Massively Parallel Algorithms for Chromosome Reconstruction

Suchendra M. Bhandarkar[1] Sridhar Chirravuri[1]
Jonathan Arnold[2] David Whitmire[3]
Department of Computer Science[1] Department of Genetics[2]
Department of Biological and Agricultural Engineering[3]
University of Georgia, Athens, GA 30602–7404, U.S.A.

Abstract

Ordering clones from a genomic library into physical maps of whole chromosomes presents a central computational problem in genetics. Chromosome reconstruction via clone ordering is shown to be isomorphic to the NP-complete *Optimal Linear Ordering* problem. Massively parallel algorithms for simulated annealing based on Markov chain distribution are proposed and applied to this problem. Perturbation methods and problem-specific annealing heuristics are proposed and described. Experimental results on a 2048 processor MasPar MP-2 system are presented. Convergence, speedup and scalability characteristics of the various algorithms are analyzed and discussed.

1 Introduction

A central problem in genetics is that of creating maps of entire chromosomes which could then be used to reconstruct the chromosome's DNA sequence. These chromosomal maps fall into two broad categories - *genetic maps* and *physical maps*. A *physical map* is defined as a partial ordering of distinguishable DNA fragments or *clones* by their position along the entire chromosome. The specific technique discussed in this paper for generating a chromosomal physical map is one based on clone ordering[7]. The clones in a clonal library or contig library are to be ordered with respect to their position along a chromosome in order to generate a chromosomal physical map.

1.1 The Physical Mapping Problem

Our approach to the physical mapping problem is based on the generation of a hybridization profile for each clone in the contig library[7]. Each clone is scored for the presence or absence of specific probes resulting in the assignment of a digital *signature* to each clone. A *1* in a specific position in the clonal signature indicates hybridization to a specific probe whereas a *0* indicates absence of hybridization to a specific probe.

We formulate the problem of chromosome reconstruction as one of computing a clone ordering that minimizes the sum of differences between successive clonal signatures:
(1) The *Hamming* distance $d(C_i, C_j)$ between two clone signatures C_i and C_j is defined to be the measure of dissimilarity between their signatures.
(2) The total linking distance D for an ordering is defined as the sum of the Hamming distances between all pairs of successive clones: $D = \sum_{i=1}^{n-1} d(C_i, C_{i+1})$.
(3) The desired ordering is deemed to be that which results in minimization of D. 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 $\lim_{m\to\infty} prob(|D_m - D_0| > \epsilon) = 0$, i.e. D_m converges in probability to D_0 as the number of probes m in the clonal library grows[21].

The problem of coming up with an optimal ordering can be shown to be isomorphic to the classical NP–complete *Optimal Linear Ordering* (OLO) problem for which no polynomial-time algorithm for determining the optimal solution is known. Stochastic optimization algorithms such as simulated annealing[13] are capable of avoiding local optima in the solution space and producing solutions that are close to the global optimum in average polynomial time. In fact, our previous research showed simulated annealing to be very successful in generating high quality physical maps from hybridized clonal data[7].

1.2 Simulated Annealing

A typical simulated annealing algorithm (Figure 1) consists of three phases (1) Perturb Phase or Move Phase (2) Evaluate Phase and (3) Decide Phase as described below:
(1) Perturb Phase: Given an n-variable objective function of the form $f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$ to be minimized and a candidate solution $\mathbf{x}_i$, $\mathbf{x}_i$ is systematically perturbed to yield another candidate solution $\mathbf{x}_j$.
(2) Evaluate Phase: $\mathbf{x}_j$ is evaluated i.e. $f(\mathbf{x}_j)$ is computed. In our case, the total linking distance associated with the new candidate solution (i.e. new clone ordering) is computed.
(3) Decide Phase: If $f(\mathbf{x}_j) < f(\mathbf{x}_i)$ then $\mathbf{x}_j$ is accepted as the new candidate solution. If $f(\mathbf{x}_j) \geq f(\mathbf{x}_i)$ then $\mathbf{x}_j$ is accepted as the new candidate solution with probability p given by the Metropolis function: $p = \exp\left(-\left(f(\mathbf{x}_j) - f(\mathbf{x}_i)\right)/T_i\right)$ for a given value of temperature T_i whereas $\mathbf{x}_i$ is retained with probability $(1-p)$. The parameter T is referred to as the temperature of the system. 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. At sufficiently high temperatures, simulated annealing resembles a random search whereas at lower temperatures it acquires the characteristics of conventional hill-climbing search.

