Chapter 29

Biomedical Applications Modelling

Suchendra Bhandarkar, Sridhar Chirravuri, and Salem Machaka

Department of Computer Science
The University of Georgia
Athens, Georgia 30602–7404, USA
Email: suchi@cs.uga.edu, chirravu@cs.uga.edu, machaka@cs.uga.edu

29.1 Introduction

In this chapter we present two biomedical applications of cluster computing using PVM [15]. The first deals with the problem of chromosome reconstruction via ordering of clones from a genomic library. This problem of chromosome reconstruction or physical mapping is central to the field of molecular genetics and can be shown to be isomorphic to the classical NP-complete Optimal Linear Arrangement (OLA) problem. We propose parallel algorithms for simulated annealing (SA) and microcanonical annealing (MCA) and apply them to this problem. The second problem deals with the analysis of heart rate variability (HRV) from an electrocardiograph (EKG) signal. An EKG measures the electrical activity of the heart and when plotted as a function of time, it exhibits certain characteristic peaks called R peaks which reflect the dominant electrocardiac phases. The instantaneous heart rate is computed as the inverse of the time period between two successive R peaks and is observed to be a highly complex non-linear function of time. The Kolmogorov ($K_2$) entropy, a commonly used measure of the HRV, characterizes the non-linear complexity of electrocardiac activity. We design a parallel algorithm for the computation of $K_2$ entropy and show its utility in real-time analysis of HRV. We describe the implementation of the above algorithms on a cluster of UNIX workstations running PVM. We also analyze and discuss the convergence, speed-up and scalability characteristics of the various parallel algorithms.
29.2 The Chromosome Reconstruction Problem

A central problem in genetics, right from its very inception, is that of creating maps of entire chromosomes which could then be used to reconstruct the chromosome's DNA sequence. These maps are central to the understanding of the structure of genes, their function and their evolution. The large number of DNA markers and the ease with which molecular markers are assayed have shifted the problem focus from the experimental collection of data to the computational issues of assembling entire maps.

Chromosomal maps fall into two broad categories – genetic maps and physical maps. Genetic maps, which are of low resolution (i.e., 1–10 million base pairs (Mb)), represent an ordering of genetic markers along a chromosome where the distance between two genetic markers is inversely proportional to their recombination frequency. While genetic maps narrow the search for genes to a particular chromosomal region, it is a physical map that ultimately allows the recovery and molecular manipulation of genes of interest. A physical map is defined as a partial ordering of distinguishable DNA fragments or clones by their position along the entire chromosome where the clones may or may not contain genetic markers. A physical map has a much higher resolution than a genetic map of the same chromosome (i.e., 10–100 thousand base pairs (Kb)).

29.2.1 Physical Mapping via Clone Ordering

The specific technique, discussed in this chapter, for generating a physical map is one based on determining an ordering of clones from a library that optimizes a prespecified objective function [7]. The optimal ordering of clones, with respect to their position along a chromosome, is then deemed to constitute a physical map. Previous work in the Department of Genetics at the University of Georgia resulted in a physical mapping algorithm ODS (Ordering DNA Sequences) based on simulated annealing [7]. The physical mapping approach in ODS can be summarized as follows:

(i) Each DNA fragment or clone in the library is scored for the presence or absence of specific oligonucleotide sites or probes by a series of biochemical experiments. This results in the assignment of a digital signature to each clone.

(ii) The Hamming distance \(d(C_i, C_j)\) between two clonal signatures \(C_i\) and \(C_j\) is defined to be the measure of dissimilarity between their signatures.

(iii) The total linking distance \(D\) for an ordering is defined as the sum of the Hamming distances between all pairs of successive clones in the ordering:

\[
D = \sum_{i=1}^{n-1} d(C_i, C_{i+1}) \quad \text{(29.2.1)}
\]
(iv) The desired (i.e., optimal) ordering or physical map is deemed to be that which results in minimization of the total linking distance $D$. This ordering criterion is based on the intuitive rationale that clones with similar digital signatures tend to overlap along the chromosome and hence should be placed next to each other on the physical map.

Let $D_m$ denote the minimum linking distance associated with the space of all possible clonal orderings and $D_0$ the linking distance associated with the true ordering. It can be shown that

$$\lim_{n \to \infty} \text{prob}(|D_m - D_0| > \epsilon) = 0$$

(29.2.2)

That is to say, $D_m$ converges in probability to $D_0$ as the size $n$ of the clonal library grows [16]. This result provides the formal basis for the physical mapping algorithm ODS. Figure 29.1 shows the physical map of Chromosome IV of the fungus *Aspergillus nidulans* constructed using ODS.

The problem of computing such an optimal clone ordering or physical map can be shown to be isomorphic to the classical NP-complete Optimal Linear Arrangement (OLA) problem [8]. No polynomial-time algorithm for finding the optimal solution to the OLA problem is known except for some simple cases that deal with error-free data [4]. Stochastic optimization algorithms such as SA [9] or MCA [6] are capable of avoiding local optima in the solution space and producing solutions that are close to a global optimum in polynomial time on average. One of the drawbacks of a serial implementation of SA and MCA is that the annealing schedules necessary to obtain a solution close to a global optimum, are computationally intensive. Parallel processing on a cluster of workstations is one of the ways in which this drawback can be alleviated [2].

