CABIOS

Vol. 12 no. 4 1996
Pages 269–280

PARODS—a study of parallel algorithms for ordering DNA sequences

Suchendra M.Bhandarkar, Sridhar Chirravuri and Jonathan Arnold[1]

Abstract

A suite of parallel algorithms for ordering DNA sequences (termed PARODS) is presented. The algorithms in PARODS are based on an earlier serial algorithm, ODS, which is a physical mapping algorithm based on simulated annealing. Parallel algorithms for simulated annealing based on Markov chain decomposition are proposed and applied to the problem of physical mapping. Perturbation methods and problem-specific annealing heuristics are proposed and described. 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 are presented. The convergence, speedup and scalability characteristics of the aforementioned algorithms are analyzed and discussed. The best SIMD algorithm is shown to have a speedup of ~1000 on the 2048 processor MasPar MP-2 system, whereas the best MIMD algorithm is shown to have a speedup of ~5 on the 8 processor Intel iPSC/860 system.

Introduction

Creating physical maps for each of the human chromosomes and those of several model systems is the central goal of the Human Genome Project (Collins and Galas, 1993). In our previous work (Cuticchia *et al.*, 1992, 1993), we presented a physical mapping algorithm ODS (**O**rdering **D**NA **S**equences) based on simulated annealing. The physical mapping approach in ODS can be summarized as follows: (i) each clone is scored for the presence or absence of specific probes resulting 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: $D = \sum_{i=1}^{n-1} d(C_i, C_{i+1})$; (iv) the desired (i.e. optimal) ordering is deemed to be that which results in minimization of the ordering criterion D (Xiong *et al.*, 1996).

The problem of computing such an optimal clone ordering can be shown to be isomorphic to the classical NP-complete Optimal Linear Ordering (OLO) problem for which no polynomial–time algorithm for finding the optimal solution is known, except for some simple cases that deal with error-free data (Booth and Lueker, 1976). Stochastic optimization algorithms such as simulated annealing (Kirkpatrick *et al.*, 1983; Geman and Geman, 1984; Aarts and Korst, 1989) are capable of avoiding local optima in the solution space and producing solutions that are close to the 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 (Cuticchia *et al.*, 1992, 1993). One of the drawbacks of a serial implementation of simulated annealing is that the annealing schedules necessary to obtain a solution close to the 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 (Huang *et al.*, 1992; Jones, 1992; Miller *et al.*, 1992a; Liuni *et al.*, 1993), sequence alignment (Date *et al.*, 1993; Ishikawa *et al.*, 1993), genetic mapping (Miller *et al.*, 1992b) and physical mapping (Bhandarkar and Arnold, 1994; Bhandarkar *et al.*, 1996).

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 well documented in the literature:

(1) Functional parallelism (Wong and Fiebrich, 1987).
(2) Control parallelism with
   (i) speculative computation (Witte *et al.*, 1991);
   (ii) parallel Markov chain generation using a systolic algorithm (Aarts *et al.*, 1986; Greening, 1990; Azencott, 1992);

Department of Computer Science, University of Georgia, Athens, GA 30602-7404 and [1]Department of Genetics, University of Georgia, Athens, GA 30602-7223, USA

E-mail: suchi@cs.uga.edu

© Oxford University Press

(iii) multiple independent or periodically interacting searches (Azencott, 1992; Lee *et al.*, 1994).

(C) Data parallelism with
- (i) parallel evaluation of multiple moves with acceptance of a single move (Casotto *et al.*, 1987);
- (ii) parallel evaluation of multiple moves with acceptance of non-interacting multiple moves (Jayaraman and Rutenbar, 1987);
- (iii) parallel evaluation and acceptance of multiple moves (Banerjee *et al.*, 1990).

Of the aforementioned parallelization strategies, we deemed the ones based on multiple searches (i.e. B (ii)) and parallel evaluation and acceptance of multiple moves (i.e. C (iii)) to be the most promising from the viewpoint of parallelizing ODS. The resulting suite of parallel SIMD/MIMD algorithms for chromosome reconstruction via clone ordering based on simulated annealing is termed PARODS (**PAR**allel Algorithms for **O**rdering **DNA** Sequences).

Systems and methods

The two multiprocessor platforms that were considered for the implementation of PARODS are (i) the MasPar MP-2 and (ii) the Intel iPSC/860, both of which reside on the University of Georgia campus.

