
ELSEVIER

Parallel Computing 24 (1998) 1177–1204

PARALLEL COMPUTING

Applications

Parallel computing for chromosome reconstruction via ordering of DNA sequences

Suchendra M. Bhandarkar [a,*], Salem Machaka [a], Sridhar Chirravuri [a], Jonathan Arnold [b]

[a] *Department of Computer Science, University of Georgia, 415 Boyd Graduate Studies Res. Cent., Athens, GA 30602-7404, USA*

[b] *Department of Genetics, University of Georgia, Athens, GA 30602-7223, USA*

Received 27 October 1996; received in revised form 12 January 1998

Abstract

In this paper we present our practical experience with the design and implementation of a suite of parallel algorithms called PARODS, for chromosome reconstruction via ordering of DNA sequences. PARODS contains parallel algorithms based on an earlier serial algorithm ODS which is a physical mapping algorithm based on simulated annealing. The parallel simulated annealing (PSA) algorithms for physical mapping in PARODS are based on Markov chain decomposition. We propose and describe perturbation methods and problem-specific annealing heuristics in the context of the physical mapping problem. We describe implementations of parallel Single Instruction Multiple Data (SIMD) algorithms on a 2048-processor MasPar MP-2 system and implementations of parallel Multiple Instruction Multiple Data (MIMD) algorithms on an 8-processor Intel iPSC/860 system and on a cluster of UNIX workstations using the Parallel Virtual Machine (PVM) system. We discuss and analyze the convergence, speedup and scalability characteristics of the aforementioned algorithms. We show the best SIMD algorithm to have a speedup of 1232 on the 2048 processor MasPar MP-2 system and the best MIMD algorithm to have a speedup of 5.35 on the 8-processor Intel iPSC/860 system and 3.40 on a 6-workstation cluster running PVM.

Keywords: Clone ordering; DNA sequencing; Chromosome reconstruction; Simulated annealing; Parallel processing

* Corresponding author. E-mail: suchi@cs.uga.edu

0167-8191/98/$ – see front matter
PII: S0167-8191(98)00053-2

1. Introduction

Chromosome reconstruction i.e., the generation of entire chromosomal maps, has been a central problem in the field of genetics right from its very inception [47]. These maps, which could then be used to reconstruct the chromosome's DNA sequence, are central to the understanding of the structure of genes, their function, their transmission and their evolution. The advent of molecular genetics has led to a wealth of DNA markers along a chromosome, making feasible the mapping of entire genomes such as that of man [4,11]. The large number of DNA markers and the ease with which molecular markers are assayed has shifted the focus of current research from the collection of mapping data to the computational problem of assembling the maps. In fact, creating and assembling physical maps for each of the human chromosomes and those of several model systems is the central goal of the Human Genome Project [14].

Chromosomal maps fall into two broad categories – *genetic maps* and *physical maps*. Genetic maps represent an ordering of genetic markers along a chromosome where the distance between two genetic markers is inversely proportional to their recombination frequency. Genetic maps are typically of low resolution i.e., 1–10 million base pairs (Mb). Lander and Green [30] pioneered the use of computational techniques for the assembly of genetic maps with many markers. Their work was of both, conceptual and practical significance and aided scientists in hunting down genes of biological, medical and statistical interest. While genetic maps enable a scientist to 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 an ordering of distinguishable DNA fragments or *clones* by their position along the entire chromosome where the clones may or may not contain genetic markers [10,41,46,51,55]. A physical map has a much higher resolution than a genetic map of the same chromosome i.e., 10–100 thousand base pairs (Kb). Examples of physical maps include cytological maps, radiation-hybrid maps, ordered clonal libraries (i.e., *contig maps*) and ultimately a chromosome's entire DNA sequence. Physical maps have provided fundamental insights into gene development, gene organization, chromosome structure, recombination and the role of sex in evolution and have also provided a means for the recovery and molecular manipulation of genes of interest. A remarkable feature about the physical mapping problem is that it is inextricably intertwined with a classical problem in computer science, namely, determining an optimal linear arrangement [19]. Addressing the aforementioned fundamental biological problems by means of physical maps first involves solving the challenging and fundamental computational problem of determining an optimal linear arrangement.

The specific technique, discussed in this paper, for generating a physical map is one based on determining an ordering of clones from a library that optimizes a prespecified objective function [15,16]. The optimal ordering of clones with respect to their position along a chromosome is then deemed to constitute a physical map. This technique for physical map assembly has proven adaptable to a wide variety of

experimental protocols such as chromosome walking [13,45], contig mapping utilizing classic restriction enzyme digestion profiles as fingerprints [12,39], contig mapping utilizing fingerprinting by oligonucleotide hybridization [23], and sequence-tagged-site (STS) content mapping [18,21,36,40,43].

1.1. The physical mapping problem

In our previous work [15,16] we presented a physical mapping algorithm ODS (Ordering DNA Sequences) based on simulated annealing. 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

$$D = \sum_{i=1}^{n-1} d(C_i, C_{i+1}). \tag{1}$$

(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

$$\lim_{n\to\infty} \mathrm{prob}(|D_{\mathrm{m}} - D_0| > \epsilon) = 0. \tag{2}$$

That is to say, D_{m} converges in probability to D_0 as the size n of the clonal library grows [54]. This result provides the formal basis for the physical mapping algorithm ODS. Fig. 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 [19]. 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 [8]. Stochastic optimization algorithms such as simulated annealing [20,29] 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. In fact, ODS which uses simulated annealing, proved very successful in generating high quality physical maps from DNA/DNA hybridization data [15,16]. One of the drawbacks of a serial implementation of simulated annealing is that the annealing schedules necessary to

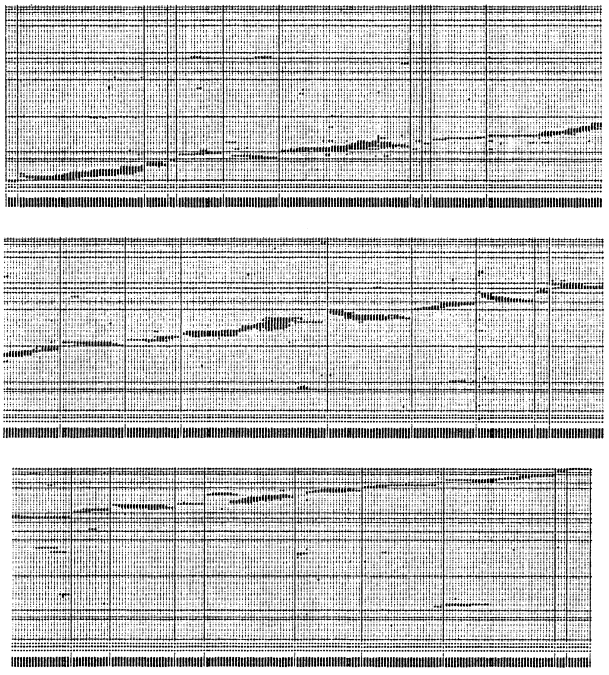

Fig. 1. An ordered physical map of *Aspergillus Nidulans* Chromosome IV.

obtain a solution close to a global optimum, are computationally intensive. Parallel processing is one of the ways in which this drawback can be alleviated. In fact, parallel processing has emerged only very recently in the context of problems in computational biology, such as sequence comparison [24,28,33,37], sequence alignment [17,26], genetic mapping [38] and physical mapping [5,6].

1.2. Simulated annealing basics

The serial simulated annealing algorithm (Fig. 2) consists of three phases. *Perturb phase:* Given an objective function and a candidate solution x_i, the candidate solution x_i is systematically perturbed to yield another candidate solution x_j. In our case, the clone ordering is systematically permuted by swapping clone positions or by reversing the ordering within a block of clones.

Evaluate phase: The new candidate solution x_j is evaluated i.e., $f(x_j)$ is computed. In our case, the total linking distance associated with the new clone ordering is computed.

Decide phase: If $f(x_j) < f(x_i)$ then x_j is accepted as the new candidate solution. If $f(x_j) \geqslant f(x_i)$ then x_j is accepted as the new candidate solution with probability p the value of which is computed using the Metropolis function [35]

```
const float T_min, T_max;
const int COUNT_LIMIT;
int count;
float T;
T = T_max;
while (T >= T_min)
{
  for (count=1; count <= COUNT_LIMIT; count = count + 1)
  {
    1. Phase One - Perturb
       (a) Randomly perturb the existing solution to generate a
           new candidate solution;
    2. Phase Two - Evaluate
       (a) Compute the cost associated with the new candidate solution;
       (b) Compute f_delta, the change in cost function that would
           result if the new candidate solution were to replace the
           existing solution;
    3. Phase Three - Decide
       (a) Accept the new candidate solution if it causes the cost
           function to decrease;
       (b) accept the new candidate solution with probability
           P_T(f_delta) computed using the Metropolis function if
           it causes the cost function to increase;
  }
  Update the temperature using the annealing schedule T = A(T);
}
```

Fig. 2. Outline of a serial simulated annealing algorithm.

$$p = \exp\left(-\frac{f(x_j) - f(x_i)}{T}\right), \tag{3}$$

for a given value of temperature T whereas x_i is retained with probability $(1-p)$. The temperature T is systematically reduced using an annealing schedule.

