303–5318, 1982.

[9] C. A. Fields and C. A. Soderlund. gm : A practical tool for automating DNA sequence analysis. *Comput. Appl. Biosci.*, 6:263–270, 1990.

[10] K. A. Frenkel. The human genome project and informatics. *Communications of the ACM*, 34(11):41–51, November 1991.

[11] N. Harris and P. Senapathy. Distribution and consensus of branch point signals in eukaryotic genes : A computerized statistical analysis. *Nucleic Acids Research*, 18:3015–3019, 1990.

[12] S. Chalasani, J. Puthukattukaran and P. Senapathy. Design and implementation of parallel algorithms for gene-finding. Tech. Report ECE 94-7, UW-Madison, 1994.

[13] E. B. Keller and W. A. Noon. *Proc. Natl. Acad. Sci.*, 81:7417–7420, 1984.

[14] E. S. Lander et. al. Computing in molecular biology: mapping and interpreting biological information. *IEEE Computer*, 24(11):6–13, 1991.

[15] E. S. Lander et. al. Mapping and interpreting biological information. *Communications of the ACM*, 34(11):33–39, Nov 1991.

[16] C. Leiserson. Fat trees: Universal network for hardware-efficient super computing. *IEEE Trans. on Computers*, C-34(10):892–901, Oct 1985.

[17] J. A. Peters. *Classic Papers in Genetics.* Prentice Hall Inc., New Jersey, 1964.

[18] P. Senapathy and M. B. Shapiro. RNA splice junctions of different classes of eukaryotes : Sequence statistics and functional implications in gene-expression. *Nucleic Acids Research*, 15:7155–7175, 1987.

[19] R. Staden. Measurements of the effects that coding for a protein has on a dna sequence and their use for finding genes. *Nucleic Acids Research*, 12(1):551–567, 1984.

[20] R. Staden and S. McLachlan. Codon preference and its use in identifying protein coding region in the dna sequences. *Nucleic Acids Research*, 10(1):141–156, 1982.

[21] Thinking Machines Corporation, Cambridge, Massachusetts. *The CMMD Reference Manual*, 2.0 beta edition, August 1992.

[22] Thinking Machines Corporation, Cambridge, Massachusetts. *The CMMD User's Manual*, 1.1 edition, January 1992.

[23] T. W. Traut. *Proc. Natl. Acad. Sci.*, 85:2944–2948, 1988.

[24] E. C. Uberbacher and R. J. Mural. *Proc. Natl. Acad. Sci.*, 88:11261–11265, 1991.