One of the drawbacks of a serial implementation of simulated annealing is that the annealing schedule necessary to obtain a solution close to the global optimum, is

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

COUNT_LIMIT = START_COUNT;
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;
  }
  COUNT_LIMIT = COUNT_LIMIT + COUNT_DELTA;
  Update the temperature using the annealing function
  T = A(T);
}
```

Figure 1: Outline of a typical simulated annealing algorithm

computationally intensive resulting in excessive run times. Parallel processing is one of the ways in which this drawback can be alleviated.

2 Parallel Processing

Parallel processing entails the utilization of multiple processors simultaneously to solve a given problem, thus reducing the time required to solve the problem when compared to serial processing. Parallel processing has emerged only very recently in the context of problems in computational biology, such as sequence comparison[9, 12, 14, 16], sequence alignment[8, 10], genetic mapping[17] and physical mapping[4].

2.1 Data Parallel Processing on the MasPar MP-2

Massively parallel systems such as the MasPar MP-2 can run an application several times faster than most serial systems provided that the application fits well in the massively parallel paradigm. This has prompted us to consider the MasPar MP-2 as a candidate multiprocessor platform for the design and implementation of a massively parallel simulated annealing (MPSA) algorithm. The MasPar MP-2 is a massively parallel distributed memory SIMD computer with the processing elements (PE's) interconnected in a toroidal 2-D mesh topology[5].

The MPL language[15] on the MasPar MP-2 is an extension of ANSI C with constructs that allow data parallel programming. Particularly, the *plural* data construct allows a variable to be replicated (with possibly different values) in the local memory of each PE. Since each PE has its own copy of a plural variable, updates to a plural variable can be accomplished with a single data parallel statement. Data parallelism in MPL stems from operations on plural variables. The MasPar MP-2 system offers extensive nearest neighbor communication on the 2-D toroidal mesh using the *xnet* primitive.

2.2 Parallel Simulated Annealing (PSA) Algorithms

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 – SIMD, MIMD, shared memory and distributed memory. Several parallelization strategies for simulated annealing have been well documented in the literature:
(A) *Functional parallelism*[20],
(B) *Control parallelism* with (i) speculative computation[19], (ii) parallel Markov chain generation using a systolic algorithm[1, 2], and (iii) multiple independent or periodically interacting searches[2].
(C) *Data parallelism* with (i) parallel evaluation of multiple moves with acceptance of a single move[6], (ii) parallel evaluation of multiple moves with acceptance of non-interacting multiple moves[11], and (iii) parallel evaluation and acceptance of multiple

moves[3].

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 simulated annealing on the MasPar MP-2 system.

3 MPSA on the MasPar MP-2

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 four models of a massively parallel simulated annealing (MPSA) algorithm based on the distribution of Markov chains on the MasPar MP-2 PE array and the synchronization method used. These models incorporate the parallelization strategies B(ii) and C(iii) discussed in Section 2.2 and are described below:

(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.

In the NILM MPSA algorithm, each PE runs an independent version of the simulated annealing algorithm. Thus 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 (Figure 1) is run synchronously on each PE. At each temperature value each PE iterates through the *perturb–evaluate–accept* cycle `COUNT_LIMIT` number of times concurrently along with all the other PE's. Two perturbation strategies were implemented and tested in the context of the NILM MPSA algorithm:

(1) Pairwise Clone Exchange: In a given ordering (i.e. permutation) σ_1, the positions of two randomly chosen clones are interchanged to generate a new permutation σ_2.

(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. The resulting clone ordering σ_2 is $(C_1, ..., C_{i-1}, C_j, C_{j-1}, ..., C_{i+1}, C_i, C_{j+1}, ...C_n)$. This strategy preserves any pre-existing ordering within the reversed block.

The perturbation function uses a parallel random number generator with distinct seeds for each PE in order to ensure the independence of the resulting parallel Markov chains of solution states. The evaluation function and the Metropolis decision function are concurrently executed on each PE until $T = T_{min}$. The best solution is selected from among all the candidate solutions on the individual PE's. The NILM MPSA model is essentially that of multiple independent searches.

In the PILM MPSA algorithm proceeds in a manner identical to the NILM MPSA algorithm. The only difference is that just before the parameter T is updated using the annealing function, the best solution from among those in all the PE's is selected and duplicated on all the other PE's. The PILM MPSA model is essentially that of multiple periodically interacting searches.

In the NIDM MPSA algorithm, the candidate solution, i.e. a single Markov chain is distributed over a group of neighboring processors, i.e. a PE cluster. The PE clusters are non-overlapping rectangular submeshes of equal size with dimensions X_{width} and Y_{width}. The perturbations in the NIDM MPSA algorithm are carried out in 3 distinct phases as described below:

(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.

(B) Inter–PE Perturbations along the Cluster Rows: The PE's are paired along the rows in each PE cluster. The evaluation of the perturbation is carried out concurrently by each PE in the PE pair. The result of the evaluation is collected in one of the PE's in each PE pair which designated as the *master*. The *master* PE in each pair decides whether or not to accept the proposed perturbation using the Metropolis function.

(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.

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 selected 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 is the most general of the four MPSA models for the MasPar MP-2 described in this paper. The PIDM MPSA algorithm is outlined in Figure 2. 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 $Y_{width} = 1$.

3.1 Heuristic Enhancements

(A) Annealing Schedule: The algorithms outlined in Figures 1 and 2 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 found that an adaptive annealing schedule yields better results. In the adaptive annealing schedule, the temperature T is updated if either of the following conditions holds:

(i) The total number of perturbations at a given temperature equals `COUNT_LIMIT`.

(ii) 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 results

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

T = T_max;
COUNT_LIMIT = START_COUNT;
while (T >= T_min)
{
  for (count=1; count <= COUNT_LIMIT;  count = count + 1)
  {
    Phase 1. Perturb existing solution(s) where each solution is distributed
             over a PE cluster to generate multiple candidate solutions
             concurrently:
             (a) Intra-Node Perturbations: Randomly perturb the existing
                  within each PE;
             (b) Inter-Node Perturbations along the PE cluster rows:
                 Swap clone blocks between disjoint PE pairs along the
                 cluster rows;
             (c) Inter-Node Perturbations along the PE cluster columns:
                 Swap clone blocks between disjoint PE pairs along the
                 cluster columns;

    Phase 2. Evaluate Candidate Solution(s):
             Compute f_delta, the change in cost for each candidate
             solution concurrently in each PE cluster using the
             pre-defined objective function f;

    Phase 3. Parallel Accept:
              Concurrently accept the perturbed candidate solutions in
              each PE cluster 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 maximum number of iterations at each temperature:
           COUNT_LIMIT = COUNT_LIMIT + COUNT_DELTA;

  Phase 7. Update the temperature using the annealing function:
           T = A(T);
}
```