As can be deduced from the Decide phase and Eq. (3), at sufficiently high temperatures, simulated annealing resembles a random search whereas at lower temperatures it acquires the characteristics of the conventional local hill-climbing search. Annealing schedules that guarantee asymptotic convergence to a global minimum are extremely lengthy [20] thus making simulated annealing a computationally intensive procedure in spite of its robustness to the presence of local minima in the objective function. Parallelization of simulated annealing is therefore imperative if one desires high quality solutions to complex multivariate optimization problems, such as reconstruction of complex chromosomes via clone ordering, in a reasonable time frame.

1.3. Parallel simulated annealing

Parallelization of simulated annealing has been attempted by several researchers especially in the area of computer-aided design, image processing and operations research. Parallel simulated annealing (PSA) algorithms have been implemented on a variety of multiprocessor platforms – Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD), shared memory and distributed

memory. Several parallelization strategies for simulated annealing have been proposed in the literature. As described below, these parallelization strategies are largely based on how the *perturb-evaluate-decide* cycle of the serial algorithm is parallelized on a given multiprocessor platform.

(A) *Functional parallelism* within a perturbation where the evaluation function is decomposed into subfunctions that can be evaluated in parallel by multiple processors [53]. This approach however, is very specific to the objective function under consideration.

(B) *Control parallelism* with

(i) *Speculative computation* where the *move-evaluate-decide* iteration cycle is so decomposed among multiple processors such that several iteration cycles are active at any time [52]. A good CPU load-balancing scheme is crucial for this approach to yield appreciable speedup [52].

(ii) *Parallel Markov chain generation* using a *systolic* algorithm [1,2,22]. This approach exploits the fact that the simulated annealing algorithm at a given temperature generates an asymptotically ergodic (and hence stationary) Markov chain of solution states [20]. However, parallel Markov chain generation using a systolic algorithm requires each processor to know the global problem state thus limiting its scalability.

(iii) *Multiple independent searches* or *periodically interacting searches* [2,31]. This approach though intuitive and simple to implement also suffers from lack of scalability since each processor needs to know the global problem state.

(C) *Data parallelism* with

(i) *Parallel evaluation of multiple perturbations with acceptance of a single perturbation* [9]. The convergence characteristics of the parallel algorithm in this approach are similar to those of the sequential algorithm however, since many good moves are wasted, the speedup is limited.

(ii) *Parallel evaluation of multiple perturbations with acceptance of non-interacting multiple perturbations* [27]. The convergence characteristics of the parallel algorithm in this approach are also similar to those of the serial algorithm. However, the speedup is limited by the overhead of having to determine a maximal set of non-interacting perturbations.

(iii) *Parallel evaluation and acceptance of multiple perturbations* [3]. This approach allows for maximum parallelism and scalability but the convergence characteristics of the parallel algorithm may be drastically different from those of the serial algorithm, thereby leading to unexpected final results. This problem can be alleviated by tracking the error and synchronizing the processors from time to time [3].

Of the aforementioned parallelization strategies, we deemed the ones based on multiple searches (i.e., B(iii)) and parallel evaluation and acceptance of multiple perturbations (i.e., C(iii)) to be the most promising from the viewpoint of parallelizing ODS. The combination of multiple searches and parallel evaluation and acceptance of multiple perturbations was seen to have the potential to offer maximum speedup and scalability with ease and simplicity of implementation. The resulting suite of parallel SIMD/MIMD algorithms, based on simulated annealing, for

chromosome reconstruction via clone ordering is termed as PARallel Algorithms for Ordering DNA Sequences (PARODS).

2. Multiprocessor platforms for PARODS

The three multiprocessor platforms that were considered for the implementation of PARODS are (i) the MasPar MP-2; (ii) the Intel iPSC/860, and (iii) a cluster of workstations running PVM, all of which reside in the Department of Computer Science at the University of Georgia.

The MasPar MP-2 is a massively parallel single-instruction-multiple-data (SIMD) computer with the processing elements (PE's) interconnected in a toroidal 2D grid or mesh topology. In order to parallelize the program ODS [16] on the MasPar MP-2, it was redesigned and recoded in the MPL programming language [34]. Being an SIMD architecture, all the PE's in the MasPar MP-2 share the same instruction stream but operate on potentially different local data. The MPL language is an extension of ANSI C with constructs that allow SIMD programming. SIMD parallelism in MPL stems from operations on *plural* variables i.e., variables that are replicated synchronously (with possibly different values) in the local memory of each PE. The MasPar MP-2 architecture permits 8-nearest-neighbors communication on the 2-D toroidal grid as well as point-to-point communication between any two PE's using a global router.

The Intel iPSC/860 is a distributed-memory multiple-instruction-multiple-data (MIMD) multiprocessor system based on the message-passing paradigm. The individual PE's in the Intel iPSC/860 are connected in a Boolean hypercube topology [42]. In order to implement the program ODS [16] on the Intel iPSC/860, the data decomposition and process decomposition strategies were designed at an abstract level. The individual processes and data structures were then implemented in a high level programming language (C in our case). The C language for the Intel iPSC/860 is extended with special parallel/distributed processing primitives for inter-node communication, host-node communication, data/process decomposition and process synchronization [25].

The Parallel Virtual Machine (PVM) [48] is a software environment designed to exploit parallel/distributed computing across a variety of computer types. PVM makes a collection of heterogeneous computers appear as a single large virtual parallel machine that is based on a distributed-memory, message-passing paradigm. From a programming standpoint, PVM has the same functionality as a distributed-memory, message-passing MIMD architecture such as the Intel iPSC/860. The PVM system is composed of two main modules. The first module is a daemon, called **pvmd**, that resides on all the computers comprising the PVM system. The second module is a library of PVM interface routines that contain a set of primitives needed for message-passing, spawning processes, coordinating/synchronizing various tasks, broadcasting and collecting data, and modifying the PVM system at run-time. The PVM system supports high level programming languages such as C. The PVM system used in our work comprised of a 6-node cluster with a token ring

interconnection network topology where each node PE was a SUN SPARC5 workstation.

3. The parallel algorithms in PARODS

A candidate solution in the serial simulated annealing algorithm can be considered to be an element of an asymptotically ergodic first-order Markov chain of solution states [20]. Consequently, we have formulated and implemented various models of PSA based on the distribution of the Markov chain of solution states on the individual PE's of the MasPar MP-2, the Intel iPSC/860 and the PVM system and on the synchronization method used. These models incorporate the parallelization strategies B(iii) and C(iii) discussed in Section 1.3.

For the MasPar MP-2 we have formulated and implemented four models of a massively parallel simulated annealing (MPSA) algorithm:

(a) Non-Interacting Local Markov chain (NILM) MPSA algorithm.
(b) Periodically Interacting Local Markov chain (PILM) MPSA algorithm.
(c) Periodically Interacting Distributed Markov chain (PIDM) MPSA algorithm.
(d) Non-Interacting Distributed Markov chain (NIDM) MPSA algorithm.

For the Intel iPSC/860 and PVM system, we have formulated and implemented three models of a PSA algorithm:

(a) NILM PSA algorithm.
(b) PILM PSA algorithm.
(c) Distributed Markov chain (DM) PSA algorithm.

In the NILM MPSA algorithm on the MasPar MP-2, and the NILM PSA algorithm on the Intel iPSC/860 and the PVM system, each PE runs an independent version of the serial simulated annealing algorithm. In essence there are as many Markov chains of solution states as there are PE's in the multiprocessor architecture. Each Markov chain is local to a given PE, and at any instant of time each PE maintains a candidate solution which is an element of its local Markov chain. The serial simulated annealing algorithm is run synchronously on each PE of the MasPar MP-2 and asynchronously on each PE of the Intel iPSC/860 or PVM system.

The choice of the perturbation strategy and the efficiency with which it enables the change in the objective function (i.e., linking distance) to be computed are critical to the performance of the simulated annealing algorithm. Consider a multivariate objective function $f(x)$ to be minimized with a solution space Ω. Let Ω_i denote the set of all possible solutions x_j that can be generated from a solution x_i using a set of permitted perturbations $\mathscr{P}$. The solutions $x_j \in \Omega_i$ are said to the neighbors of solution x_i and Ω_i the neighborhood of solution x_i under the perturbation set $\mathscr{P}$. The pair of sets Ω and $\mathscr{P}$ can be represented with a directed graph $\mathscr{G}$ where each vertex in $\mathscr{G}$ corresponds to a solution in Ω and there exists a directed edge from vertex i to vertex j iff $x_j \in \Omega_i$. A necessary condition for the convergence of the simulated annealing algorithm is that the perturbation set $\mathscr{P}$ should ensure that $\mathscr{G}$ is *strongly connected* i.e., there should exist a directed path between any pair of vertices i and j in $\mathscr{G}$. In other words, it should be possible for the simulated annealing algorithm to visit any

solution in Ω starting from any other solution using a finite sequence of perturbations from $\mathscr{P}$ [44]. If $p_{i,j}(T)$ is the transitional probability of visiting solution x_j from solution x_i at a temperature value T, then the aforementioned necessary condition could also be stated as

$$p_{i,j}(T) > 0, \qquad \forall i, j \in \Omega. \tag{4}$$

The two perturbation strategies that satisfy the aforementioned necessary condition [32] were implemented and tested for the NILM MPSA and the NILM PSA algorithms:

Pairwise Clone Exchange: In a given permutation (i.e., ordering) σ_1: $(C_1, C_2, \ldots, C_n)$, the positions of two randomly chosen clones C_i and C_j are interchanged to generate a new permutation σ_2: $(C_1, \ldots, C_{i-1}, C_j, C_{i+1}, \ldots, C_{j-1}, C_i, C_{j+1}, \ldots, C_n)$.

