

DNA Strand Separation Prediction: A Parallel Implementation

Gabriela Hristescu
Department of Computer Science
Rutgers University
Piscataway, NJ 08855, U.S.A.
hristesc@cs.rutgers.edu

Craig J. Benham *
Biomathematical Sciences Department
Mount Sinai School of Medicine
New York, NY 10029, U.S.A.
benham@msvax.mssm.edu

Martin Farach †
Department of Computer Science
Rutgers University
Piscataway, NJ 08855, U.S.A.
farach@cs.rutgers.edu

Abstract *The strand separation problem is that of calculating the probability of separation of each base pair in the DNA sequence under specified physical and chemical conditions. A computational method that performs an approximate statistical mechanical analysis of this phenomenon has previously been developed and shown to be accurate when compared with experiments.*

In this paper we propose a parallelization for the DNA strand separation prediction problem and evaluate the performance of two parallel implementations on a shared memory multiprocessor, one using static and the other dynamic scheduling. We show that DNA strand separation can be efficiently parallelized using a loop-based decomposition. For the DNA molecule of about 4K bases we chose as input, static scheduling performs well on up to 8 processors, but causes substantial load imbalance on 16 processors. Dynamic scheduling delivers a very good speedup in all considered cases.

Keywords: DNA Strand Separation, Parallel Applications, Dynamic Scheduling

*Supported by grants RO1-GM47012 (NIH) and BIR 93-10252 (NSF).

†Supported by NSF Career Development Award CCR-9501942, an Alfred P. Sloan Research Fellowship and NATO Grant 960215.

1 Introduction

The massive DNA sequencing effort known as the Human Genome Project has focused wide attention on the need for accurate computational methods to search DNA sequences for gene coding and regulatory regions. To date most of these efforts have treated DNA as merely a sequence from a four letter alphabet and have looked for subsequences which are correlated with function activity. However, recent developments have demonstrated the need for new computational models based on the structural, physical and mechanical properties of DNA. Many important DNA activities, including its essential functions of gene expression and replication, require that the two strands of the DNA double helix separate in certain regions to provide access to the individual bases that encode the genetic information. So the stringent control of DNA strand separation has emerged as an essential factor in our understanding of the regulation of DNA function.

The strand separation problem is that of calculating the probability of separation of each base pair in the DNA sequence under specified physical and chemical conditions. Exact enumerative methods are computationally expensive, so their use is limited to sequences

that are too short to be biologically interesting. A computational method, which we call Benham’s algorithm, that performs an approximate statistical mechanical analysis of this phenomenon has previously been developed and shown to be accurate when compared with experiments [1, 2]. While this method is much faster than the fastest known exact method, for the 4K base long circular DNA molecule we chose as input, this algorithm takes about 3 days to run on a uniprocessor. The challenge is to improve the execution time so that the analysis can be extended to the largest possible DNA sequences.

In this paper we propose a parallelization of this method for calculating DNA strand separation probabilities. We show this to be an *embarrassingly parallel problem* up to a certain partitioning granularity. Although the number of tasks produced by this decomposition is sufficiently large for medium-size parallel systems, task length is extremely variable, and achieving an efficient parallelization requires workload balancing. We evaluate the performance of two parallel implementations on a 16-processor Silicon Graphics Challenge system, one using static scheduling and the other using dynamic scheduling. Static scheduling performs well for up to 8 processors, but on 16 processors it suffers from a substantial load imbalance, which flattens the speedup. On the other hand, with dynamic scheduling the algorithm performs well even on 16 processors.

The remainder of the paper is organized as follows: In Section 2 we briefly sketch the biological and theoretical background relating to the DNA strand separation problem. Section 3 outlines the sequential algorithm. In Section 4 we describe the parallelization of Benham’s method. Section 5 presents the performance results from static and dynamic scheduling implementations on the SGI Challenge. In Section 6 we discuss issues related to the input sequences and in Section 7 we present the conclusions.