### 29.3 PVM Algorithms for Chromosome Reconstruction

#### 29.3.1 Simulated Annealing and Microcanonical Annealing

Stochastic optimization algorithms such as simulated annealing (SA) [9] and microcanonical annealing (MCA) [6] are a subcategory of stochastic hill-climbing search techniques and are characterized by their capacity to escape from local optima in the objective function. A single iteration of a stochastic optimization algorithm consists of three phases: (i) perturb, (ii) evaluate, and (iii) decide.

In the perturb phase, the current solution $x_i$ to a multivariate objective function $E(x)$ which is to be minimized (i.e., the linking distance $D$ in our case), is systematically perturbed to yield another candidate solution $x_t$. In our case, the clone ordering is permuted by reversing the ordering within a randomly chosen block of clones. In the evaluate phase, $E(x_t)$ (i.e., the linking distance $D$ for the new clone ordering) is computed. In the decide phase, $x_t$ is accepted and replaces $x_i$ *probabilistically* using a stochastic decision function. The stochastic decision function is *annealed* in a manner such that the search process resembles a random search in
Figure 29.1 An ordered physical map of *Aspergillus nidulans* Chromosome IV.
the earlier stages and a greedy local search or a deterministic hill-climbing search in the latter stages. The major difference between SA and MCA arises from the difference in the stochastic decision function used in the decision phase. But their common feature is that, starting from an initial solution, they generate, in the limit, an ergodic Markov chain of solution states which asymptotically converges to a stationary Boltzmann distribution. The Boltzmann distribution asymptotically converges to a globally optimal solution when subject to the annealing process [9].

Simulated Annealing

In the decide phase of SA, the new candidate solution \( x_j \) is accepted with probability \( p \), which is computed using the Metropolis function given by

\[
p = \left\{ \begin{array}{ll}
1 & \text{if } E(x_j) < E(x_i) \\
\exp \left( -\frac{E(x_j) - E(x_i)}{T} \right) & \text{if } E(x_j) \geq E(x_i)
\end{array} \right. \quad (29.3.1)
\]

or the Boltzmann function \( B(T) \) given by:

\[
p = B(T) = \frac{1}{1 + \exp \left( \frac{E(x_j) - E(x_i)}{T} \right)} \quad (29.3.2)
\]

at a given value of temperature \( T \), whereas \( x_i \) is retained with probability \( 1 - p \).

The Metropolis function and the Boltzmann function give SA the capability of probabilistically accepting new candidate solutions that are locally suboptimal compared to the current solution, thus enabling it to climb out of local minima. Several iterations of SA are carried out for a given value of \( T \), which is then systematically reduced using an annealing function. The iterations carried out for a single value of \( T \) are referred to as an annealing step. As can be seen from equations (29.3.1) and (29.3.2), at sufficiently high temperatures, SA resembles a completely random search, whereas at lower temperatures it acquires the characteristics of a deterministic hill-climbing search or local greedy search.

Both the Metropolis function and the Boltzmann function ensure that SA generates an asymptotically ergodic (and hence stationary) Markov chain of solution states at a given temperature value. It has been shown that logarithmic annealing schedules of the form \( T_k = R/\log k \) for some value of \( R > 0 \) are asymptotically good, i.e., they ensure asymptotic convergence to a global minimum with unit probability in the limit \( k \to \infty \) [9].

Microcanonical Annealing

MCA models a physical system whose total energy, i.e., sum of kinetic energy and potential energy, is always conserved. The potential energy of the system is the multivariate objective function \( E(x) \) to be minimized, whereas the kinetic energy \( E_k > 0 \) is represented by a demon or a collection of demons. In the latter case, the total kinetic energy is the sum of all the demon energies. The demon energy
(or energies) serve(s) to provide the system with an extra degree (or degrees) of freedom, enabling MCA to escape from local minima.

In the decide phase of MCA, if $E(x_j) < E(x_i)$ then $x_j$ is accepted as the new solution. If $E(x_i) \geq E(x_j)$ then $x_i$ is accepted as the new solution only if $E_k \geq E(x_j) - E(x_i)$. If $E(x_j) \geq E(x_i)$ and $E_k < E(x_j) - E(x_i)$ then the current solution $x_i$ is retained. In the event that $x_j$ is accepted as the new solution, the kinetic energy demon is updated $E_k^{n+1} = E_k^n + [E(x_i) - E(x_j)]$ in order to ensure the conservation of the total energy. The kinetic energy parameter $E_k$ is annealed in a manner similar to the temperature parameter $T$ in SA. MCA can also be shown to converge to a global minimum with unit probability given a logarithmic annealing schedule [6]

### 29.3.2 Parallel SA and MCA Using PVM

As described earlier, a candidate solution in the serial SA or MCA algorithm can be considered to be an element of an asymptotically ergodic first-order Markov chain of solution states. Consequently, we have formulated and implemented two models of parallel SA (PSA) and parallel MCA (PMCA) algorithms based on the distribution of the Markov chain of solution states using PVM.

- The Non-Interacting Local Markov chain (NILM) PSA and PMCA algorithms
- The Periodically Interacting Local Markov chain (PILM) PSA and PMCA algorithms.

In the NILM PSA and NILM PMCA algorithms, each processor within the PVM system runs an independent version of the serial SA or MCA algorithm, respectively. In essence, there are as many Markov chains of solution states as there are physical processors within the PVM system. Each Markov chain is local to a given processor. and at any instant of time, each processor maintains a candidate solution which is an element of its local Markov chain of solution states. The serial SA or MCA algorithm run asynchronously on each processor, i.e., at each temperature value or kinetic energy value, each processor iterates through the perturb-evaluate-accept cycle COUNT.LIMIT number of times concurrently (but asynchronously) with all the other processors.