Clone Block Reversal: Let $(C_1, C_2, \ldots, C_n)$ be the current clone ordering σ_1. Two clones C_i and C_j (where $i < j$ and $i, j \leqslant n$) are chosen at random, and the clone ordering in the block between C_i and C_j is reversed resulting in the clone ordering σ_2: $(C_1, \ldots, C_{i-1}, C_j, C_{j-1}, \ldots, C_{i+1}, C_i, C_{j+1}, \ldots, C_n)$.

The perturbation function uses a parallel random number generator with distinct seeds for each PE in order to ensure the independence of the resulting Markov chains of solution states. The NILM MPSA and the NILM PSA models are essentially multiple independent searches of the solution space.

In the PILM MPSA algorithm on the MasPar MP-2 and the PILM PSA algorithm on the Intel iPSC/860 and PVM system, at each temperature value T, the *perturb-evaluate-accept* cycle of the serial simulated annealing algorithm (Fig. 2) is executed concurrently in all the PE's COUNT_LIMIT number of times just as in the case of the NILM MPSA or the NILM PSA algorithm. However, just before the parameter T is updated using the annealing schedule, the best candidate solution from among those in all the PE's is selected and duplicated on all the other PE's. The PILM MPSA and PILM PSA models are essentially multiple periodically interacting searches.

In the case of the NIDM MPSA algorithm for the Maspar MP-2, the candidate solution (and hence a single Markov chain) is distributed over a group of neighboring processors i.e., over a PE cluster. Since the PE array in the Maspar MP-2 has a 2-D mesh topology, it is convenient to have the PE clusters in the form of non-overlapping rectangular submeshes of equal size (Fig. 3). The dimensions of the PE cluster are denoted by X_{width} and Y_{width}. Since the dimensions of the PE array in the MasPar MP-2 are typically powers of 2, so are X_{width} and Y_{width}. The perturbations in the NIDM MPSA algorithm are carried out in three distinct phases.

(A) Intra-PE Perturbations: All the PE's concurrently perform the *perturb-evaluate-accept* cycle on their individual portions of the candidate solution as in the case of the NILM MPSA model. The change in linking distance ΔD is computed *locally* within each PE, and the evaluation and acceptance phases are executed concurrently within each PE. Note that the evaluation of the Metropolis function requires only the value of ΔD, not the absolute value of the linking distance D. Also note that both types of intra-PE perturbations (i.e., pairwise clone exchange and clone block reversal) are characterized by two clone positions. In the case of pairwise clone

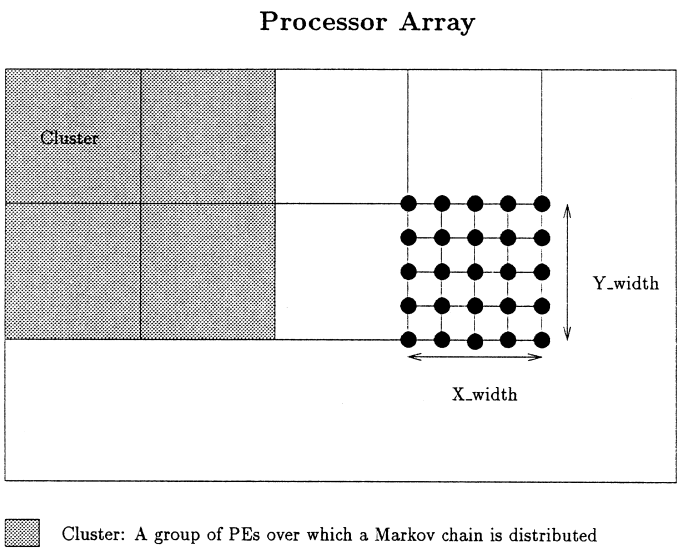

Fig. 3. Distribution of Markov chains over a PE cluster in the MasPar MP-2 PE array.

exchange the two clone positions denote the clones to be exchanged whereas in the case of clone block reversal, they denote the end points of the clone block to be reversed. When both the clone positions associated with an intra-PE perturbation lie *strictly within* the block of clones local to the PE, then the intra-PE perturbations are *non-interacting* given the fact that the clones in each PE are distinct. The term "non-interacting" denotes the fact that the value of ΔD computed in one PE does not affect nor is it affected by the value of ΔD computed in any other PE. However when either one or both of the clone positions associated with an intra-PE perturbation lie(s) on the boundary of the clone block associated the PE, then the intra-PE perturbation can no longer be considered to be non-interacting. In this case, error in the computation of ΔD is permitted, and the intra-PE perturbation and those that interact with it are based on an erroneous evaluation of the Metropolis function.

If the clone positions for an intra-PE perturbation are chosen by sampling a uniform distribution then the probability p of having non-interacting intra-PE perturbations between k processors is given by

$$p = \prod_{i=1}^{k} \left(1 - \frac{2}{n_i}\right)^2, \tag{5}$$

where n_i is the number of clones in the ith PE. If $n_i = n$ for all PE's then

$$p = \left(1 - \frac{2}{n}\right)^{2k}. \tag{6}$$

As can be seen, $p \to 1$ in the limit as $n \to \infty$. Also, for a fixed value of n, $p \to 0$ in the limit as $k \to \infty$. Thus, the probability of having strictly non-interacting intra-PE

perturbations decreases exponentially as a clonal data set of a fixed size is distributed over an increasing number of PE's. We have observed that with values of $p = 0.6$ (i.e. with 40% interacting or erroneous perturbations permitted) the asymptotic convergence of the PSA algorithm is not adversely affected in practice [6]. This observation and Eq. (6) place an upper limit on the number of PE's over which the clonal data of a given size can be distributed.

(B) Inter-PE Perturbations along the Cluster Rows: The PE's are paired along the rows in each PE cluster. One of the PE's is denoted as the *master* and the other as the *slave*. Two types of inter-PE perturbations are considered: *single clone exchange* and *clone block exchange*. The master PE initiates the perturbation by sending a randomly chosen single clone or block of clones from its local clone ordering to the slave PE as depicted in Fig. 4. The slave PE responds by sending a randomly chosen single clone or block of clones (of the same size) from its local clone ordering to the master PE. The master PE and slave PE concurrently evaluate the change ΔD in their respective local clone orderings that would result from the exchange (or replacement). The two values of ΔD are summed in both, the master PE and the slave PE. The master PE decides whether or not to accept the proposed perturbation using the Metropolis function. If the perturbation is accepted then the master and slave PE's update their respective local clone orderings. As in the case of intra-PE perturbations, the inter-PE perturbations between disjoint PE pairs are non-interacting if and only if both the clones or clone blocks to exchanged lie strictly within the respective clone blocks assigned to the master or slave PE. If the above condition is not satisfied then the inter-PE perturbations are no longer non-interacting and some perturbations based on improper evaluation of the Metropolis function are permitted.

(C) Inter-PE Perturbations along the Cluster Columns: This phase is identical to phase (B) except that the PE's are paired along the columns in each PE cluster.

Since each PE cluster in the MasPar MP-2 runs an independent version of the simulated annealing algorithm, the NIDM MPSA model is a combination of multiple independent searches and parallel evaluation and acceptance of multiple moves. The PIDM MPSA algorithm is similar to the NIDM MPSA algorithm except that before each update of the temperature parameter T, the best candidate solution is picked from among the available solutions in the various PE clusters and duplicated in all the PE clusters. The PIDM MPSA model is a combination of multiple

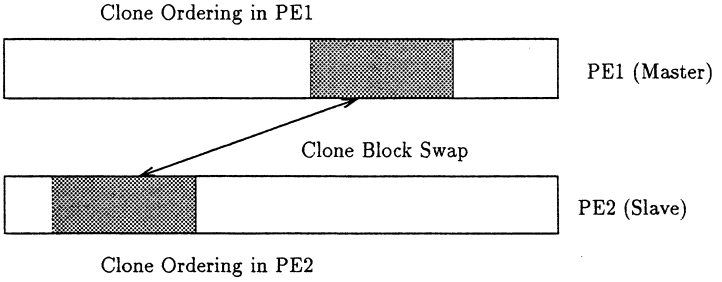

Fig. 4. Inter PE perturbation.

periodically interacting searches and parallel evaluation and acceptance of multiple moves. The control structure of the PIDM MPSA algorithm, outlined in Fig. 5, is the most general of the four MPSA models for the MasPar MP-2 described in this paper. Deletion of Phase 5 in Fig. 5 would result in the NIDM MPSA algorithm. The NILM MPSA algorithm and the PILM MPSA algorithm are special cases of the NIDM MPSA algorithm and the PIDM MPSA algorithm respectively wherein the PE cluster reduces to a single PE i.e., $X_{\text{width}} = 1$ and $Y_{\text{width}} = 1$.