The MasPar MP-2 is a massively parallel SIMD computer with the processing elements (PEs) interconnected in a toroidal two-dimensional grid or mesh topology. In order to parallelize the program ODS (Cuticchia *et al.*, 1993) on the MasPar MP-2, it was redesigned and recoded in the MPL programing language (*MasPar Parallel Applications Language (MPL), User Guide*, 1993; *MasPar Parallel Applications Language (MPL), Reference Guide*, 1993). Being an SIMD architecture, all the PEs in the MasPar MP-2 share the same instruction stream, but operate on different data (Hord, 1990). The MPL language is an extension of ANSI C with constructs that allow SIMD programing. 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 Intel iPSC/860 is a MIMD multiprocessor system with the individual PEs connected in a Boolean hypercube topology (Pease, 1977). In order to implement the program ODS (Cuticchia *et al.*, 1993) 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 programing language such as C (*Intel iPSC/860 Concurrent Programming User Guide*, 1992). 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 (*Intel iPSC/860 C Commands and Routines: Reference Guide*, 1992).

Algorithms

A typical serial simulated annealing algorithm (Figure 1) consists of three phases:

Perturb phase: Given an n-variable objective function of the form $f(\mathrm{x}) = f(x_1, x_2 \ldots, x_n)$ to be minimized 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. $f(\mathrm{x}_j)$ is computed. In our case, the total linking distance associated with the new candidate solution (i.e. new clone ordering) is computed.

Decide phase: If $f(\mathrm{x}_j) < f(\mathrm{x}_i)$, then x_j is accepted as the new candidate solution. If $f(\mathrm{x}_j) \geq f(\mathrm{x}_i)$, then x_j is accepted as the new candidate solution with probability p, whose value is computed using the Metropolis function: $p = \exp(-(f(\mathrm{x}_j) - f(\mathrm{x}_i))/T)$ for a given value of temperature T. The current solution x_i is retained with probability $(1 - p)$.

The perturb–evaluate–decide cycle, which constitutes a single iteration of the simulated annealing algorithm, is carried out a fixed number of times for a given value of T, which is then systematically reduced. A sequence of monotonically decreasing positive values for T, denoted by $\{T_k\}$, is called a temperature schedule, and the generating function for this sequence is called the annealing function.

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. Consequently, we have formulated and implemented various models of PSA based on the distribution of the Markov chain of solution states (on the individual PEs of the MasPar MP-2 and the Intel iPSC/860) and the sychronization method used. These models incorporate the parallelization strategies B (ii) and C (iii) discussed in the previous section.

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, we have formulated and implemented three models of a PSA algorithm: (a) Non-Interacting Local Markov chain (NILM) PSA algorithm; (b) Periodically Interacting Local Markov chain (PILM) PSA algorithm; (c) Distributed Markov chain (DM) PSA algorithm.

In the NILM MPSA and the NILM PSA algorithms, each PE in the MasPar MP-2 and the Intel iPSC/860, respectively, runs an independent version of the serial simulated annealing algorithm. In essence, there are as many Markov chains of solution states as there are PEs 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.

Two perturbation strategies 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, the positions of two randomly chosen clones are interchanged to generate a new permutation σ_2.

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 $i,j \leq n$) 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, at each temperature value T, the perturb–evaluate–accept cycle of the serial simulated annealing algorithm (Figure 1) is executed concurrently in all the PEs COUNT_LIMIT number of times just as in the case of NILM MPSA or the NILM PSA algorithm. However, just before the parameter T is updated using the annealing function, the best candidate solution from among those in all the PEs is selected and duplicated on all the other PEs. 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

```
const float T_min, T_max, T_delta
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
 function: T = A(T);
}
```

Fig. 1. Outline of a serial simulated annealing algorithm.

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 two-dimensional mesh topology, it is convenient to have the PE clusters in the form of non-overlapping rectangular submeshes of equal size. 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 two, so are X_{width} and Y_{width}. The perturbations in the NIDM MPSA algorithm are carried out in three distinct phases as described below.

(A) Intra-PE perturbations

All the PEs 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 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 PEs is given by:

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

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

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

As can be seen, $p \rightarrow 1$ in the limit as $n \rightarrow \infty$. Also, for a fixed value of n, $p \rightarrow 0$ in the limit as $k \rightarrow \infty$. 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 PEs. Banerjee *et al.* (1990) 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. This observation and equation (2) place an upper limit on the number of PEs over which the clonal data of a given size can be distributed.