2 Background

Each strand of a DNA molecule is a linear chain comprised of repeating units of sugar and phosphate groups. Attached to each unit is one of four possible bases, abbreviated as A, G, C and T. In 1953 Watson and Crick proposed that a DNA molecule consists of two strands entwined in a double helix. In this helical duplex structure, an A on one strand always occurs opposite a T on the other and C is similarly always paired with G. The genetic endowment of an organism is encoded in the sequence of bases of its genomic DNA. This endowment consists of the genes, which encode the construction of proteins, together with a variety of (usually) non-coding regulatory regions. The latter control essential physiological activities of the DNA such as the occasions and rates of expression of individual genes, initiation of DNA replication and other processes.

Because the mechanisms of both gene expression and replication require access to the individual bases within a pair, strand separation is a mandatory step in the initiation of these processes. Energy is required to drive local separation of the strands of the double helix.

The double helix of DNA naturally twists around an axis, undergoing one full rotation per 10.4 bases. DNA can store energy by being over- or underwound, much as a spring can. We call any deviation from the baseline rate of turns per bases *superhelicity*, and define the superhelical parameter α to be the number of turns by which the DNA is over or undertwisted compared to its relaxed state, when the conformation of the molecular central axis is held fixed. The energy of superhelicity can be used to drive the local separation processes needed to access individual bases. The DNA within organisms commonly occurs in a negatively superhelical, undertwisted state. If this negative superhelicity is sufficient, it can drive local separations of the DNA duplex at the most susceptible sites. Modulation of DNA superhelicity has been shown experimentally to affect many biological events, including the ini-

tiation of gene expression and replication.

A-T base pairs require less energy to separate than do G-C pairs. Therefore, strand separations will concentrate at A+T-rich regions within a negatively superhelical molecule. However, superhelical stresses cannot be reduced to local sequence analysis. Separation at one site alters its helicity, which changes the level of torsional stress experienced by all other sites. Whether a particular region separates depends not just on its local attributes, but also on how well a separation transition there competes with transitions at all other sites experiencing the same stress. For this reason, the analysis of superhelical transitions requires a global treatment of the entire molecular sequence. Because every base pair can separate, there are 2^N possible states of a molecule containing N base pairs. So the complexity of a direct enumeration of states grows exponentially with molecular size. Finally, strand separation is a heteropolymeric transition, meaning that its free energy cost depends on the identities of the base pairs involved.

The Model

A theoretical method has been developed by Benham that uses statistical mechanical methods to predict the sites on a circular DNA molecule where imposed superhelical stresses destabilize the duplex [1, 2]. Sample calculations analyzing the strand separation transition in superhelical pBR322 DNA have shown that Benham’s algorithm gives accurate results in a less computationally expensive manner. Moreover, analysis of molecules on which experiments detecting superhelical strand separation have been performed yields predictions that agree closely with the experimental results. When applied to a range of genomic DNA sequences, the sites that are predicted to be destabilized by stresses are usually specific types of genetic regulatory elements.

At thermodynamic equilibrium, a population of identical superhelical DNA molecules will distribute itself among all its states of strand separation, with the size of the popu-

lation of individual states decreasing exponentially as the state energy increases. To evaluate this distribution one needs to determine the states and their energies. No exact enumeration method for population size determination is known which avoids enumerating all 2^N states of strand separation for a molecule containing N base pairs. So the approach used by Benham exactly evaluates all the low energy states, then estimates the aggregate influence of the high energy states. The state of lowest energy G_{min} is found, and a threshold energy θ is specified. All the states whose free energies exceed G_{min} by no more than θ are computed. The calculated values of certain quantities then can be refined by estimating the density of high energy states to approximately evaluate the aggregate influence of all the states that do not satisfy this threshold condition. The most important quantity to calculate is the equilibrium probability of separation of each base pair in the superhelical molecule (also called the *transition profile*), as this shows the sites that are strongly destabilized by superhelicity. Benham’s algorithm guarantees an accuracy bound as a function of the threshold θ .