In the DM PSA algorithm on the Intel iPSC/860 or the PVM system, the candidate solution, is distributed over the PE's in the hypercube or the token ring such that each PE has knowledge only about its own local clone ordering. The perturbations in the DM PSA algorithm are carried out in 2 distinct phases:

```
const float T_min, T_max;
const int COUNT_LIMIT;
int count;
float T;

T = T_max;
while (T >= T_min)
{
  for (count=1; count <= COUNT_LIMIT;  count = count + 1)
  {
   Phase 1: (a) Intra-Node Perturbation: Randomly perturb the existing (sub)solution
                within each PE concurrently;
            (b) Intra-Node Evaluation: Compute the change in cost f_delta within each PE
                concurrently;
            (c) Intra-Node Acceptance: Concurrently accept the perturbed candidate
                solutions in each PE with probability P_T(f_delta) computed using
                the Metropolis function;

   Phase 2: (a) Inter-Node Perturbations along the PE Cluster Rows: Swap clone blocks
                between disjoint PE pairs along the cluster rows concurrently;
            (b) Inter-node Evaluation along PE Cluster Rows: Compute the change in cost
                f_delta within master PE of each PE pair concurrently;
            (c) Inter-Node Acceptance along PE Cluster Rows: Concurrently accept the
                perturbed candidate solutions in each PE pair with probability P_T(f_delta)
                computed using the Metropolis function;

   Phase 3: (a) Inter-Node Perturbations along the PE Cluster Columns: Swap clone blocks
                between disjoint PE pairs along the cluster rows concurrently;
            (b) Inter-node Evaluation along PE Cluster Columns: Compute the change in cost
                f_delta within master PE of each PE pair concurrently;
            (c) Inter-Node Acceptance along PE Cluster Columns: Concurrently accept the
                perturbed candidate solutions in each PE pair with probability P_T(f_delta)
                computed using the Metropolis function;
   }

  Phase 4. Determine the best candidate solution based on its total cost at the
           current temperature, using a parallel reduction mechanism;

  Phase 5. Duplicate the best candidate solution determined in Phase 4
           amongst all the PE clusters;

  Phase 6. Update the temperature using the annealing schedule: T = A(T);
}
```

Fig. 5. The PIDM MPSA algorithm on the MasPar MP-2.

(A) Intra-PE Perturbations: All the PE's concurrently perform the *perturb-evaluate-accept* cycle on their individual portions (i.e., subsolutions) of the candidate solution as in the case of the NILM PSA model. The discussion pertaining to the non-interacting nature of the intra-PE perturbations in the case of the DM PSA algorithm is the same as that for the intra-PE perturbations in the PIDM MPSA and NIDM MPSA algorithms described previously.

(B) Inter-PE Perturbations: In the case of the Intel iPSC/860, the PE's are paired along the inter-PE links in each dimension of the hypercube (Fig. 6 (a)). In a hypercube with $N = 2^n$ PE's, there are $2^{n-1} = N/2$ disjoint PE pairs in each of the n dimensions such that the PE's that constitute a single pair are directly connected by an inter-PE link in that dimension. In an inter-PE perturbation, the PE's within a PE pair evaluate the resulting change in the cost function. In each PE pair, one of the PE's is designated as the *master* PE and the other as the *slave* PE. The master-slave assignment along the ith dimensional link is done based on the value of the ith bit in the PE address.

The master PE initiates an inter-PE perturbation by randomly selecting a clone or a block of clones from its local ordering and sending it along with its neighboring clones to the slave PE. The slave PE on receipt of the clone or block of clones from the master PE randomly selects a clone or a block of clones (of the same size) respectively from its local ordering. The slave PE then evaluates the change in cost function ΔD that would result if the corresponding clones or block of clones were to

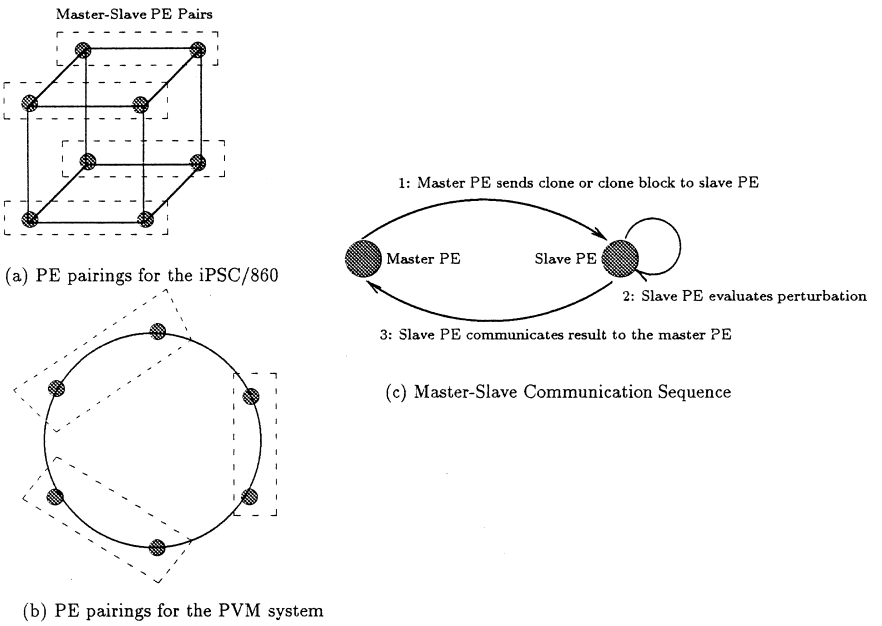

Fig. 6. PE pairing and message passing sequence between master PE and slave PE on the Intel iPSC/860 and the PVM system.

be swapped. Using the result of the evaluation and the Metropolis function the slave PE decides whether or not the swap should be carried out. If the slave PE decides to swap, then it informs the master PE of its decision and sends to the master PE the value of ΔD and the corresponding clone or block of clones from its local ordering with which the master PE is required to swap its own clone or block of clones respectively. If the slave PE decides not to swap then it simply informs the master PE of its decision. In the event of a swap, both the master PE and the slave PE update their respective local clone orderings and the cost function by an amount ΔD. The message-passing sequence between master PE and slave PE is depicted in Fig. 6 (c).

Since there are $2^{n-1} = N/2$ disjoint PE pairs in each dimension and the clones are in each PE pair are unique, $2^{n-1} = N/2$ inter-PE perturbations can carried out in parallel along each dimension of the hypercube. The discussion regarding the non-interacting nature of the inter-PE perturbations in the case of the DM PSA algorithm is the same as that for the PIDM MPSA and NIDM MPSA algorithms discussed earlier. The inter-PE perturbation phase proceeds sequentially along the dimensions of the hypercube i.e., from one dimension of the hypercube to the next. The PE's in the hypercube represent a distributed Markov chain of solution states. On termination of the program, each PE transmits its local clone ordering to the host. The host assembles the local clone orderings from the individual PE's to construct the overall clone ordering. The DM PSA algorithm is based on the parallel evaluation and acceptance of multiple moves.

The inter-PE perturbations in the case of the DM PSA algorithm on the PVM system are very similar to those in the case of the DM PSA algorithm on the Intel iPSC/860. The PE's in the token ring are paired as $\{p, (p+1) \bmod N\}$ and as $\{p, (p-1) \bmod N\}$ in alternating iterations where p is an even-valued PE address and N is the total number of PE's (Fig. 6 (b)). Thus, there are $N/2$ disjoint PE pairs at every iteration and the inter-PE perturbation phase alternates between the two possible pairings. The master-slave assignment is done based on the even/odd nature of the PE addresses in each PE pair.

Figs. 7–10 depict the PSA algorithm for the Intel iPSC/860. In Fig. 7 the process that runs on the host computer is outlined. In Fig. 7, Phase 3:Periodic Synchronization is carried out only in the case of the PILM PSA algorithm. Fig. 8 outlines the process that runs on each PE. Figs. 9 and 10 describe the intra-PE perturbations and inter-PE perturbations, respectively. The inter-PE perturbations would be carried out only in the case of the DM PSA algorithm. Typically, the total number of inter-PE perturbations is a fraction (denoted by the variable FACTOR in Fig. 10) of the total number of intra-PE perturbations. In our case, the value of FACTOR was in the range 1–5%.