A parallel random number generator is used to generate the Markov chains of solution states in the perturb phase. By assigning a distinct seed to each processor at the start of execution, we ensure the independence of the Markov chains on the various processors. The evaluation function and the decision function (for PSA and PMCA) are executed concurrently on the solution state within each processor. On termination of the annealing processes on all the processors, the best solution is selected from among all the solutions on the individual processors. The NILM model is essentially that of multiple independent searches.

The PILM PSA or PILM PMCA algorithm is similar to its NILM counterpart except for one major difference. Just before the parameter $T$ or $E_k$ is updated
using the annealing function, the best candidate solution from among those in all the processors is selected and duplicated on all the other processors. The goal of this synchronization procedure is to focus the search in the more promising regions of the solution space, which suggests that the PILM PSA or PILM PMCA algorithm should be potentially superior to its NILM counterpart. The PILM model is essentially that of multiple periodically interacting searches.

In the case of all the four algorithms, NILM PSA, NILM PMCA, PILM PSA and PILM PMCA, a master process is used as the overall controlling process. The master process runs on one of the processors within the PVM system. The master process spawns child processes on each of the other processors, broadcasts the data subsets needed by each child process, collects the final results from each of the child processes and terminates the child processes. In addition to the above-mentioned functions for task initiation, task coordination and task termination, the master process also runs its own version of the SA or MCA algorithm just as any of its child processes.

In the case of the PILM PSA and the PILM PMCA algorithms, at the end of the COUNT_LIMIT number of perturb-evaluate-decide iterations at each temperature or kinetic energy value, the master process collects the results from each child process along with its own result, broadcasts the best result to all the child processes, and also replaces its own result with the best result. The master process updates its temperature or kinetic energy value using the annealing function and proceeds with its local version of the SA or MCA algorithm. On reaching the final temperature or kinetic energy value, the master process collects the final results from each of the child processes along with its own, selects the best result as the final solution and terminates the child processes. The master process for the NILM PSA or NILM PMCA algorithms is similar to that of its PILM counterpart except for the absence of periodic interaction between the master process and any of the child processes during the annealing procedure. The master process for all the four algorithms is depicted in Figure 29.2.

The child process, in the case of the PILM PSA and PILM PMCA algorithms, collects the clonal data and certain initialization parameters from the master process. Each of the child processes runs an independent version of the SA or MCA algorithm on its data set. The child processes interact periodically with the master process. At the end of COUNT_LIMIT iterations at each temperature or kinetic energy value, each child process sends its result to the master process and waits for the master process to transmit the best result thus far. On receipt of the best result, each child process updates its temperature or kinetic energy value using the annealing function and proceeds with its local version of the SA or MCA algorithm on the received result. When the final value of the temperature or kinetic energy is reached, each child process transmits its result to the master process. The child process, in the case of the NILM PSA or NILM PMCA, algorithm, is similar to that of its PILM counterpart except for the absence of periodic interaction between the master process and any of the child processes. The child process for all the four algorithms is depicted in Figure 29.3.
master();

begin master
  Phase 1: Initial Setup:
  (a) Spawn Child Processes;
  (b) Read Input: clonal data, clonal distance matrix and
     initial seeds for the parallel random number generator;
  (c) Broadcast the input to all spawned child processes;

  Phase 2: The Annealing Algorithm and Process Coordination
  in the case of PSA:
    T = T_max; Finished = (T <= T_min);
  in the case of PMCA:
    E_k = E_max; Finished = (E_k <= E_min);

  while (not Finished)
    begin while
      for (Count = 1; Count <= COUNT_LIMIT; Count = Count + 1;)
        begin for
          (a) Perturb Phase: same as serial SA or MCA;
          (b) Evaluate Phase: same as serial SA or MCA;
        
          (c) Decide Phase:
          if PSA: SA decision criterion:
          if PMCA: MCA decision criterion:
          endif for

          In the case of PILM PSA or PILM PMCA:
            (a) Receive clone ordering from each child process;
          (b) Select best clone ordering;
          (c) Send best clone ordering to each child process;

        if PSA: Update temperature $T = A(T)$;
        if PMCA: Update kinetic energy $E_k = A(E_k)$;
      endwhile

    Phase 3: Output Result:
      (a) Receive clone ordering from each child process;
      (b) Select best clone ordering;

  end while;
child();
begin
  Phase 1: Initial Setup;
  Receive input from Master Process: clonal data, clonal distance
  matrix and initial seeds for parallel random number generator;

  Phase 2: Annealing Algorithm and Coordination with Master Process;
  in the case of PSA:
    T = T_max; Finished = (T <= T_min);
  in the case of PMCA:
    E_k = E_max; Finished = (E_k <= E_min); 
  while (not Finished)
    beginwhile
      for (Count = 1; Count <= COUNT_LIMIT; Count = Count +1)
        beginfor
          (a) Perturb Phase: same as serial SA or MCA;
          (b) Evaluate Phase: same as serial SA or MCA;
          (c) Decide Phase:
            if PSA: SA decision criterion;
            if PMCA: MCA decision criterion;
        endfor
      endfor
    in the case of PILM PSA or PILM PMCA:
      (a) Send clone ordering to Master process;
      (b) Receive best clone ordering from Master process;
      if PSA: Update temperature T = A(T);
      if PMCA: Update kinetic energy E_k = A(E_k);
    endwhile