(B) Inter-PE perturbations along the cluster rows

The PEs are paired along the rows in each PE cluster. One of the PEs 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. 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 PEs update their respective local clone ordering. 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 be 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 PEs 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 periodically interacting searches and parallel evaluation and acceptance of multiple moves. The control structure of the PIDM MPSA algorithm, outlined in Figure 2, is the most general of the four MPSA models for the MasPar MP-2 described in this paper. Deletion of Phase 5 in Figure 2 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. $X_{width} = 1$ and $Y_{width} = 1$.

In the DM PSA algorithm on the Intel iPSC/860, the candidate solution is distributed over the PEs in the hypercube such that each PE has knowledge only about its own local clone ordering. The perturbations in the DM PSA algorithm are carried out in two distinct phases.

(A) Intra-PE perturbations

All the PEs concurrently perform the perturb–evaluate–accept cycle on their individual portions (i.e. subsolutions)

```
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 solution 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 function: T = A(T);
}
```

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

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

During this phase, the PEs in the hypercube are paired along the inter-PE links in each dimension of the hypercube. In a hypercube with $N = 2^n$ PEs, there are $2^{n-1} = N/2$ disjoint PE pairs in each of the n dimensions such that the PEs that constitute a single pair are directly connected by an inter-PE link in that dimension. In an inter-PE perturbation, the PEs within a PE pair evaluate the resulting change in the cost function. In each PE pair, one of the PEs 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 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 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

```
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;
  (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 function 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. 3. The process that runs on the host computer of the Intel iPSC/860.

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.

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 be 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 PEs 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 PEs to construct the overall clone ordering. The DM PSA algorithm is based on the parallel evaluation and acceptance of multiple moves.

In Figure 3, the process that runs on the host computer is outlined. In Figure 3, Phase 3: Periodic Synchronization is carried out only in the case of the PILM PSA algorithm. Figure 4 outlines the process that runs on each PE. Figures 5 and 6 describe the intra-PE perturbations and inter-PE perturbations, respectively. Typically, the total number of inter-PE perturbations is a fraction (denoted by the variable FACTOR in Figure 6) of the total number of intra-PE perturbations. In our case, the value of FACTOR was in the range 1–5%.

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 (Romeo and Sangiovanni-Vincentelli, 1991) 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

```
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 function: T = A(T);
  }
 Phase 3: Result Integration: Send local
          ordering to Host;
}
```

Fig. 4. The process that runs on the node PEs of the Intel iPSC/860.

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 is reduced to only

```
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 candiate
        subsolutions in each PE with
        probability P_T(f_delta) computed
        using the Metropolis function;
  }
```

Fig. 5. The intra-PE perturbations on the node PEs of the Intel iPSC/860.