Consider a DNA molecule containing N base pairs on which superhelicity α is imposed. The free energy of a state of strand separation of this molecule depends on the number n of separated base pairs, the number n_{AT} of these that are A-Ts and the number r of contiguous runs of separation:

$$(1) \quad G(n, n_{AT}, r) = \frac{2\pi^2 CK}{4\pi^2 C + Kn} \left(\alpha - \frac{n}{A}\right)^2 + ar + b_{AT}n_{AT} + b_{GC}(n - n_{AT})$$

This equation contains five energy parameters - the torsional stiffness C , the nucleation free energy a needed to initiate separation in a region, the coefficient K of the quadratic free energy associated to superhelicity, and b_{AT} and b_{GC} , the free energies needed to separate one base pair of the appropriate type. Values for all these parameters have been determined experimentally, so there are no free parameters in this model.

According to Formula (1), the lowest energy state from the collection of states with n base pairs in r runs will be the A+T-richest one. The absolute minimum free energy G_{min} can be found from among these states. Because the incremental energy needed to open new sites is higher than that needed to extend existing sites, the state of absolute minimum energy will have a small number of runs of separation (usually 0 or 1).

To compute the contribution of the low energy states, a threshold θ is chosen and all states whose free energy exceeds G_{min} by no more than θ are found. Because the free energy of a state depends only on the values of n , n_{AT} and r , bounds can be found which gives necessary and sufficient conditions for the free energy threshold to be satisfied.

Condition 1 : $n_1 \leq n \leq n_2$

where n_1 and n_2 are the roots of the quadratic inequality:

$$2\pi^2 C K (\alpha - [n/A])^2 + (4\pi^2 C + K n) \times (a + b_{AT} n - G_{min} - \theta) \leq 0$$

Condition 2 : $r \leq r_{max}(n)$

$$\text{where } r_{max} = \left[1 + \frac{G_{min} + \theta - G(n, n, 1)}{a} \right]$$

Condition 3 : $n_{AT} \geq n_{AT_{min}}$

$$\text{where } n_{AT_{min}} = n - \frac{G_{min} + \theta - G(n, n, r)}{b_{GC} - b_{AT}}$$

The collection of values (n, n_{AT}, r) satisfying the above bounds forms the set of low energy states, with which the approximate partition function Z_{cal} is constructed:

$$(2) \quad Z_{cal} = \sum_{i|G(i)-G_{min}<\theta} e^{(-G(i)/RT)}$$

Then the approximate equilibrium value of a parameter ζ is computed to be:

$$(3) \quad \bar{\zeta} = \sum_{i|G(i)-G_{min}<\theta} \zeta_i \frac{e^{(-G(i)/RT)}}{Z_{cal}}$$

For each value of the parameters n , n_{AT} and/or r that does not satisfy the threshold condition a density of states estimate is made of the aggregate contribution $Z(n, n_{AT}, r)$ of all states with the specified values. This can be used to refine the calculated equilibrium average values of any parameter that depends only

on n , n_{AT} and r :

$$(4) \quad \zeta = \frac{\sum_{i|G(i)-G_{min}<\theta} \zeta_i e^{(-G(i)/RT)} + \sum_{neg} \zeta(n, n_{AT}, r) \bar{Z}(n, n_{AT}, r)}{Z_{cal} + \bar{Z}_{neg}}$$

The transition profile cannot be refined in this way because it depends on positional information. However, the density of states calculation may be used to estimate its accuracy. In calculations reported by Benham [2], accuracies exceeding 99% were achieved by specifying a reasonably high threshold θ .

3 The Sequential Algorithm

Benham's algorithm [2] takes as input a DNA sequence of a circular molecule and computes the transition profile and other important parameters in the strand separation transition for a range of superhelicities α . The basic method consists of three phases:

Phase 1: Sequence Preprocessing

In this phase the base sequence of the molecule of N characters is analyzed to determine information needed at later stages in the calculation. Identification of states and evaluation of their energetics relies on knowledge about the locations and A+T-richness of all runs of length i , up to a maximum of L_{max} base pairs. The molecule being circular, there are N runs of length i , one starting at each of the N positions in the sequence. Two matrices, *richness* and *location* with dimensions $L_{max} \times N$ are constructed. *richness*[i][j] is the A+T content of the j th A+T-richest run of length i and *location*[i][j] gives the location of the first base pair of that run. The *richness* matrix is computed by first gathering information about the A+T content of the different runs and applying a counting sort procedure to order the matrix entries for a fixed run length i by decreasing A+T-richness, which corresponds to increasing energetic cost of strand separation. For separation transition, states involving multiple runs, the *location* matrix will be used to determine whether the runs of a candidate state overlap

or abut, or whether they are distinct and thus determine an actual state.