In the case of the Intel iPSC/860 the host is a predesignated front-end computer for the Intel iPSC/860 system. In the case of the PVM system, one of the PE's in the cluster is designated as the host computer at run time. The host computer in the PVM system also serves as one of the node PE's (though not simultaneously). The PSA algorithms for the PVM system are very similar to their Intel iPSC/860 counterparts (except for the differences already mentioned) and are hence not repeated here.

```
const float T_min, T_max;
const int START_COUNT, COUNT_DELTA;
float T;
{
 Phase 1: Initial Setup
  (a) Read Input;
  (b) T =  T_max;
  (c) Compute Inter-Clone Distance Matrix;
  (d) Determine the number of node PE's in the system;

 Phase 2: Data and Process Decomposition
  (a) Load the node process on each node PE;
  (b) If the algorithm is NILM PSA or PILM PSA then
       (b.1) Load the entire clone data set on each node PE;
       (b.2) Load the entire inter-clone distance matrix on each node PE;
      Else if the algorithm is DM PSA
       (b.3) Partition the clone data set and interclone distance
             matrix into non-overlapping sets of almost equal size;
  (c) Send Annealing parameters T_min, T_max, START_COUNT, COUNT_DELTA
      to the node PE's;

  Phase 3: Periodic Synchronization
     while (T >= T_min)
     {
       Receive the clone ordering from each PE;
       Select the best clone ordering;
       Send the best clone ordering to each PE;
       Update the temperature using the annealing schedule: T = A(T);
     }

  Phase 4: Integration of Results
    (a) Receive the clone ordering from each PE;
    (b) If the algorithm is NILM PSA or PILM PSA then
          (b.1) Select the best clone ordering;
        Else if the algorithm is DM PSA
          (b.2) Assemble the final clone ordering from the partial
                clone orderings from each PE;
    (c) Output Result;
}
```

Fig. 7. The process that runs on the host computer of the Intel iPSC/860.

3.1. Annealing schedule

The algorithms discussed thus far implicitly assume a fixed-length annealing schedule where the total number of iterations of the *perturb-evaluate-decide* cycle are determined a priori. We have modified the annealing schedule to make it *adaptive* [44] in the following manner:

(a) The temperature is updated if the total number of perturbations at a given temperature equals the maximum limit COUNT_LIMIT or the total number of *successful* perturbations equals a predefined percentage (typically 0.1–0.2%) of COUNT_LIMIT. A perturbation is deemed successful if it reduces the cost function. In spite of the overhead of having to keep track of the successful perturbations, the adaptive annealing schedule is more efficient because at higher temperatures the total number of iterations at a given temperature are reduced to only 0.1–0.2% of COUNT_LIMIT thus speeding up the algorithm. At lower temperatures since there

```
const float T_min, T_max, f_delta, FACTOR;
const int COUNT_LIMIT;
float T;
int count;
{
  Phase 1: Initial Setup
   (a) Receive Clones From the Host;
   (b) Receive Inter-Clone Distance Matrix from the Host;
   (c) Receive annealing parameters T_min, T_max, START_COUNT, COUNT_DELTA
       from the Host;
   (d) Compute Dimension of Cube;

  Phase 2: Annealing Process
    T = T_max;
    while (T >= T_min)
    {
     (a) Intra-PE Perturbations;

     (b) Inter-PE Perturbations;

     (c) Periodic Synchronization:
              Send local clone ordering to Host;
              Wait;
              Receive clone ordering from Host;
              Update local clone ordering;

     (d) Update the temperature using the annealing schedule: T = A(T);
    }
  Phase 3: Result Integration: Send local ordering to Host;
}
```

Fig. 8. The process that runs on the node PE's of the Intel iPSC/860.

```
Intra-PE Perturbations:
 for (count=1; count <= COUNT_LIMIT; count=count+1)
  {
    (1) Randomly perturb the local clone ordering within each PE to
        concurrently generate multiple candidate subsolutions;
    (2) Evaluate Candidate Subsolution(s): Compute f_delta, the
        change in cost for each candidate subsolution concurrently
        in each PE using the pre-defined objective function f;
    (3) Parallel Accept: Concurrently accept the perturbed candidate
        subsolutions in each PE with probability P_T(f_delta) computed
        using the Metropolis function;
  }
```

Fig. 9. The intra-PE perturbations on the node PE's of the Intel iPSC/860.

are typically very few successful perturbations, the total number of iterations at a given temperature is closer to COUNT_LIMIT, thus increasing the chances of finding a globally optimal solution.

(b) The algorithm is halted if the same linking distance repeats for a certain number of successive temperature values (typically 3) in the annealing schedule. At this point, the algorithm is assumed to have reached a globally optimal solution. This prevents the annealing process from having to run (somewhat needlessly) until the

```
Inter-PE Perturbations:
Compute Dimension n of the hypercube;
for (count=1; count <= (FACTOR * COUNT_LIMIT); count=count+1)
 {
  for (i = 0; i < n; i=i+1)
   {
    Resolve Master-Slave Relationship with neighboring PE along
    the inter-PE link in dimension i;
    if (Master)
    {
     Randomly select clone/clone block from local clone ordering;
     Send clone/clone block to Slave;
     Wait;
     Receive decision from Slave;
     if (decision = swap) then
      {
       Receive clone/clone block from Slave;
       Update local clone ordering;
      }
    }
    if (Slave)
    {
      Receive clone/clone block from Master;
      Randomly select clone/clone block from local clone ordering;
      Evaluate change in cost function f_delta arising from swap;
      Accept the swap with probability P_T(f_delta) computed
      using the Metropolis function;
      Send decision to Master;
      if (decision = swap)
      {
       Send local clone/clone block to Master;
       Update local clone ordering;
      }
     }
    }
 }
```

Fig. 10. The inter-PE perturbations on the iPSC/860.

final temperature value is reached. The annealing schedule that was chosen was a geometric schedule [44] of the form $T_{\text{next}} = \beta T_{\text{prev}}$ where $\beta = 0.9$.

3.2. Further heuristic enhancements

Stochastic Synchronization: In the PILM MPSA and PIDM MPSA algorithms on the MasPar MP-2 and the PILM PSA algorithm on the Intel iPSC/860 and the PVM system, the best solution at the end of the iterations performed at each temperature value is duplicated on all the PE clusters. This synchronization is *deterministic*. Alternatively one can make the synchronization process stochastic by using the Boltzmann decision function wherein the best solution with linking D is duplicated on the ith PE/PE cluster containing solution with linking distance D_i with a probability P_{B}, where

$$P_{\text{B}} = \frac{1}{\left(1 + \exp\left(\frac{D-D_i}{T}\right)\right)}. \tag{7}$$

Determining a Maximal Set of Disjoint Successful Perturbations: At lower temperatures where fewer successful perturbations are likely, higher speedups can be gained by merging *disjoint* successful perturbations to yield a candidate solution that is capable of lowering the cost function to an extent greater than the individual perturbations themselves. The problem of finding a maximal set of disjoint successful perturbations can be shown to be isomorphic to the classical NP-complete *maximum independent set* problem [7,19]. A *greedy* algorithm for finding a maximal set of disjoint successful perturbations (in fact, the greedy algorithm is capable of finding only a suboptimal solution) was implemented and integrated with the PILM MPSA and PIDM MPSA algorithms on the MasPar MP-2 and the PILM PSA algorithm on the Intel iPSC/860 and the PVM system. The greedy algorithm has an order of complexity that is linear with respect to the number of perturbations.

4. Implementation issues and experimental results

All the aforementioned parallel algorithms were run on a DNA/DNA hybridization data set derived from Chromosome IV of the fungus *Aspergillus nidulans* [51]. The data set consisted of 592 clones with each clone having a 115 bit clonal signature. For further details on the biochemical experiments underlying the generation of the clonal data the interested reader is referred to some of our previous work [15,16,43,51]. Table 1 summarizes the results of the parallel algorithms on the MasPar MP-2, Intel iPSC/860 and the PVM system.

In the case of the MIMD PSA algorithms on the Intel iPSC/860 and the PVM system, the inter-clone distances were precomputed and stored on the PE's in the form of a lookup table or distance matrix. Due to the limited PE memory on the MasPar MP-2, *on-the-fly* computation of inter-clone distances was resorted to instead of a distance matrix look-up. It should be noted that the memory requirement of the distance matrix scales as $O(N^2)$ for N clones.

Table 1
Experimental results

System	N	Algorithm	T (s)	D	σ	η (%)
MasPar MP-2	2048	NILM MPSA	397.00	521	1188.58	58.04
		PILM MPSA	383.00	510	1232.03	60.16
		NIDM MPSA	587.00	612	803.84	39.25
		PIDM MPSA	502.00	648	939.00	45.85
Intel iPSC/860	8	NILM PSA	69.80	504	5.35	66.88
		PILM PSA	223.90	527	1.67	20.88
		DM PSA	304.60	673	1.23	15.38
PVM Cluster	6	NILM PSA	58.70	502	3.40	56.67
		PILM PSA	73.90	522	2.70	45.00
		DM PSA	175.07	681	1.14	19.00

(N = no. of processors, T = execution time in seconds, D = final linking distance, σ = speedup, η = efficiency of processor utilization)

In order to obtain a fair comparison between the various algorithms, the product (denoted by λ) of the number of PE's and the maximum number of iterations performed by a single PE at a given temperature was kept constant. For example, if an algorithm is run with N PE's with a maximum of M iterations per PE at any given temperature, then with $N/2$ PE's the maximum number of iterations per PE at any given temperature would be $2M$ and with 1 PE the maximum number of iterations per PE at any given temperature would be MN. In the case of the MPSA implementations on the MasPar MP-2, the value of λ was chosen to be $1,024,000$ whereas in the case of the PSA implementations on the Intel iPSC/860 and the PVM system, the value of λ was chosen to be $200,000$.