  Phase 3:
    Send clone ordering to Master Process and Exit;
endchild:

Figure 29.3 The child process for the PILM/NILM PSA/PMCA algorithms
29.4 Heart Rate Variability and Kolmogorov Entropy

An EKG signal measures the electrocardiac activity of the heart. A typical EKG signal, when plotted as a function of time, exhibits certain characteristic peaks, called R peaks, which reflect the dominant electrical phases of the underlying heart beat phenomenon. The instantaneous heart rate is computed as the inverse of the time interval between two consecutive R peaks in the EKG signal and is typically a very complex function of time. In most biological systems, the heart rate is monitored and continuously adjusted by a complex and highly sensitive regulatory mechanism that responds to several extracardiac stimuli [13]. For example, variations in the instantaneous heart rate may be caused by several factors such as body position, physical activity, body temperature, respiration rate, blood volume, blood pressure and emotional state [12]. We use the term heart rate variability (HRV) to denote the fact that the instantaneous heart rate is a complex nonlinear function of time. It is imperative in many situations to be able to characterize and measure HRV from the EKG signal in order to quantitatively analyze the underlying cardiac activity.

It is difficult to reconstruct a complex dynamic system such as the heart, which is believed to have several degrees of freedom, using a relatively few time series data points from the EKG. Consequently, several mathematical techniques have been developed to characterize the HRV. These techniques include classical techniques based on the computation of statistical parameters of time series data such as the mean, standard deviation, power spectrum and autoregression (AR) [12]. The more recent techniques are based on the computation of parameters that characterize the underlying nonlinear chaotic dynamics of the HRV such as the correlation-dimension, Lyapunov exponents [1], Kolmogorov (i.e., $K_2$) entropy [11] and Approximate Entropy (ApEn) [12].

The HRV is not only a measure of the nonlinear complexity of the underlying heart rate data, but also has significant clinical ramifications. Qualitative analysis of the HRV has often been used as an indicator of cardiovascular health by cardiologists [14]. The heart rate of healthy persons is known to have less regularity (i.e., higher degree of chaos) than the heart rate of cardiac patients [10]. More specifically, low HRV in patients with an acute myocardial infarction often implies greater risk for short-term cardiac morbidity and mortality [3]. HRV analysis has been used successfully in humans to noninvasively evaluate autonomic responses to specific maneuvers and drugs, as well as responses to more chronic pre-existing pathologic conditions [12]. For example, low HRV in a fetus is an early indication of fetal distress [3]. Due to its close association with cardiovascular health, it is important to be able to quantitatively measure and analyze HRV.

29.4.1 $K_2$ Entropy

Kolmogorov formulated a method to compute the entropy and correlation dimension of chaotic dynamic systems [11]. Grassberger and Procaccia [11] developed a measure, namely, the $K_2$ entropy, as an estimate of the asymptotic lower bound
for the metric Kolmogorov entropy computed from time series data. The algorithm used to compute $K_2$ entropy can also be used to estimate a good lower bound for the correlation dimension of complex dynamic systems. The serial computation of $K_2$ entropy from time series data is described as follows:

Consider a time series with $N$ points. Let $x_i$ denote the $ith$ point in the time series. The time series data can be vectorized by constructing $d$-dimensional vectors where each vector consists of $d$ consecutive points in the time series $X^d = \{x_1, x_{i+1}, x_{i+2}, \ldots, x_{i+d-1}\}$. In such a $d$-dimensional vectorization (or embedding) of the time series data, the correlation integral $C_d(\epsilon)$ is defined as the limiting ratio of the number of vector pairs with distance (in $d$-dimensional Euclidean space $\mathbb{R}^d$) between them not exceeding $\epsilon$ to the square of the number of points in the time series, i.e., $N^2$:

\[ C_d(\epsilon) = \lim_{N \to \infty} \frac{1}{N^2} \left( \# \{(n, m) \text{ such that } \left\| \sum_{i=1}^{d} |x_{n+i} - x_{m+i}|^2 \right\|^{1/2} < \epsilon \} \right) \]  

(29.4.1)

The $K_2$ entropy, is given by:

\[ K_2 = \lim_{\epsilon \to 0} \lim_{d \to \infty} \frac{1}{\tau} \ln \left( \frac{C_d(\epsilon)}{C_{d+1}(\epsilon)} \right) \]  

(29.4.2)

In our case, the Takens constant $\tau$ was assumed to be unity. The $K_2$ entropy is a quantitative measure of the complexity or degree of chaotic behavior of the dynamic system underlying the time series data. In general, lower values of $K_2$ entropy indicate a certain degree of order (or periodicity) in the behavior of the dynamic system, whereas a higher values of $K_2$ entropy indicate more complex or chaotic behavior. The $K_2$ entropy values are interpreted and classified as follows:

(i) $K_2$ entropy = 0 implies that the dynamic system is either constant or periodic.

(ii) $K_2$ entropy $> 1$ (approaching $\infty$) implies that the dynamic system is random.

and

(iii) $0 < K_2$ entropy $< < \infty$ implies that the dynamic system is chaotic.