```
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 of
      Master;
      Update local clone ordering;
     }
    }
   }
 }
```

Fig. 6. The inter-PE perturbations on the iPSC/860.

0.1–0.2% of COUNT_LIMIT, thus speeding up the algorithm. At lower temperatures, since there 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 final temperature value is reached. The annealing schedule that was chosen was a geometric schedule (Romeo and Sangiovanni-Vincentelli, 1991) of the form $T_{next} = \beta T_{prev}$, where $\beta = 0.9$.

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, 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 distance D is duplicated on the ith PE/PE cluster containing solution with linking distance D_i with a probability P_B, where $P_B = 1/(1 + \exp(\frac{D-D_i}{T}))$.

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 graph coloring problem (Boissin and Lutton, 1993) where one is required to color the nodes of a graph with as few colors as possible with no two adjacent nodes (i.e. nodes with a common edge) having the same color. The largest set of nodes with the same color then corresponds to the maximal set of disjoint successful perturbations. 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. The greedy algorithm proceeds to color the graph by coloring the node with the highest degree and thereafter at each stage in the algorithm, an uncolored node with the highest degree is chosen and assigned a suitable color depending on the colors of its neighboring nodes. The greedy algorithm has an order of complexity that is linear with respect to the number of nodes in the graph.

Implementation

All the aforementioned algorithms were run on a DNA/DNA hybridization data set derived from Chromosome IV of the fungus *Aspergillus nidulans* (Wang *et al.*, 1994a,b). The data set consisted of 592 clones with each clone having a 115 bit clonal signature.

In the case of the MIMD PSA algorithms on the Intel iPSC/860, the inter-clone distances were pre-computed and stored on the PEs in the form of a lookup table or distance matrix. Owing to the limited PE memory on the MasPar MP-2, on-the-fly computation of inter-clone distances instead of a distance matrix lookup was resorted to. Note that the memory requirements of the distance matrix scale as $O(N^2)$ for N clones. In order to obtain a fair comparison between the various algorithms, the product (denoted by λ) of the number of PEs 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 PEs with a maximum of M iterations per PE at any given temperature, then with $N/2$ PEs the maximum number of iterations per PE at any given temperature would be $2M$ and with 1 per PE the maximum number of iterations per PE at any given temperature would be $M \cdot N$. 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, 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 pre-specified 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 implementation 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.

(1) *Speedup.* On the Intel iPSC/860, 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. 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.

On the MasPar MP-2, the PILM MPSA algorithm showed the best speedup characteristics and also converged to a clone ordering with 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.

In Figures 7 and 8, the linking distance is displayed as a function of time for the NILM PSA algorithm on the Intel iPSC/860 and the PILM MPSA algorithm on the MasPar MP-2, respectively, with the number of PEs as a variable parameter. In Figures 9 and 10, we show the speedup as a function of final linking distance for the NILM PSA algorithm on the Intel iPSC/860 and the PILM MPSA

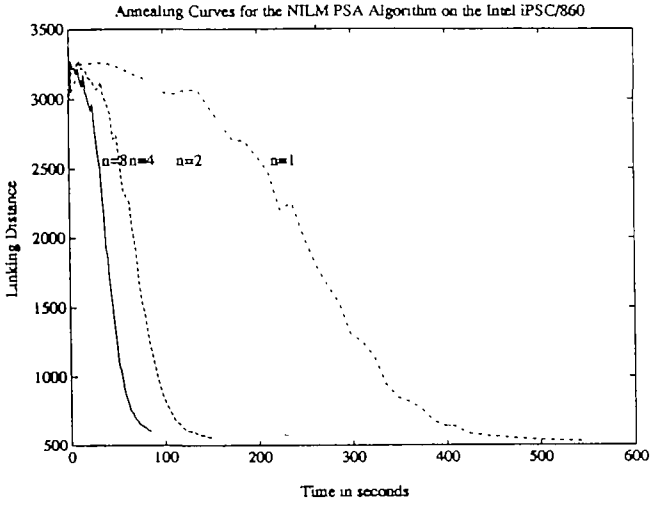