The merging of a maximal set of disjoint successful perturbations was invoked only when the number of successful perturbations fell below a prespecified percentage of the maximum number of iterations allowed at a given temperature, i.e. COUNT_LIMIT. In our case we chose this percentage to be 0.05% for implementations on both machines. The initial temperature value $T_{\max}$ was chosen to be 50 and the final temperature value $T_{\min}$ was chosen to be 0.1. The experimental results prompt us to make the following observations:

4.1. Speedup

On the Intel iPSC/860 and the PVM system, the NILM PSA algorithm showed the best speedup characteristics. However, the PILM PSA algorithm yielded a better final linking distance for the same number of iterations when compared to the NILM PSA algorithm in case of both, the Intel iPSC/860 and the PVM system. The synchronization overhead in the case of the PILM PSA algorithm led to a degradation in speedup and outweighed the benefits of convergence to a better solution. The NILM PSA algorithm exhibited a speedup of 5.35 on an 8-processor Intel iPSC/860 and a speedup of 3.40 on a 6-processor PVM system.

On the MasPar MP-2 the PILM MPSA algorithm showed the best speedup characteristics and also converged to a clone ordering with the least overall linking distance. The NILM MPSA algorithm performed slightly worse than the PILM MPSA algorithm. The synchronization overhead in this case is outweighed by the benefits of improved convergence and lower final linking distance. This could be attributed to the low synchronization overhead associated with a massively parallel SIMD architecture such as the MasPar MP-2. The PILM MPSA algorithm when implemented on the 2048-processor MasPar MP-2 system exhibited a speedup of 1232.

In Figs. 11–13 the linking distance is displayed as a function of time for the NILM PSA algorithm on the Intel iPSC/860, the NILM PSA algorithm on the PVM system, and the PILM MPSA algorithm on the MasPar MP-2, respectively, with the number of PE's as a variable parameter. Figs. 14–16 show the speedup as a function of final linking distance for the NILM PSA algorithm on the Intel iPSC/860, the NILM PSA algorithm on the PVM system and the PILM MPSA algorithm on the MasPar MP-2 respectively, with the number of PE's as a variable parameter. In all of the aforementioned algorithms, the inter-PE perturbations were based on clone block reversals and the deterministic synchronization technique was used.

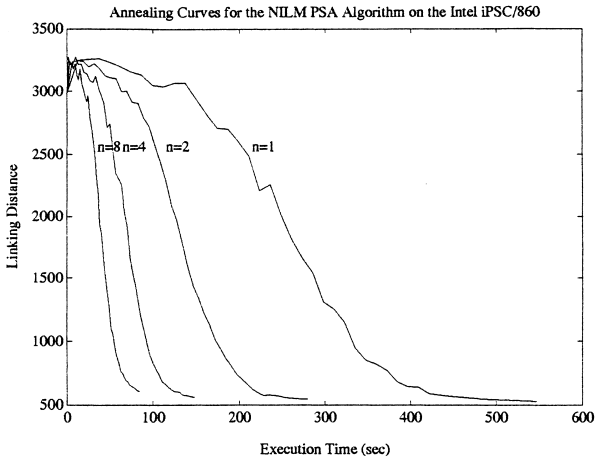

Fig. 11. NILM PSA algorithm on the Intel iPSC/860: Linking distance as a function of the execution time for varying no. of PE's n with $\lambda = 200,000$.

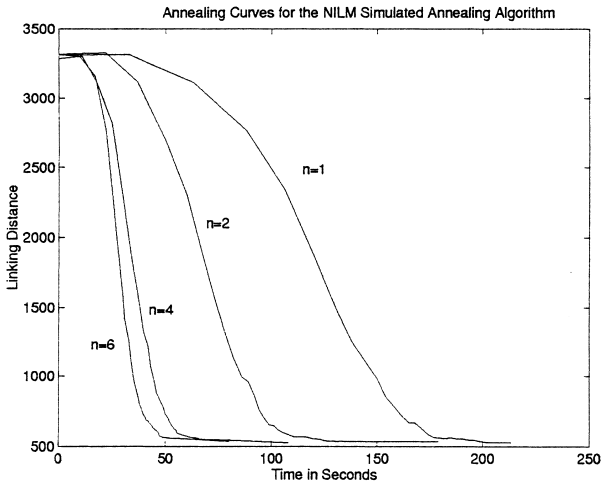

Fig. 12. The NILM PSA algorithm on the PVM system: Linking distance as a function of execution time for varying no. of processors, $\lambda = 200,000$.

4.2. Data distribution

The DM PSA algorithm on the Intel iPSC/860 and the PVM system, and the PIDM MPSA and NIDM MPSA algorithms on the MasPar MP-2 exhibited a tendency to get trapped in a local minimum. This could be attributed to the fact that the distribution of clonal data across more than one PE results in the acceptance of erroneous perturbations and thereby alters the state transition probabilities associated with the Markov chain of solution states generated by the serial simulated

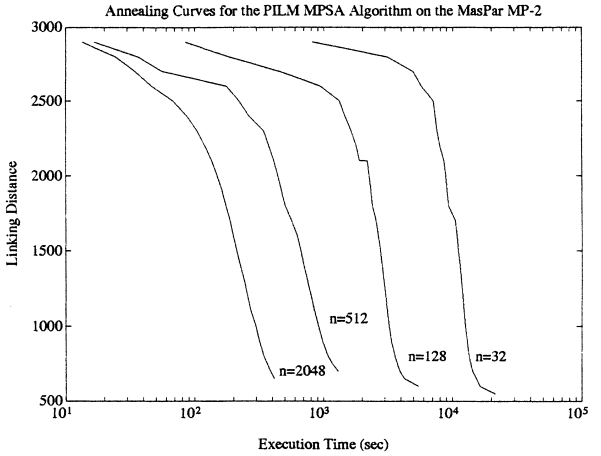

Fig. 13. PILM MPSA algorithm on the MasPar MP-2: Linking distance as a function of the log of execution time for varying no. of PE's n with $\lambda = 1,024,000$.

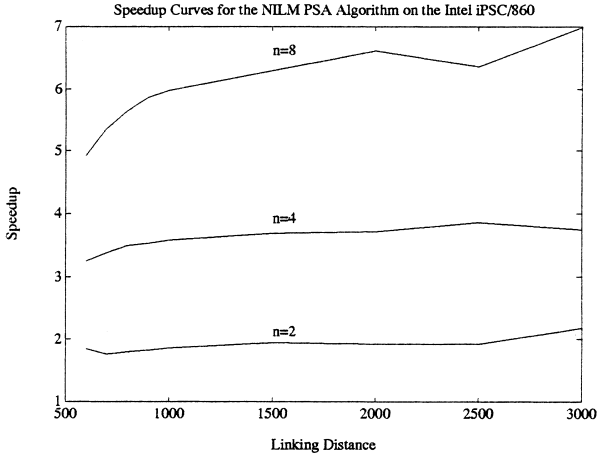

Fig. 14. NILM PSA algorithm on the Intel iPSC/860: Speedup as a function of linking distance for varying no. of PE's n with $\lambda = 200,000$.

annealing algorithm. Data distribution makes the DM PSA algorithm on the Intel iPSC/860 and the PVM system, and the PIDM MPSA and NIDM MPSA algorithms on the MasPar MP-2 more vulnerable to the presence of local minima. We surmise that this problem can be alleviated by either:

(i) restricting the perturbations to be non-interacting,

(ii) allowing interaction between non-neighboring PE's on the hypercube, token ring or 2-D mesh,

(iii) implementing a more gradual annealing schedule,

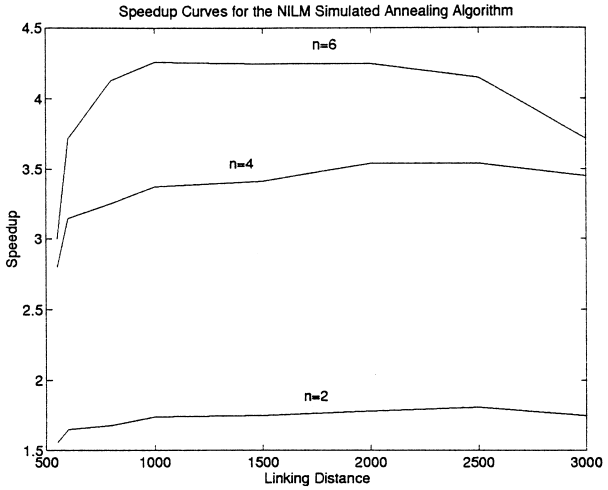

Fig. 15. The NILM PSA algorithm on the PVM system: Speedup as a function of linking distance for varying no. of processors, $\lambda = 200,000$.

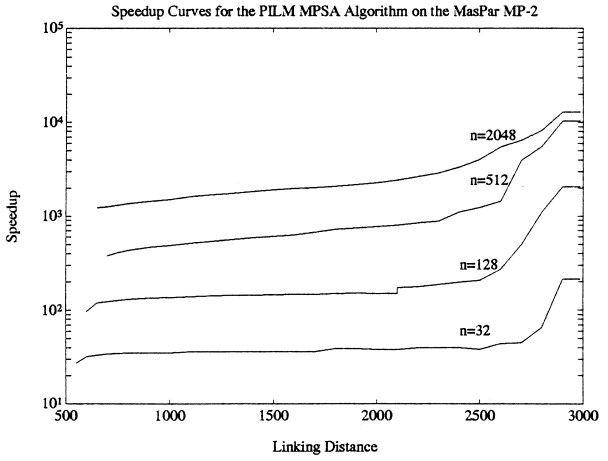

Fig. 16. PILM MPSA algorithm on the MasPar MP-2: Logarithm of speedup as a function of linking distance for varying no. of PE's n with $\lambda = 1,024,000$.