For the purpose of verification of the algorithms for computation of the correlation integral and the $K_2$ entropy, a model system, i.e., the Henon system, was considered. The Henon system is a model chaotic system whose time series data points are generated using the recurrence equations [1):

\[ X_{n+1} = \alpha - X_n^2 + \beta Y_n \]
\[ Y_{n+1} = X_n \quad \text{where} \quad \alpha = 1.4, \ \beta = 0.3 \]  

(29.4.3)

The Henon system is known to have a characteristic $K_2$ entropy of 0.325 and a correlation dimension of 1.22 [11].
Figure 29.4 Correlation integral curves for the Henon system.

Figure 29.4 shows the plot of $C_d(\epsilon)$ as a function of $\epsilon$ on a dual logarithmic (i.e. log-log) scale for various values of the embedding dimension $d$ for the Henon model system. This plot is referred to as the correlation integral (i.e., $C_d$) plot and the curves in the plot are referred to as the $d$-curves. With reference to the $C_d$ plot in Figure 29.4, the region where all the $d$-curves merge into a single horizontal line is
called the saturation region. In the saturation region, the $d$-dimensional distances between all possible vector pairs are contained within a hypersphere of radius $\epsilon$, resulting in an asymptotic value of unity for $C_d(\epsilon)$ (equation (29.4.1)).

The depopulation region in the $C_d$ plot is characterized by small values of $\epsilon$ where there are very few or no vector pairs whose inter-vector distances in $\mathbb{R}^d$ are less than the value of $\epsilon$. The depopulation region (Figure 29.4) is characterized by the abrupt termination of the $d$-curves for small values of $\epsilon$. This is particularly noticeable in the case of $d$-curves at higher dimensional embeddings (i.e., higher $d$ values) where the possibility of finding vector pairs with inter-vector distances less than $\epsilon$ reduces dramatically.

The scaling region in the $C_d$ plot is defined as the one between the saturation and depopulation regions where all the $d$-curves have a constant slope and are parallel to each other (Figure 29.4). The $K_2$ entropy is computed as the asymptotic distance between successive $d$-curves in the scaling region in the limit $d \to \infty$. The correlation-dimension is computed as the asymptotic slope of the $d$-curves in the scaling region in the limit $d \to \infty$.

### 29.4.2 Serial Computation of the Correlation Integral

The computational complexity of the serial algorithm for the computation of $K_2$ entropy stems primarily from the computation of $C_d$ (equation (29.4.1)). When computing the $K_2$ entropy with a maximum embedding dimension $D << N$ (where $N =$ number of time series data points), there are approximately $N \cdot (\frac{(N - 1)}{2})$ $d$-dimensional vector pairs (where $d \leq D$) to be considered. From equation (29.4.1), and the outline of the serial algorithm given in Figure 29.5, it can be shown that the computational complexity of the serial algorithm is $O(N^2 D^2)$. For large values of $N$, the computation on conventional single/serial processor systems becomes quite unwieldy, and the high response time severely limits the usefulness of the algorithm for many time-critical and real-life applications [5]. This is the primary motivation for considering a parallel algorithm on a PVM system of UNIX workstations.

### 29.5 A Parallel Algorithm for $K_2$ Entropy Computation using PVM

The PVM algorithm is designed by partitioning the outermost loop (with loop index $d$) of the serial algorithm (Figure 29.5) and having different processors iterate through the outermost loop for different ranges for parameter $d$. The master process spawns the child processes, broadcasts the time series data to all the child processes and awaits the results of the child processes. On receipt of the results of each of the child processes, the master process computes the correlation integral $C_d$ by assembling the CompareVec matrices received from each of the child processes. Each child process receives the time series data from the master process and computes its share of the outermost loop partition. On completion, it sends its CompareVec matrix to the master process. The master and slave processes are depicted in Figures 29.6 and 29.7, respectively.
for (d=1; d < D; d++)
beginfor

    /* Find the correlation-integral C_d(epsilon) by embedding 
       the time-series data points in the d-th dimension */

    for (n = 1; n < (N-2*d+1); n++)
beginfor
        for (m = n; m < (N-1); m++)
beginfor
            distance = 0;
            for (i = 0, i < (d-1); i++)
beginfor
                distance = distance + (X[n+i]-X[m+i]) * (X[n+i]-X[m+i]);
            endfor

            distance = sqrt(distance); /* Euclidean distance */

            i = 0;
            for (eps = 0; eps <= eps_final; eps = eps + eps_increment;
beginfor
                if (distance <= eps ) CompareVec[i] = CompareVec[i] + 2.0;
                i = i+1;
            endfor
        endfor
    endfor

    /* Print the values of correlation-integral C_d(epsilon) */

    for (i = 0; i <= eps_final/eps_increment; i++)
beginfor
        if (CompareVec[i] > 0) print(log(CompareVec[i]));
endfor

Figure 29.5 Serial algorithm for the computation of C_d.
master();

beginmaster

Phase 1: Initial Setup:
(a) Spawn Child Processes;
(b) Read Input: time series data;
(c) Compute the range \([d_{\text{min}}, d_{\text{max}}]\) for each child process;
\(/*\) The \([d_{\text{min}}, d_{\text{max}}]\) ranges for the child processes are non-overlapping */
(d) Broadcast time series data to all child processes;
(e) Transmit the \([d_{\text{min}}, d_{\text{max}}]\) range to each child process;

Phase 2: Collect Results:
(a) Receive CompareVec[][] matrix from each child process;
(b) Assemble the CompareVec[][] matrices in CompareVecSum[][];
(c) Print results:
for (d = 1; d < D; d++)
  for (i = 0; i <= eps_final/eps_increment; i++)
    if (CompareVecSum[d][i] > 0) print(log(CompareVecSum[d][i]));

endmaster

Figure 29.6 Master process for parallel computation of \(C_d\).