Two additional matrices *start* and *stop* with dimensions $L_{max} \times L_{max}$ are constructed. *start*[i][j] and *stop*[i][j] are the first and the last position respectively in the *richness* and *location* matrix where runs of length *i* and A+T-richness *j* occur.

Finally, the $3 \times L_{max}$ matrix *max_at* is computed, where *max_at*[i][j] is the A+T-richness of the A+T-richest state containing *j* total separated base pairs in *i* distinct runs. Computing *max_at*[1][j] (states with only one run) is found in *richness*[j][1]. *max_at*[2][j] has to consider valid 2 run states (non-overlapping or abutting runs) to find the A+T-richest one. Because of large storage requirements, positional information about A+T-rich states is not kept; computation for larger number of runs will make use of A+T-richness information but has to re-check positional information in order to validate a considered state. *max_at*[3][j] considers valid 3 run states by combining 1 run and 2 run states and checking the validity by testing positional information.

The following two phases are executed for a range of linking differences α .

Phase 2: Determining the contribution of low energy states

During this phase the absolute minimum free energy G_{min} is computed and all states that exceed G_{min} by no more than a specified threshold θ are found. The contribution of these low energy states to the partition function Z_{cal} , equilibrium probability that each base pair in the sequence undergoes separation $p[k]$ $1 \leq k \leq N$, ensemble averages of free energy G , as well as other parameters is computed.

Using *max_at*[r][n], the free energy of states with *n* base pairs in *r* runs containing *max_at*[r][n] A+T base pairs is computed using Formula (3) and the minimum free energy G_{min} is chosen from the resulting collection.

To satisfy the free energy threshold criterion $G(n, n_{AT}, r) - G_{min} < \theta$, bounds on the values of *n*, n_{AT} and *r* are found using the formulas

from the previous section. First the roots of the quadratic inequality Condition 1 are determined and the range of *n* is restricted to values $n_1 \leq n \leq n_2$. After fixing the value of *n* in this range, the upper bound on the number of runs r_{max} can be computed from Condition 2 and $n_{AT_{min}}$ from Condition 3. The contribution to the partition function, separation transition profile, average free energy and other parameters of all valid states having up to r_{max} , but not exceeding 3 runs of transition of total length *n* and A+T-richness larger than $n_{AT_{min}}$ are computed using matrices *richness* and *location*.

Phase 3: Estimating the contribution of high energy states

The partition function as well as other parameters like the number of runs of transition, of separated base pairs and of separated A+T base pairs can be refined using Formula (4).

4 The Parallel Approach

Of the three phases of the DNA strand separation algorithm described in Section 3, the second phase in which the contributions of low energy states are computed is by far the most time consuming phase of the computation. This section describes how we parallelized the second phase, focusing on the issues of program partitioning and task scheduling.

Program Partitioning

For a highly iterative computation as the one featured in this phase, loop-based decomposition is the common method used for program partitioning. In our case, up to a certain granularity of decomposition, the algorithm exhibits an embarrassing parallelism with inter-task communication limited to task set up and result accumulation.

The second phase of the algorithm consists in computing the parameters characterizing strand separation for a range of linking differences. We chose to split the computation

into steps, each corresponding to an iteration in this range and to partition each step into independent tasks. Within each iteration, the innermost loop corresponds to the computation of the contributions from states with a given number of runs, total length of the runs and A-T richness higher than a computed threshold. We chose this as the basic task in our parallelization. To eliminate repeated communication due to global data updating, we used local accumulators on each processor. The local accumulators are added to global accumulators only once at the end of each iteration.

This partitioning allowed us to investigate the computational profile of each iteration which is highly dependent on the corresponding value of the linking difference α .