Fig. 7. NILM PSA algorithm on the Intel iPSC/860: linking distance as a function of the execution time for varying numbers of PEs with $\lambda = 200\,000$.

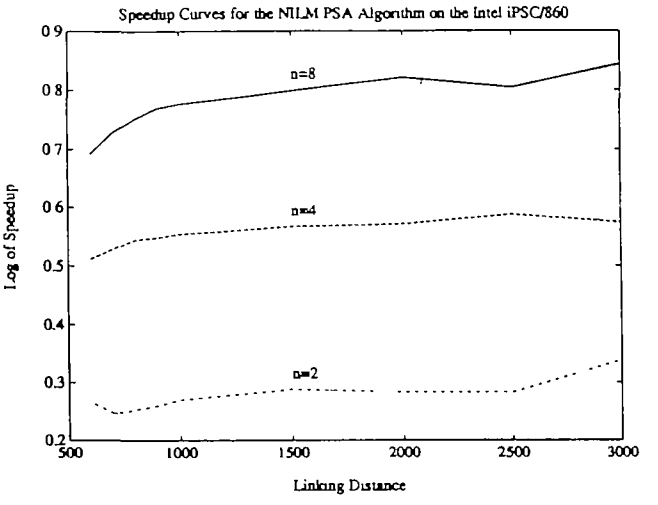

Fig. 9. NILM PSA algorithm on the Intel iPSC/860: logarithm of speedup as a function of linking distance, $\lambda = 200\,000$.

algorithm on the MasPar MP-2, respectively, with the number of PEs as a variable parameter. In Table I, the dependence of the execution time on the number of PEs for a given value of the final linking distance ($D = 700$) is shown for all the three PSA models implemented on the Intel iPSC/860. Table II shows the dependence of the execution time on the number of PEs used for a given value of the final linking distance ($D = 700$) for all the four MPSA models implemented on the MasPar MP-2. In all of the aforementioned algorithms, the inter-PE perturbations were based on clone block reversals and the deterministic synchronization technique was used.

(2) *Data distribution.* The DM PSA algorithm on the Intel iPSC/860 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 annealing algorithm. Data distribution makes the DM PSA algorithm on the Intel iPSC/860 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 PEs on the hypercube or two-dimensional mesh, (iii) implementing a more gradual annealing schedule, (iv) allowing a higher number of

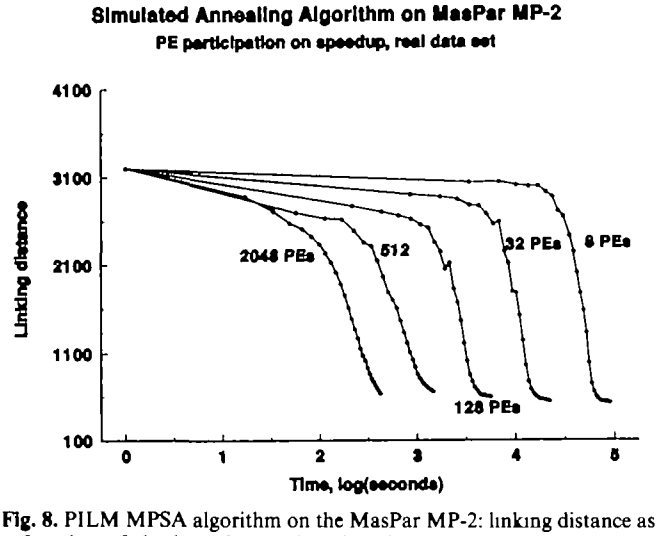

Fig. 8. PILM MPSA algorithm on the MasPar MP-2: linking distance as a function of the log of execution time for varying numbers of PEs, $\lambda = 1\,024\,000$.

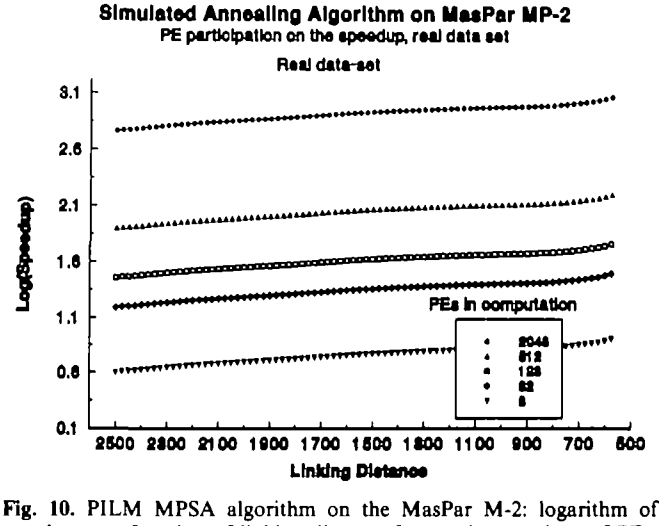

Fig. 10. PILM MPSA algorithm on the MasPar M-2: logarithm of speedup as a function of linking distance for varying numbers of PEs, $\lambda = 1\,024\,000$.

Table I. Dependence of execution time(s) on the number of PEs: a comparison of the various PSA algorithms on the Intel iPSC/860, $\lambda = 200000$, FACTOR = 0.02

No. of PEs	$D = 700$		
	NILM PSA	PILM PSA	DM PSA
8	69.80	223.90	304.60
4	110.56	261.90	323.70
2	212.35	294.50	362.10
1	373 39	373.39	373.39

Table II. Dependence of execution time(s) on the number of PEs: a comparison of the various MPSA algorithms on the MasPar MP-2, $\lambda = 1024000$

No. of PEs	$D = 700$			
	NILM MPSA	PILM MPSA	NIDM MPSA	PIDM MPSA
2048	397.0	383.0	587.0	502.0
512	1356.0	1294.0	2234.0	1909.0
128	4017.0	3900.0	5986.0	5117.0

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 PEs 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 two 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 [equation (2)]. 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.

(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 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.

(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 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.

(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 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 PEs 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 or the PILM MPSA algorithm run on the Intel iPSC/860 or MasPar MP-2, respectively, with a greater number of iterations for the same number of PEs.

Discussion

In this paper, we presented the design, analysis and implementation of PARODS—a suite of parallel algorithms for ordering 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 by ordering 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 PEs of the multiprocessor architecture. The MIMD algorithms were implemented on an 8 processor Intel iPSC/860 hypercube and the SIMD algorithms on a 2048 processor MasPar MP-2 mesh computer.

Of the three MIMD PSA algorithms implemented on the Intel iPSC/860, the NILM PSA model achieved the best performance in terms of rate of convergence and