29.6 Optimal Scaling Region Determination Algorithm

The scaling region in a \(C_d\) plot is the one where all the \(d\)-curves are optimally parallel to each other [11]. Grassberger and Procaccia [11] present a manual procedure for determining the scaling region by visual inspection of the \(C_d\) plot. We present a serial exhaustive search algorithm for determination of the optimal scaling region in the \(C_d\) plot.

The objective function for the exhaustive search algorithm is formulated so as to have a minimum in the scaling region, where all the \(d\)-curves are optimally parallel to each other. The search algorithm proceeds by exhaustively constructing straight lines (with varying slopes and \(y\)-intercepts) that intersect the \(d\)-curves in the \(C_d\) plot. For each candidate line of intersection, the objective function value is computed as the summation of squared errors. Each error term in the summation is defined to be the deviation of the candidate line of intersection from the normal to that particular \(d\)-curve at the point of intersection. Note that the value of the objective function would generally be high in the depopulation region where all the \(d\)-curves have different slopes. However, in the saturation region, the objective function would have a very low value (generally zero), which is not desirable. In order to prevent the objective function from straying into the saturation region, the value of the objective function is normalized by dividing it by the local entropy. We define local
child()
beginchild

Phase 1: Initial Setup
(a) Receive time series data from master process;
(b) Receive \([d_{\min}, d_{\max}]\) value from child process;

Phase 2: Compute Correlation Integral in the range
for \(d = d_{\min}; d <= d_{\max}; d++\);
beginfor
for \((n = 1; n < (N-2*d+1); n++\);
beginfor
for \((m = n; m < (N-1); m++\);
beginfor
  distance = 0;
  for \((i = 0, i < (d-1); i++\);
  beginfor
    distance = distance + \((X[n+i]-X[m+i]) * (X[n+i]-X[m+i])\);
  endfor
  distance = sqrt(distance); /* Euclidean distance */
i = 0;
  for \((eps = 0; eps <= eps_{final}; eps = eps + increment)\);
  beginfor
    if \((distance <= eps)\) CompareVec[d][i] = CompareVec[d][i] + 2.0;
i = i+1;
  endfor
endfor
endfor

Phase 3: Transmit CompareVec[][] matrix to master process;

endchild

\textbf{Figure 29.7} Child process for parallel computation of \(C_d\)
entropy of the $C_d$ plot with respect to a straight line as the vertical distance between two successive $d$-curves with the highest possible $d$ values intersecting that straight line. It must be noted that there may exist some areas in the depopulation region where local entropy cannot be defined. This normalization procedure effectively penalizes the objective function (i.e., forces it to have a high value) in and around the saturation region. In the actual implementation, values of local entropy close to zero were replaced by a small positive constant to avoid overflow error. The normalized objective function is computed exhaustively for several candidate intersection lines characterized by the angle of the slope (measured in degrees) and the $y$-intercept. The global minimum on the objective function surface yields the angle of the slope and $y$-intercept of a line that is optimally orthogonal to all the $d$-curves and whose intersection points with the $d$-curves lie entirely within the scaling region. We define these intersection points to constitute the optimal scaling region.

29.7 Experimental Results

29.7.1 Chromosome Reconstruction

The PSA and PMCA algorithms were implemented on a PVM system comprised of a cluster of eight SUN SPARC5 UNIX workstations using C as the programming language. The various algorithms were run on a clone data set derived from Chromosome IV of the fungus *Aspergillus nidulans*, which was made available to us by Professor Jonathan Arnold, Department of Genetics, University of Georgia. The data set consisted of 592 clones with each clone having a 115 bit signature.

In order to obtain a fair comparison between the various PSA and PMCA algorithms, the product (denoted by $\lambda$) of the number of processors $n$ and the maximum number of iterations performed by a single processor at a given temperature or kinetic energy value (i.e., COUNT LIMIT) was kept constant. For example if one of the algorithms is run with 4 processors with COUNT LIMIT = 50,000 then with 2 processors, COUNT LIMIT would be 100,000 and with 1 processor COUNT LIMIT would be 200,000.

It was observed that the NILM PSA algorithm had a faster rate of convergence than the PILM PSA algorithm in terms of execution time. However, the PILM PSA algorithm was found to have a faster rate of convergence in terms of the number of iterations. This can be attributed to the fact that, although the PILM PSA algorithm needs fewer iterations than the NILM PSA algorithm, the average time per iteration in the case of the PILM PSA algorithm is greater. This is due to the overhead of interprocessor communication and synchronization entailed during the periodic interaction between the master process and the child processes in the case of the PILM PSA algorithm.