Task Scheduling

The partitioning we chose creates enough parallelism (number of tasks) for a small parallel system. Unfortunately, the amount of computation executed by each task is variable and unpredictable. To avoid performance loss, task scheduling must maintain a good workload balance among processes. For this study we have evaluated two different task scheduling techniques, a simple static task scheduling as well as a dynamic scheduling method. With static scheduling tasks are assigned to processes following a round-robin distribution. For dynamic scheduling a collection of queues is used, into which tasks are inserted and from which tasks are removed in order to be executed. This collection of tasks may be either a single centralized queue or a set of distributed queues. A centralized queue is simpler, but requires communication and is subject to contention. With distributed queues, each processor has an attached queue and an initial assignment of tasks to them.

The static scheduling has negligible overhead, but causes significant load imbalance for larger number of processors. Dynamic scheduling achieves better load balancing, provided there is enough parallelism, but at the cost of non-zero overhead caused by both mutual

exclusion synchronization and communication due to task queue update.

To reduce the synchronization overhead involved in dynamic scheduling we implemented a task stealing scheme with local initial scheduling. At the beginning of each iteration tasks are assigned statically to local queues. As long as it is not empty, a process executes tasks only from the local queue, thus avoiding communication and synchronization. When its local queue becomes empty, the process takes tasks from adjacent queues, proceeding modularly to all queues. This technique is usually known as task stealing and it is largely employed in irregular fine-grain shared memory applications [3].

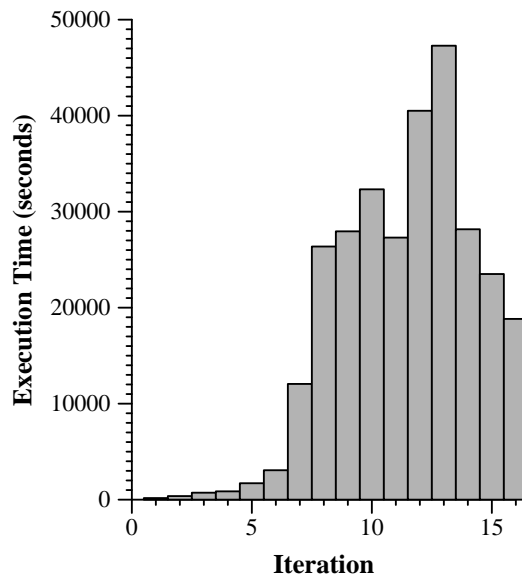


Figure 1: Sequential Execution Time

5 Performance

This section describes the platform, software environment, input, output and the performance results as overall speedup and speedup per iteration for 4, 8 and 16 processors.

Platform

The parallel implementations of the DNA strand separation computation using static and dynamic scheduling have been evaluated on an

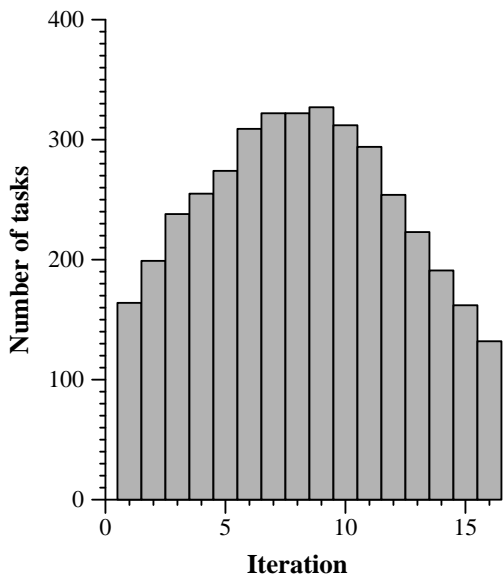


Figure 2: Number of Tasks

SGI Challenge multiprocessor. The SGI Challenge is a cache-coherent bus-based centralized shared memory multiprocessor. The system we used has 16 150 MHz MIPS 4400 processors, with a 16 KBytes data and 16 KBytes instruction direct-mapped first level cache and a 1 MBytes 2-way associative unified second level cache. The system has 1 GBytes of main memory. Our parallel programs are written in C using the Argonne National Laboratories (ANL) *parmacs* [4] macros for parallel constructs.