(iv) allowing a higher number of inter-PE perturbations relative to the number of intra-PE perturbations at each temperature value, or

(v) allowing for a higher number of iterations at each temperature value.

All the above options come at the expense of greater run time and consequently lower speedup.

We observed that in the case of the MasPar MP-2, it is only when the PE cluster is limited to two PE's that the PIDM MPSA algorithm and the NIDM MPSA

algorithm achieve speedup and convergence performances comparable to those of the PILM MPSA algorithm and the NILM MPSA algorithm, respectively. The reason for this is that the single hop communication delay in the MasPar MP-2 architecture is comparable to the CPU clock speed. Thus distributing the Markov chain over 2 adjacent processors in the MasPar MP-2 architecture does not result in an inter-PE communication overhead that is sufficient to cause serious performance degradation. Also, distributing the clonal data across two processors does not cause a significantly high number of erroneous perturbations to be generated (Eq. (6)). It should be noted that distribution of data typically makes an algorithm more scalable in the sense of its being able to handle larger data sets. In our case this scalability comes at the cost of an undesirable (but inevitable) tradeoff between speedup and convergence to a desirable solution.

4.3. Perturbation strategy

The intra-PE perturbation strategy based on clone block reversal performed better than the one based on clone exchange for all the three PSA algorithms on the Intel iPSC/860 and the PVM system and all the four MPSA algorithms on the MasPar MP-2 that were evaluated. In the case of inter-PE perturbations, the exchange of clone blocks was found to result in better performance than exchange of single clones. These observations are consistent with those of Szu and Hartley [49,50] who have shown that the convergence properties of simulated annealing can be considerably improved if there is a nonzero probability of a perturbation that represents a large transition in the solution space as opposed to strictly local perturbations. The pairwise clone exchange strategy is a strictly local search whereas the clone block reversal strategy is a semilocal search which consists of occasional large transitions in the solution space. Therefore, it comes as no surprise that the clone block reversal strategy is superior to the pairwise clone exchange strategy in so far as the convergence of the simulated annealing algorithm is concerned. The same argument holds when comparing the exchange of clone blocks with the exchange of single clones in the case of inter-PE perturbations.

4.4. Synchronization strategy

The use of stochastic synchronization with the Boltzmann function did not substantially improve the convergence characteristics of the PILM PSA algorithm on the Intel iPSC/860 and the PVM system or the PILM MPSA and PIDM MPSA algorithms on the MasPar MP-2. In fact, a slight degradation of performance was noticed when deterministic synchronization was replaced by stochastic synchronization in the case of all the aforementioned algorithms.

4.5. Merging a maximal set of disjoint perturbations

The merging of a maximal set of disjoint successful perturbations did improve the performance of the PILM PSA algorithm on the Intel iPSC/860 and the PVM

system, and the PILM MPSA and PIDM MPSA algorithms on the MasPar MP-2 in terms of the final linking distance for given values of the number of PE's and λ. Unfortunately computing the maximal set at every synchronization step led to a higher overhead in terms of execution time and was not justified in terms of the improvement in the final linking distance. The improved final linking distance could have been obtained with a lower overall execution time by having the straightforward NILM PSA algorithm run on the Intel iPSC/860 or the PVM system, or the PILM MPSA algorithm run on the MasPar MP-2, with a greater number of iterations for the same number of PE's.

5. Conclusions and future research

In this paper, we presented our practical experience with the design, analysis and implementation of PARODS – a suite of parallel algorithms for chromosome reconstruction via ordering of DNA sequences. The algorithms in PARODS were based on an earlier serial algorithm ODS which is a physical mapping algorithm based on simulated annealing. PARODS was motivated by the fact that while ODS proved very successful in generating high quality physical maps from DNA/DNA hybridization data, the annealing schedules necessary to obtain optimal or nearly optimal solutions proved to be computationally intensive.

In the current implementation of PARODS, we have designed and analyzed three models for a MIMD PSA algorithm and four models for a SIMD MPSA algorithm in the context of chromosome reconstruction via ordering of clones in a library. All the aforementioned models were based on the decomposition of the Markov chain of solution states generated by the serial simulated annealing program ODS and the mapping of these Markov chains on the individual PE's of the underlying multiprocessor architecture. The MIMD algorithms were implemented on an 8-processor Intel iPSC/860 system and a PVM system consisting of 6 SUN SPARC5 workstations, and the SIMD algorithms on a 2048-processor MasPar MP-2 system.

Of the three MIMD PSA algorithms implemented on the Intel iPSC/860 and the PVM system the NILM PSA model achieved the best performance in terms of rate of convergence and speedup, but the Periodically Interacting Local Markov chain (PILM) model achieved better performance in terms of the final linking distance. The NILM PSA algorithm exhibited a speedup of 5.35 on an 8-processor Intel iPSC/860 and 3.40 on a 6-processor PVM system. Of the four SIMD MPSA algorithms implemented on the MasPar MP-2, the PILM MPSA model achieved the best performance in terms of rate of convergence and the final linking distance. The PILM MPSA algorithm when implemented on the 2048-processor MasPar MP-2 system exhibited a speedup of 1232. Our results have shown that distribution of clonal data across the processors leads to degradation in performance in the case of both, the MIMD PSA algorithms on the Intel iPSC/860 and the PVM system, and the SIMD MPSA algorithms on the MasPar MP-2. However, as clonal data sets increase in size, PSA and MPSA models that work with distributed data might be the only feasible alternative and hence merit further investigation.

Our experimental results tabulated in Table 1 showed that dedicated multiprocessor platforms such as the MasPar MP-2 and the Intel iPSC/860 resulted in better efficiency of processor utilization than the PVM system. This is expected since the interprocessor communication overhead is higher in the case of the PVM system due to the higher latency of the general purpose network used. Nevertheless, the speedup and processor utilization efficiency of the PVM system are comparable to those of the MasPar MP-2 and the Intel iPSC/860. The PVM system has a significant cost/performance advantage over dedicated multiprocessor platforms since an investment in dedicated hardware resources is not called for. Given the cost/performance advantage of the PVM system and the ubiquity of general-purpose computers in almost every research environment, we expect parallel applications on PVM-based systems (or similar systems that exploit a network of general-purpose computers) to become more popular.

In terms of future research, our immediate goal is to make the algorithms in PARODS more scalable in terms of being able to handle distributed clonal data and larger clonal data sets without degradation in performance. To this end we intend to investigate memory-scalable and retrieval time-efficient data structures for the storage of clonal signatures and inter-clone distances. We also intend to investigate alternative ordering criteria to the one in Eq. (1) which is based on minimization of the summation of pairwise Hamming distances between consecutive clones in the ordering.

Acknowledgements

The support of the National Science Foundation (NSF BIR 94-22896, CCR-8717033 and CDA-8820544) is gratefully acknowledged. The research instrumentation grant by MasPar Computer Corporation which made possible the acquisition of the MasPar MP-2 by the University of Georgia is also acknowledged. The authors wish to thank the anonymous reviewers for their insightful comments and helpful suggestions which greatly improved the paper.

References

[1] E.H.L. Aarts, F.M.J. de Bont, J.H.A. Habers, P.J.M. van Laarhoven, A parallel statistical cooling algorithm, Lecture notes in computer science, Proceedings of the Third Annual Symposium on Theoretical Aspects of Computer Science, vol. 210, Springer, Berlin, 1986, pp. 87–97.

[2] R. Azencott, (Ed.), Simulated Annealing: Parallelization Techniques, Wiley, New York, 1992.

[3] P. Banerjee, M.H. Jones, J.S. Sargent, Parallel simulated annealing algorithms for cell placement on the hypercube multiprocessor, IEEE Trans. Parallel and Distributed Systems 1 (1990) 91–106.

[4] B. Bellane-Chantelot, B. LaCroix, P. Ougen, A. Billault, S. Beaufils, S. Bertrand, I. Georges, F. Gilbert, I. Gros, G. Lucotte, L. Susini, J.J. Codani, P. Gernouin, V.G. Pook, J. Lu-Kuo, T. Ried, D. Ward, I. Chumakov, D. Le Paslier, E. Barilott, D. Cohen, Mapping the whole human genome by fingerprinting yeast artificial chromosomes, Cell 70 (1992) 1059–1068.