Figure 2: The PIDM MPSA algorithm

in a decrease in the cost function.
In spite of the overhead, the adaptive annealing schedule is more efficient. The total number of iterations at higher T values are reduced to only 0.1% - 0.2% of the maximum number of iterations at a given value of T i.e. `COUNT_LIMIT`, thus speeding up the algorithm. At lower T values, 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.

If the same linking distance repeats for a certain number of consecutive temperature values (typically 3) in the adaptive annealing schedule, the algorithm is assumed to have reached a global optimum. The algorithm is terminated, thus preventing it from having to run (somewhat needlessly) until the final temperature value is reached.

(B) Stochastic Synchronization: The synchronization process can be made stochastic by using the Boltzmann decision function. The best solution with linking D is duplicated on the ith PE/PE cluster containing a solution with linking distance D_i with a probability P_B, where $P_B = 1/(1 + \exp((D - D_i)/T))$.

4 Experimental Results

The massively parallel simulated annealing algorithms with various combinations of perturbation methods and heuristic enhancements were implemented on a 2048 processor MasPar MP-2 system at the University of Georgia, Athens, Georgia. Due to the limited PE memory, *on–the–fly* computation of inter-clonal distances instead of a distance matrix look-up 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 PE's utilized and `COUNT_LIMIT` was kept constant.

The various MPSA algorithms were run on a real clone data set derived from chromosome IV of the fungus *Aspergilus Nidulans*[18] and also a synthetic clone data set. The real data set consisted of 592 clones with a clonal signature length of 115 bits whereas the synthetic clone data set consisted of 225 clones with a clonal signature length of 50 bits. On comparison between the various MPSA algorithms we observed the following:
(1) The PILM MPSA algorithm showed the best convergence characteristics and also yielded a clone ordering with the least overall linking distance. Figure 3 shows the linking distance of the real clone data set as a function of execution time of the PILM MPSA algorithm (plotted on a logarithmic scale) with the number of PE's as a variable parameter. Figure 4 shows the logarithm of the speedup of the PILM MPSA algorithm as a function of linking distance for the real clone data set. With 2048 processors the speedup obtained was approximately 1000 for linking distances close to the optimal linking distance.
(2) In contrast, the PIDM MPSA algorithm and NIDM MPSA algorithm exhibited a tendency to get trapped in a local minimum. This could be attributed to the fact that the distribution of data across a PE cluster alters the state transition probabilities