Table 29.1 shows the speed-up results for all the four algorithms: NILM PSA, PILM PSA, NILM PMCA and PILM PMCA for a varying number of processors $n$ and varying values of $\lambda$. For a given value of $n$, varying the value of $\lambda$ directly affects the the interprocessor communication overhead ratio $\tau = T_{\text{comm}} / T_{\text{cpu}}$ where
\( T_{\text{comm}} \) is the total time spent in interprocessor communication and \( T_{\text{cpu}} \) is the total time spent in computation. A higher value of \( \lambda \) for a given value of \( n \) implies a lower value for \( \tau \) and vice versa. For a linking distance equal to the optimal value of 550, the NILM PSA algorithm is seen to have a speed-up of 3.1 for \( n = 8 \) and \( \lambda = 200,000 \) and a speed-up of 6.1 for \( n = 8 \) and \( \lambda = 1,200,000 \). This result is expected since the interprocessor communication overhead ratio \( \tau \) is lower at higher values of \( \lambda \), resulting in a higher speed-up. The PILM PSA algorithm is seen to have a speed-up of 3.0 for \( n = 8 \) and \( \lambda = 200,000 \) and a speed-up of 4.7 for \( n = 8 \) and \( \lambda = 1,200,000 \). As expected, the PILM PSA algorithm has a lower speed-up than the NILM PSA algorithm for given values of \( n \) and \( \lambda \) due to its higher interprocessor communication overhead. As also expected, for a given value of \( n \) the speed-up of the PILM PSA algorithm shows an increasing trend with increasing \( \lambda \).

We observe that the parallelization of the MCA algorithm does not result in any speed-up (Table 29.1). On the contrary, there is a degradation in performance as the number of processors is increased. This phenomena can be explained by the fact that the serial MCA algorithm is considerably faster than the SA algorithm for the same number of iterations. For given values of \( n \) and \( \lambda \), the interprocessor communication overhead ratio \( \tau \) is far greater in the case of the PMCA algorithm as compared to the PSA algorithm. The interprocessor communication overhead dominates the performance of the PMCA algorithm for smaller values of \( \lambda \), resulting in a degradation in performance as the number of processors is increased. As expected, the NILM PMCA and PILM PMCA algorithms show an increasing trend in speed-up (just as their PSA counterparts) with increasing \( \lambda \) for a given value of \( n \). The NILM PMCA algorithm shows a speed-up of 0.4 for \( n = 8 \) and \( \lambda = 200,000 \) and a speed-up of 1.3 for \( n = 8 \) and \( \lambda = 1,200,000 \) (Table 29.1). The PILM PMCA algorithm, on the other hand, shows a speed-up of 0.6 for \( n = 8 \) and \( \lambda = 200,000 \) and a speed-up of 2.1 for \( n = 8 \) and \( \lambda = 1,200,000 \) (Table 29.1).

### 29.7.2 \( K_2 \) Entropy Computation

The correlation integral \( C_d \) was computed for the model Henon system (Figure 29.4) on the 8-node SUN SPARC5 workstation cluster using PVM. The runtime statistics of the parallel algorithm are tabulated in Table 29.2 for a varying number of processors and varying number of time series data points. As can be seen in Table 29.2, the speed-up is acceptable for all data sizes. For a given data size, the speed-up is slightly sublinear in the number of processors. For a given number of processors, the speed-up improves with increasing data size. Both these observations are consistent with our expectations since the communication overhead ratio \( \tau \) increases with increasing number of processors for a given data size and decreases with increasing data size for a given number of processors.

The values for the \( K_2 \) entropy and correlation-dimension of the model Henon system, computed using the parallel algorithm for the computation of \( C_d \) in conjunction with the serial algorithm for determination of the optimal scaling region, agrees well with the values reported by Grassberger and Procaccia [11], as shown in
Table 29.1 Speed-up Results for the PSA and PMCA Algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>NIIM PSA</th>
<th>PHLM PSA</th>
<th>NIIM PMCA</th>
<th>PHLM PMCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1.8</td>
<td>1.4</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>200,000</td>
<td>4</td>
<td>3.0</td>
<td>1.8</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>6</td>
<td>3.4</td>
<td>2.7</td>
<td>0.6</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>8</td>
<td>3.1</td>
<td>3.0</td>
<td>0.4</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.1</td>
<td>1.4</td>
<td>0.8</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>400,000</td>
<td>4</td>
<td>3.9</td>
<td>1.8</td>
<td>0.9</td>
<td>1.4</td>
</tr>
<tr>
<td>6</td>
<td>5.8</td>
<td>2.9</td>
<td>0.9</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>8</td>
<td>5.5</td>
<td>3.5</td>
<td>0.6</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.2</td>
<td>1.9</td>
<td>1.4</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>800,000</td>
<td>4</td>
<td>3.7</td>
<td>1.9</td>
<td>2.0</td>
<td>1.7</td>
</tr>
<tr>
<td>6</td>
<td>6.3</td>
<td>3.2</td>
<td>1.6</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>8</td>
<td>5.4</td>
<td>3.5</td>
<td>1.2</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1.8</td>
<td>1.5</td>
<td>1.6</td>
<td>1.7</td>
<td>1.7</td>
</tr>
<tr>
<td>1,000,000</td>
<td>4</td>
<td>4.5</td>
<td>2.0</td>
<td>1.9</td>
<td>1.7</td>
</tr>
<tr>
<td>6</td>
<td>5.5</td>
<td>3.3</td>
<td>1.5</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>8</td>
<td>5.5</td>
<td>4.7</td>
<td>1.6</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.3</td>
<td>1.5</td>
<td>1.4</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>1,200,000</td>
<td>4</td>
<td>4.7</td>
<td>1.9</td>
<td>1.8</td>
<td>1.1</td>
</tr>
<tr>
<td>6</td>
<td>6.7</td>
<td>4.7</td>
<td>2.1</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td>8</td>
<td>6.1</td>
<td>4.7</td>
<td>1.3</td>
<td>2.1</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Table 29.3. Since Grassberger and Procaccia obtained the values of the K2 entropy and correlation-dimension for the Henon system manually (i.e., by visual inspection of the C2 plot), a precise comparison based on numerical accuracy, between their results and ours, is not possible.