Input

The DNA sequence analyzed contains 4363 base pairs and we considered a maximum run length of 200 base pairs. This sequence is the same as used in previous calculations [1, 2]. The energy threshold used was of 12 Kcal and the linking difference range was between -20 and -50 turns.

Figure 1 shows the sequential execution time for the 16 iterations which are required for the analysis of the pBR322 sequence. Due to the large growth of the number of considered states, the execution time increases noticeably up to the 8th iteration from about 40 seconds to about 30000 seconds. For the remaining iterations (9th to 16th) the execution time stays

within the same order of magnitude because the conditions result in a smaller set of states to be considered. The total sequential execution time for the DNA strand separation computation is about 3 days on a 150 MHz MIPS R4400.

Figure 2 shows the number of tasks per iteration generated by the program partitioning described in Section 4. Each iteration has between 100 and 300 tasks. This means that each processor has to process between 6 and 20 tasks per iteration in the 16 processor case. Task length varies significantly from one iteration to another, but because iterations are separated through global synchronization barriers, only the variation in task length within the same iteration causes load imbalance.

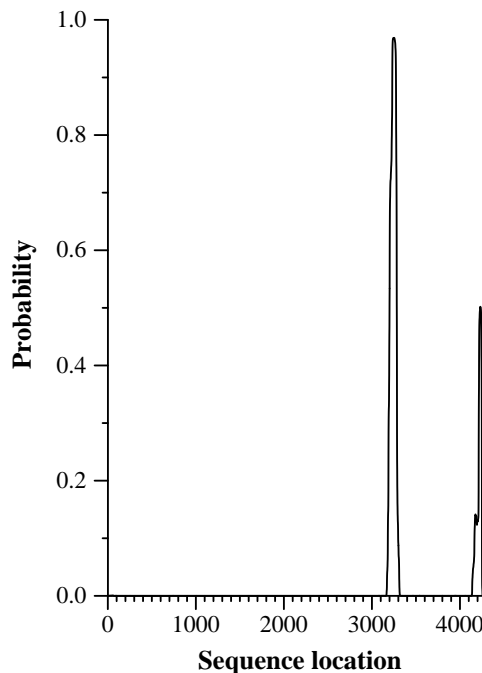


Figure 3: Separation transition profile

Output

The output parameter that provides the most biologically useful information is the transition profile. This is the equilibrium probability $p(x)$ of strand separation of the base pair at each position x along the sequence. The probability profile we calculated for the pBR322 DNA

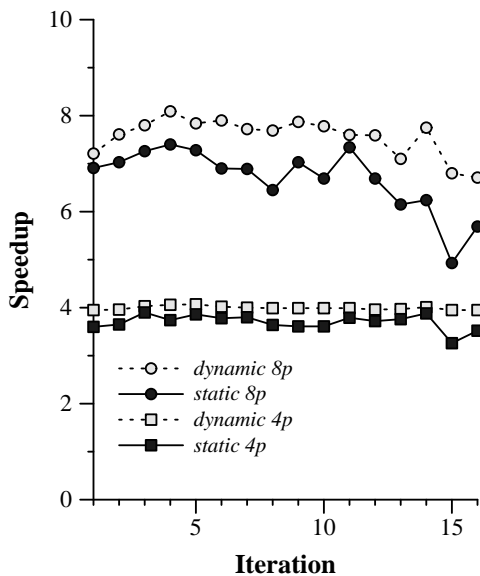


Figure 4: Speedup for 4 and 8 processors

sequence at a linking difference of $\alpha = -30$ turns is shown in Figure 3. Transition is seen to be confined to two positions, centered around 3250 and 4200 BP, respectively. These are the precise locations where separation was experimentally detected to occur in this molecule [5].