speedup, but the PILM model achieved better performance in terms of the final linking distance. The NILM PSA algorithm exhibited a speed up of ~5 on an 8 processor Intel iPSC/860. 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 ~1000. 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 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.

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 intend to extend PARODS by inclusion of parallel programs for physical mapping that are designed to run on a network of workstations. For this purpose, we intend to use the Parallel Virtual Machine (PVM) (Sunderam, 1990) or Message Passing Interface (MPI) (Dongarra *et al.*, 1993) which, like Network Linda (Carriero and Gelernter, 1988), are distributed parallel programing environments specifically designed for a network of workstations. This would enable PARODS to be used by a wider circle of researchers in genome informatics.

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. We wish to thank the anonymous referees for their insightful comments and useful suggestions which greatly improved the final paper.

References

Aarts,E.H.L. and Korst,K. (1989) *Simulated Annealing and Boltzman Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley Inc., New York.

Aarts,E.H.L., de Bont,F.M.J., Habers,J.H.A. and van Laarhoven,P.J.M. (1986) A parallel statistical cooling algorithm. In *Lecture Notes in Computer Science: Proceedings of the 3rd Annual Symp. Theoretical Aspects of Computer Science*. Springer-Verlag, Berlin, 210, pp. 87–97.

Azencott,R. (ed.) (1992) *Simulated Annealing: Parallelization Techniques*. John Wiley Inc., New York.

Banerjee,P., Jones,M.H. and Sargent,J.S. (1990) Parallel simulated annealing algorithms for cell placement on the hypercube multiprocessor. *IEEE Trans. Parallel Distrib. Syst.*, **1**, 91–106.

Bhandarkar,S.M. and Arnold,J. (1994) Parallel simulated annealing on the hypercube for chromosome reconstruction. In *Proceedings of the IMACS 14th World Congress on Computational and Applied Mathematics*, **3**, IMACS Press, New York, pp. 1109–1112.

Bhandarkar,S.M., Chirravuri,S., Arnold,J. and Whitmire,D. (1996) Massively parallel algorithms for chromosome reconstruction. In *Proceedings of a Pacific Symposium on Biocomputing*, World Sci. Pub., Singapore, pp. 85–92.

Boissin,N. and Lutton,J.L. (1993) A parallel simulated annealing algorithm. *Parallel Comput.* **19**, 859–872.

Booth,K.S. and Lueker,G.S. (1976) Testing for the consecutive one's property, interval graphs and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, **13**, 335–379.

Carriero,N. and Gelernter,D. (1988) Applications experience with Linda. In *Proceedings of the ACM Symposium on Parallel Programming*. ACM Press, New York, pp. 173–187.

Casotto,A., Romeo,F. and Sangiovanni-Vincentelli,A. (1987) A parallel simulated annealing algorithm for the placement of macro cells. *IEEE Trans. Computer-Aided Design*, **9**, 838–847.

Collins,F. and Galas,D. (1993) A new five-year plan for the U.S. human genome project. *Science*, **262**, 53–56.

Cuticchia,A.J., Arnold,J. and Timberlake,W.E. (1992) The use of simulated annealing in chromosome reconstruction experiments based on binary scoring. *Genetics*, **132**, 591–601.

Cuticchia,A.J., Arnold,J. and Timberlake,W.E. (1993) ODS: ordering DNA sequences—a physical mapping algorithm based on simulated annealing. *Comput. Applic. Biosci.*, **9**, 215–219.

Date,S., Kulkarni,R., Kulkarni,B., Kulkarni-Kale,U. and Kolaskar,A.S. (1993) Multiple alignment of sequences on parallel computers. *Comput. Applic. Biosci.*, **9**, 397–402.

Dongarra,J.J., Hempel,R., Hey,A.J.G. and Walker,D.W. (1993) A proposal for a user-level message passing interface (MPI) in a distributed memory environment. Technical Report TM-12231, Oak Ridge National Laboratory, Oak Ridge, TN.