[5] S.M. Bhandarkar, J. Arnold, Parallel simulated annealing on the hypercube for chromosome reconstruction, Proceedings of the IMACS 14th World Congress on Computational and Applied Mathematics, vol. 3, Atlanta, GA, 1994, pp. 1109–1112.
[6] S.M. Bhandarkar, S. Chirravuri, J. Arnold, D. Whitmire, Massively parallel algorithms for chromosome reconstruction, Proceedings of the Pacific Symposium on Biocomputing, Big Island, Hawaii, 1996, pp. 85–92.
[7] N. Boissin, J.L. Lutton, A parallel simulated annealing algorithm, Parallel Computing 19 (1993) 859–872.
[8] K.S. Booth, G.S. Lueker, Testing for the consecutive one's property, interval graphs and graph planarity using pq-tree algorithms, J. Comput. Systems Sci. 13 (1976) 335–379.
[9] A. Casotto, F. Romeo, A. Sangiovanni–Vincentelli, A parallel simulated annealing algorithm for the placement of macro cells, IEEE Trans. Computer-Aided Design 9 (1987) 838–847.
[10] G.A. Churchill, C. Burks, M. Eggert, M. Engle, M. Waterman, Assembling DNA sequence fragments by shuffling and simulated annealing, Technical Report, LA-UR-93-2287, Los Alamos National Laboratory, Los Alamos, NM, 1993.
[11] D. Cohen, I. Chumakov, J. Weissenbach, A first-generation physical map of the human genome, Nature 366 (1993) 698–701.
[12] A. Coulson, J. Sulston, S. Brenner, J. Karn, Toward a physical map of the genome of the nematode Caenorhabditis Elegans, Proc. Natl. Acad. Sci. 83 (1986) 7821–7825.
[13] F.S. Collins, M.L. Drumm, J.L. Cole, W.K. Lockwood, G.F. Vande Woude, M.C. Ianuzzi, Construction of a general human chromosome jumping library with application to cystic fibrosis, Science 235 (1987) 1046–1049.
[14] F. Collins, D. Galas, A new five-year plan for the US human genome project, Science 262 (1993) 53–56.
[15] A.J. Cuticchia, J. Arnold, W.E. Timberlake, The use of simulated annealing in chromosome reconstruction experiments based on binary scoring, Genetics 132 (1992) 591–601.
[16] A.J. Cuticchia, J. Arnold, W.E. Timberlake, ODS: Ordering DNA sequences – A physical mapping algorithm based on simulated annealing, CABIOS 9 (1993) 215–219.
[17] S. Date, R. Kulkarni, B. Kulkarni, U. Kulkarni-Kale, A.S. Kolaskar, Multiple alignment of sequences on parallel computers, CABIOS 9 (1993) 397–402.
[18] S. Foote, D. Vollrath, A. Hilton, D.C. Page, The human Y chromosome: Overlapping DNA clones spanning the euchromatic region, Science 258 (1992) 60–66.
[19] M.S. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[20] S. Geman, D. Geman, Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images, IEEE Trans. Pattern Analysis and Machine Intelligence 6 (1984) 721–741.
[21] F.D. Green, M.V. Olson, Chromosomal region of the cystic fibrosis gene in yeast artificial chromosomes: A model for human genome mapping, Science 250 (1990) 94–98.
[22] D.R. Greening, Parallel simulated annealing techniques, Physica D 42 (1990) 293–306.
[23] J.D. Hoheisel, E. Maier, R. Mott, L. McCarthy, A.V. Grigoriev, L.C. Schalkwyk, D. Nizetic, F. Frances, H. Lehrach, High resolution cosmid and P1 maps spanning the 14Mb genome of the fission yeast S Pombe, Cell 73 (1993) 109–120.
[24] X. Huang, W. Miller, S. Schwartz, R.C. Hardison, Parallelization of a local similarity search algorithm, CABIOS 8 (1992) 155–166.
[25] Intel iPSC/860 Concurrent Programming User Guide, Intel Corp., Beaverton, OR, 1992.
[26] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, M. Kanehisa, Multiple sequence alignment by parallel simulated annealing, CABIOS 9 (1993) 267–273.
[27] R. Jayaraman, R. Rutenbar, Floor planning by annealing on a hypercube multiprocessor, Proceedings of the IEEE International Conference on Computer Aided Design, 1987, pp. 346–349.
[28] R. Jones, Sequence pattern matching on a massively parallel computer, CABIOS 8 (1992) 377–384.
[29] S. Kirkpatrick, C. Gelatt Jr., M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 498–516.

[30] E.S. Lander, P. Green, Construction of multi-locus genetic linkage maps in humans, Proc. Natl. Acad. Sci. 84 (1987) 2363–2367.
[31] F.H. Lee, G.S. Stiles, V. Swaminathan, Parallel annealing on distributed memory systems, Technical Report, Department of Electrical and Computer Engineering, Utah State University, Logan, UT, 1994.
[32] S. Lin, B. Kernighan, An effective heuristic for the traveling salesman problem, Operations Research 21 (1973) 498–516.
[33] S. Liuni, N. Prunella, G. Pesole, T. D'Orazio, E. Stella, A. Distante, SIMD parallelization of the WORDUP algorithm for detecting statistically significant patterns in DNA sequences, CABIOS 9 (1993) 701–708.
[34] MasPar Parallel Applications Language (MPL), Reference Guide, MasPar Computer Corp., Sunnyvale, CA, 1993.
[35] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chemical Physics 21 (1953) 1087–1092.
[36] T. Mizukami, W.I. Chang, I. Garkatseve, N. Kaplan, D. Lombardi, T. Matsumoto, O. Niwa, A. Kounosu, M. Yanagida, T.G. Marr, D. Beach, A 13kb resolution cosmid map of the 14 Mb fission yeast genome by nonrandom sequence-tagged site mapping, Cell 73 (1993) 121–132.
[37] P.L. Miller, P.M. Nadkarni, W.R. Pearson, Comparing machine-independent versus machine-specific parallelization of a software platform for biological sequence comparison, CABIOS 8 (2) (1992) 167–175.
[38] P.L. Miller, P.M. Nadkarni, P.A. Bercovitz, Harnessing networked workstations as a powerful parallel computer: A general paradigm illustrated using three programs for genetic linkage analysis, CABIOS 8 (2) (1992) 141–147.
[39] M.V. Olson, J.E. Dutchik, M.Y. Graham, G.M. Brodeur, C. Helms, M. MacCollin, R. Scheinman, M. Frank, Random-clone strategy for genomic restriction mapping in yeast, Proc. Natl. Acad. Sci. 83 (1986) 7826–7830.
[40] M.V. Olson, L. Hood, C. Cantor, D. Botstein, A common language for physical mapping of the human genome, Science 245 (1989) 1434–1435.
[41] R. Parsons, S. Forrest, C. Burks, Genetic algorithms for DNA sequence assembly, in: L. Hunter, D. Searles, J. Shavlik (Eds.), Proceedings of the First International Conference of Intelligent Systems for Molecular Biology, AAAI Press, Menlo Park, CA, 1993, pp. 310–318.
[42] M.C. Pease, The indirect binary N-cube microprocessor array, IEEE Transactions on Computers, C/25 (5) (1977) 458–473.
[43] R.A. Prade, J. Griffith, K. Kochut, J. Arnold, W.E. Timberlake, In vitro reconstruction of the Aspergillus Nidulans genome, Genetics, Proc. Nat. Acad. Sci. 94 (26) (1997) 14564–14569.
[44] F. Romeo, A. Sangiovanni–Vincentelli, A theoretical framework for simulated annealing, Algorithmica 6 (1991) 302–345.
[45] C.L. Smith, R.D. Kolodner, Mapping of Escherichia coli chromosomal Tn5 and F insertions by pulsed field gel electrophoresis, Genetics 119 (1988) 227–236.
[46] C. Soderlund, C. Burks, GRAM and GenfragII: Solving and testing the single-digest, partially ordered restriction map problem, CABIOS 10 (1994) 349–358.
[47] A.H. Sturtevant, The linear arrangement of six sex-linked factors in Drosophila as shown by their mode of association, J. Exp. Zool. 14 (1913) 43–49.
[48] V. Sunderam, PVM: A framework for parallel distributed computing, Concurrency: Practice and Experience 2 (2) (1992) 315–339.
[49] H. Szu, R. Hartley, Fast simulated annealing, Physics Letters A 122 (3/4) (1987) 157–162.
[50] H. Szu, R. Hartley, Nonconvex optimization by fast simulated annealing, Proc. IEEE 75 (11) (1987) 1538–1540.
[51] Y. Wang, R.A. Prade, J. Griffith, W.E. Timberlake, J. Arnold, A fast random cost algorithm for physical mapping, Proc. Natl. Acad. Sci. 91 (1994) 11094–11098.
[52] E.E. Witte, R.D. Chamberlain, M.A. Franklin, Parallel simulated annealing using speculative computation, IEEE Trans. Parallel and Distributed Systems 2 (4) (1991) 483–494.

[53] C.P. Wong, R.D. Fiebrich, Simulated annealing-based circuit placement on the connection machine system, Proceedings of the International Conference on Computer Design, 1987, pp. 78–82.
[54] M. Xiong, H.J. Chen, R.A. Prade, Y. Wang, J. Griffith, W.E. Timberlake, J. Arnold, On the consistency of a physical mapping method to reconstruct a chromosome in vitro, Genetics 142 (1) (1996) 267–284.
[55] P. Zhang, E.A. Schon, S.G. Fischer, E. Cayanis, J. Weiss, S. Kistler, P.E. Bourne, An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA, CABIOS 10 (1994) 309–317.