In most medical applications requiring off-line analysis of EKG data, a data set containing a bare minimum of 1800 heart rate points (i.e., 10 minutes of heart rate data at a sampling rate of 3 Hz) is needed. A data set of larger size would obviously improve the precision of the subsequent analysis. A turnaround time of one minute for the off-line analysis of 1800 heart rate data points would be considered acceptable - a capability that is currently provided by the 8-node PVM system. On-line analysis of EKG data (during cardiac surgery, for example) would require a turn around time in the millisecond range, which is 3–4 orders of magnitude
Table 29.2 Runtime Statistics of the Parallel Algorithm for Computation of the Correlation Integral $C_d$ of the Henon System. $T$: Execution Time (min), $\sigma$: Speed-up

<table>
<thead>
<tr>
<th>No. of Processors</th>
<th>2K $T$</th>
<th>2K $\sigma$</th>
<th>4K $T$</th>
<th>4K $\sigma$</th>
<th>8K $T$</th>
<th>8K $\sigma$</th>
<th>16K $T$</th>
<th>16K $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.21</td>
<td>1.00</td>
<td>9.13</td>
<td>1.00</td>
<td>27.39</td>
<td>1.00</td>
<td>248.74</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2.86</td>
<td>1.82</td>
<td>4.94</td>
<td>1.85</td>
<td>14.30</td>
<td>1.91</td>
<td>129.22</td>
<td>1.94</td>
</tr>
<tr>
<td>4</td>
<td>1.52</td>
<td>3.41</td>
<td>2.64</td>
<td>3.46</td>
<td>7.83</td>
<td>3.49</td>
<td>70.07</td>
<td>3.55</td>
</tr>
<tr>
<td>8</td>
<td>1.01</td>
<td>5.16</td>
<td>1.75</td>
<td>5.29</td>
<td>5.16</td>
<td>5.29</td>
<td>46.67</td>
<td>5.33</td>
</tr>
<tr>
<td>8</td>
<td>0.75</td>
<td>6.95</td>
<td>1.30</td>
<td>7.02</td>
<td>3.82</td>
<td>7.15</td>
<td>34.26</td>
<td>7.26</td>
</tr>
</tbody>
</table>

Table 29.3 Comparison of Results for the Model Henon System. $K_2$: $K_2$ Entropy. CD: Correlation Dimension

<table>
<thead>
<tr>
<th></th>
<th>$K_2$</th>
<th>CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grassberger and Procaccia</td>
<td>0.325</td>
<td>1.22</td>
</tr>
<tr>
<td>PVM Implementation</td>
<td>0.334</td>
<td>1.24</td>
</tr>
</tbody>
</table>

smaller than that of our current PVM implementation. We intend to address this issue in our future research.

29.8 Conclusions

In this chapter, we presented PVM-based parallel algorithms for two very important biomedical problems: chromosome reconstruction and heart-rate variability analysis. We designed and analyzed two models for a parallel simulated annealing (PSA) algorithm and two models for a parallel microcanonical annealing (PMCA) algorithm in the context of chromosome reconstruction via clone ordering. These models were based on the decomposition of the Markov chain of solution states across multiple processors and were termed as the Periodically Interacting Local Markov chain (PILM) model and the Non-Interacting Local Markov chain (NILM) model. The algorithms were implemented on a PVM system comprised of eight SUN SPARC5 workstations running UNIX.

Between the NILM PSA and PILM PSA models, the former showed faster convergence to the globally optimal solution and higher speed-up as well. This could be attributed to the higher interprocessor communication overhead associated with the PILM PSA model. Both, the PILM and the NILM PMCA models showed very little improvement in terms of speed of convergence and speed-up over the serial
MCA algorithm. This can be attributed to the fact that the PMCA algorithm has a higher interprocessor communication overhead ratio $\tau$, compared to the PSA algorithm. In other words, since the serial MCA algorithm is inherently fast and computationally efficient, the performance of the PMCA algorithm was dominated by the interprocessor communication overhead, leading to an overall performance degradation as the number of processors was increased. This was further corroborated by our observation that decreasing the value of $\tau$ (by increasing the number of iterations of the PMCA algorithm performed by each processor for a given kinetic energy value) improved the speed-up characteristics of the NILM and PILM PMCA algorithms.

In this chapter, we also presented a PVM-based parallel algorithm for heart rate variability analysis. We designed a parallel algorithm for the computation of the correlation integral and a serial exhaustive search-based algorithm for the determination of the optimal scaling region in the correlation integral plot. These algorithms were used to compute the $K_2$ entropy and the correlation dimension of experimental heart rate data. These algorithms were also verified on the model Henon system. The parallel algorithm for the computation of the correlation integral showed very good speed-up on the 8-node PVM system and could be readily applied to real life situations which call for timely analysis of chaotic time series data arising from an underlying complex dynamic system. The application of the PVM algorithm as an on-line prognostic/diagnostic tool for determining cardiovascular health needs to be further explored.

Acknowledgments

This research was partly funded by an NRCPGP grant by the US Department of Agriculture to Dr. Suchendra Bhandarkar.

29.9 Bibliography