Geman,S. and Geman,D. (1984) Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis Machine Intelligence*, **6**, 721–741.

Greening,D.R. (1990) Parallel simulated annealing techniques. *Physica D*, **42**, 293–306.

Hord,R.M. (1990) *Parallel Supercomputing in SIMD Architectures*. CRC Press, Boca Raton, FL.

Huang,X., Miller,W., Schwartz,S. and Hardison,R.C. (1992) Parallelization of a local similarity search algorithm. *Comput. Applic. Biosci.*, **8**, 155–166.

Intel iPSC/860 Concurrent Programming User Guide. 1992. Intel Corp., Beaverton, OR.

Intel iPSC/860 C Commands and Routines: Reference Guide. 1992. Intel Corp., Beaverton, OR.

Ishikawa,M., Toya,T., Hoshida,M., Nitta,K., Ogiwara,A. and Kanehisa,M. (1993) Multiple Sequence Alignment by Parallel Simulated Annealing. *Comput. Applic. Biosci.*, **9**, 267–273.

Jayaraman,R. and Rutenbar,R. (1987) Floor planning by annealing on a hypercube multiprocessor. In *Proceedings of the IEEE International Conference on Computer Aided Design*, IEEE Press, New York, pp. 346–349.

Jones,R. (1992) Sequence pattern matching on a massively parallel computer. *Comput. Applic. Biosci.*, **8**, 377–384.

Kirkpatrick,S., Gelatt,C. Jr and Vecchiet,M. (1983) Optimization by simulated annealing. *Science*, **220**, 498–516.

Lee,F.H., Stiles,G.S. and Swaminathan,V. (1994) *Parallel Annealing on Distributed Memory Systems*. Technical Report, Department of Electrical and Computer Engineering, Utah State University, Logan, UT.

Liuni,S., Prunella,N., Pesole,G., D'Orazio,T., Stella,E. and Distante,A. (1993) SIMD parallelization of the WORDUP algorithm for detecting statistically significant patterns in DNA sequences. *Comput. Applic. Biosci.*, **9**, 701–708.

MasPar Parallel Applications Language (MPL). User Guide. 1993. MasPar Computer Corp., Sunnyvale, CA.

MasPar Parallel Applications Language (MPL), Reference Guide. 1993. MasPar Computer Corp., Sunnyvale, CA.

Miller,P.L., Nadkarni,P.M. and Pearson,W.R. (1992a) Comparing machine-independent versus machine-specific parallelization of a software platform for biological sequence comparison. *Comput. Applic. Biosci.*, **8**, 167–175.

Miller,P L., Nadkarni,P.M. and Bercovitz,P.E. (1992b) Harnessing networked workstations as a powerful parallel computer: a general paradigm illustrated using three programs for genetic linkage analysis. *Comput. Applic. Biosci.*, **8**, 141–147.

Pease,M.C. (1977) The Indirect Binary n-cube Microprocessor Array. *IEEE Trans. Comput.*, **25**, 458–473.

Romeo,F. and Sangiovanni-Vincentelli,A. (1991) A theoretical framework for simulated annealing. *Algorithmica*, **6**, 302–345.

Sunderam,V. (1990) PVM: a framework for parallel distributed computing. *Concurrency. Practice Experience*, **2**, 315–339.

Wang,Y., Prade,R.A., Griffith,J., Timberlake,W.E. and Arnold,J. (1994a) A fast random cost algorithm for physical mapping. *Proc. Natl Acad. Sci. USA*, **91**, 11094–11098.

Wang,Y., Prade,R.A., Griffith,J., Timberlake,W.E. and Arnold,J. (1994b) ODS_BOOTSTRAP: assessing the statistical reliability of physical maps by bootstrap resampling. *Comput. Applic. Biosci.*, **10**, 625–634.

Witte,E.E., Chamberlain,R.D. and Franklin,M.A. (1991) Parallel simulated annealing using speculative computation. *IEEE Trans. Parallel Distrib. Syst.*, **2**, 483–494.

Wong,C.P. and Fiebrich,R.D. (1987) Simulated annealing-based circuit placement on the connection machine system. In *Proceedings of IEEE International Conference on Computer Design*, IEEE Press, New York, pp. 78–82.

Xiong,M., Chen,H.J., Prade,R.A., Wang,Y., Griffith,J., Timberlake,W.E. and Arnold,J. (1996) On the consistency of a physical mapping method to reconstruct a chromosome *in vitro*. *Genetics*, **142**, 267–284.

Received on October 25, 1995; accepted on June 12, 1996