Speedup

The overall speedup values in the static scheduling case are: 3.7 for 4, 6.4 for 8 and 7.3 for 16 processors. Using dynamic scheduling the values of the speedup are: 3.9 for 4, 7.3 for 8 and 11.9 for 16 processors. This means an improvement for the dynamic scheduling method over the static one of 5.4% for 4, 14% for 8 and 63% for 16 processors. Figures 4 and 5 show the speedup for both scheduling methods measured separately for each iteration. In the 4 processor case the scheduling technique has a negligible impact on performance, although dynamic scheduling is slightly better. On 8 processors, the performance is also very good, although the gap between dynamic and static scheduling increases especially in the computationally intensive iterations (8 to 16). The performance efficiency also drops compared to the 4 processor case, particularly for the last two iterations when the number of tasks de-

creases while the computation remains high.

If only the performance for 4 and 8 processors had been evaluated, then one could have drawn the conclusion that a simple parallelization using static scheduling suffices. The speedup on 16 processors (Figure 5) clearly shows the advantage of the dynamic scheduling method. Increasing the number of processors to 16 exposes more load imbalance due both to variation in task size and limited parallelism. Dynamic scheduling reduced the imbalance, outperforming the static scheduling method by as much as a factor of 2. Nevertheless, even with dynamic scheduling, the program still suffers from imbalanced work load due to the relatively small number of tasks and their variation in length. This granularity effect is particularly significant for the last iterations, because of small number of tasks of large size.

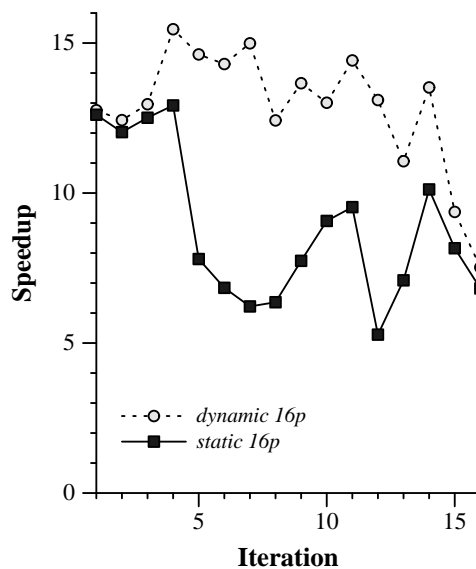


Figure 5: Speedup for 16 processors

Load Balancing

To confirm that the differences in speedup are due to load imbalance, we plotted in Figures 6 and 7 the execution time per processor in the 16 processor case for the 10th computational iteration, for which the speedup of the two scheduling methods differed by a fac-

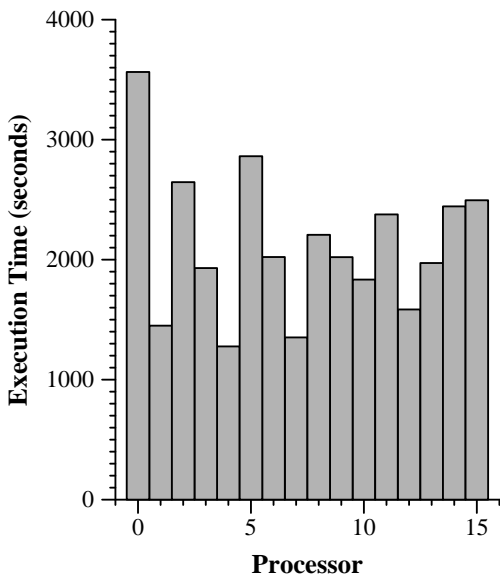


Figure 6: Execution Time using static scheduling for iteration 10

tor of 1.5. We can see that the workload is dramatically imbalanced for static scheduling while it becomes well balanced when dynamic scheduling is employed. There is an additional overhead for dynamic scheduling due to task queue access communication and synchronization, which makes the cumulated execution time in the dynamic scheduling case larger than the cumulated execution time in the static scheduling case.

Memory requirement

To store the main matrices used in the computation, the program requires approximately 7 MBytes mostly read-only data. Since the second-level cache for the SGI Challenge architecture is 1 MB, superliniar speedup effects occasionally occur.

6 Discussion

Sample calculations indicate that the algorithm scales approximately quadratically with molecular length, and exponentially with the threshold θ . However, this scaling depends strongly on the specific DNA sequence being analyzed. Fixing θ and the sequence length N ,

the number of states satisfying the threshold condition will increase rapidly with the overall A+T-content of the molecule being analyzed. The computation time also depends strongly on the distribution of A+T-richness within the sequence. As the A+T-richness becomes increasingly concentrated in a single region, a given set of A+T-rich runs which together satisfy the threshold condition becomes more likely to contain a pair of runs that either overlap or abut. So, many of the candidate states satisfying Conditions 1-3 will be rejected because the runs involved are not distinct. In sequences where the A+T-richness is either more evenly distributed or concentrated at several positions, more of the candidate states will be comprised exclusively of distinct runs. Thus, the time required to complete the analysis increases.

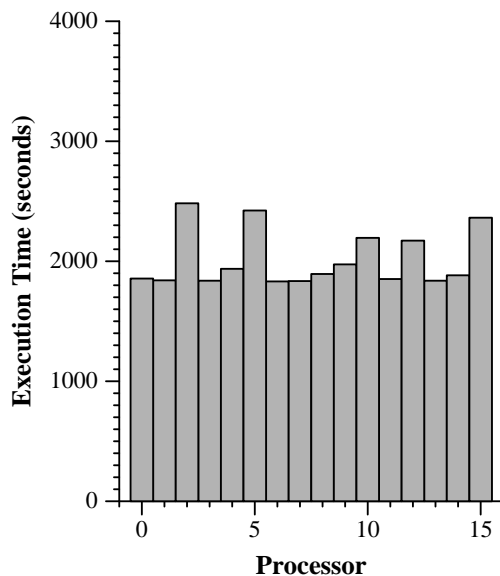


Figure 7: Execution Time using dynamic scheduling for iteration 10

The length of the circular DNA molecules for which this prediction analysis is applicable varies from hundreds of base pairs to 50K base pairs, with the most common length around 10K.

Larger sizes of the molecules and higher concentration of A-Ts result in increased execution time of the algorithm, making the parallel approach a stringent necessity. The experi-

ments we conducted show that even a small-scale multiprocessor (<32 nodes) gives very good speedup, reducing the execution time from days to hours.

7 Conclusions

In this paper we presented a simple parallelization approach for DNA strand separation computation. Although the parallel approach does not reduce the complexity of the problem, it is nevertheless of practical interest for the molecular biology research community because it extends the range of computationally tractable tools for DNA sequence analysis. For example, on a 16-processor SGI Challenge, the total execution time for the 4K base DNA molecule we chose as input is reduced from about 3 days to about 7 hours.

Starting from a method and the corresponding sequential algorithm developed by Benham we implemented and evaluated two parallel solutions based on static and dynamic scheduling, respectively. For the chosen input the dynamic scheduling solution outperforms the static scheduling one by as much as 63% in the 16 processor case.

Acknowledgements

We are very grateful to Michiel Noordewier who encouraged us to pursue this study.

References

- [1] C.J. Benham. *Theoretical analysis of heteropolymeric transitions in superhelical DNA molecules of specified sequence*. J. Chem. Phys. 92, 6294–6305, 1990.
- [2] C.J. Benham. *Energetics of the Strand Separation Transition in Superhelical DNA*. J. Mol. Biol. 225, 835–847, 1992.
- [3] Jaswinder Pal Singh, Anoop Gupta and Marc Levoy. *Parallel Visualization Algorithms: Performance and Architectural Implications*. IEEE Computer 27(7), June 1994.
- [4] Ewing L. Lusk and Ross A. Overbeek. *A minimalist approach to portable, parallel programming*. In Leah H. Jamieson, Dennis B. Gannon and Robert J. Douglass, editors, *The Characteristics of Parallel Algorithms*, pages 351–362, MIT Press, 1987.
- [5] D. Kowalski, D. Natale and M. Eddy. *Stable DNA Unwinding, not "Breathing", Accounts for Single-Strand-Specific Nuclease Hypersensitivity of Specific A+T-Rich Sequences*. Proc. Nat'l. Acad. Sci. USA 85, 9464–9468, 1988.