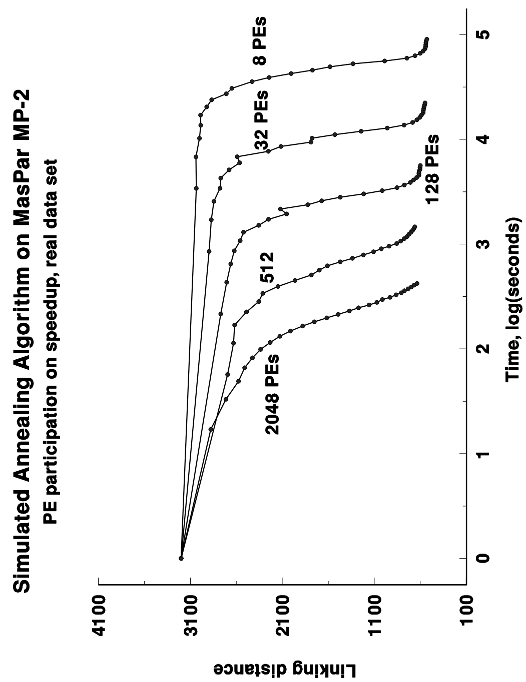

Figure 3: Linking distance as a function of the log of execution time for varying no. of PE's, real data set, $\lambda = 1,024,000$

associated with the Markov chain of solution states generated by the serial simulated annealing algorithm.

(3) The intra-PE perturbation strategy based on clone block reversal performed better than the one based on clone exchange for all the four MPSA algorithms that we evaluated (Table 1). 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) The use of stochastic synchronization with the Boltzmann decision function did not substantially improve the convergence characteristics of either the PILM MPSA algorithm or the PIDM MPSA algorithm. In fact, in the case of the PILM MPSA algorithm, a slight degradation of performance was noticed when deterministic synchronization was replaced by stochastic synchronization (Table 2).

5 Conclusions

In this paper, we have designed and analyzed four different models for a massively parallel simulated annealing (MPSA) algorithm for chromosome reconstruction via clone ordering. The algorithms were implemented on a 2048 processor MasPar MP-2 at the University of Georgia. Of the four MPSA algorithms that were implemented, 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

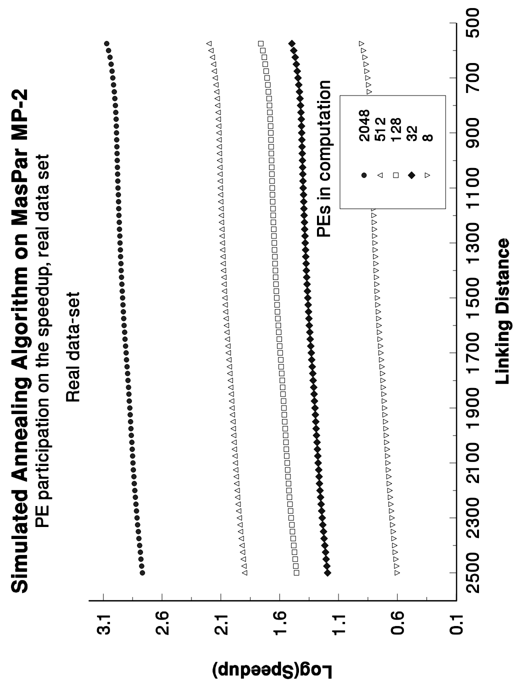

Figure 4: Logarithm of speedup as a function of linking distance for varying no. of PE's, real data set, $\lambda = 1,024,000$

Table 1: PILM MPSA Algorithm: Comparison of pairwise clone exchange and clone block reversal perturbation methods, No. of PE's used = 2048

D: Final Linking Distance				
Data Set	Perturbation Strategy	λ		
		204,800	409,600	819,200
Real	Pairwise Clone Exchange	583	556	529
	Clone Block Reversal	560	531	504
Artificial	Pairwise Clone Exchange	1520	1466	1432
	Clone Block Reversal	1419	1308	1292

Table 2: Comparison of synchronization methods for the PILM MPSA algorithm with 2048 PE's, T: execution time in secs., D: Final Linking Distance

Data Set	Synchronization Method	λ	T	D
Real	Deterministic	409,600	1418.0	583
		2,048,000	5094.0	507
	Stochastic	409,600	1583.0	632
		2,048,000	5551.0	523
Artificial	Deterministic	409,600	1223.0	1308
		2,048,000	6015.0	1289
	Stochastic	409,600	1395.0	1403
		2,048,000	6506.0	1334

2048 processor MasPar MP-2 system exhibited a speedup of approximately 1000. Our results have shown that distribution of clonal data across the PE's in the MasPar MP-2 leads to degradation in performance. However, as clonal data sets increase in size, MPSA models that work with distributed data might be the only feasible alternative and hence merit further investigation.

Owing to the limited local PE memory, the MPSA algorithms presented here had to incorporate *on-the-fly* computation of inter-clonal distances. Distance matrix lookup techniques could have improved the execution times of our algorithms by an order of magnitude if sufficient local PE memory was available. Although computationally efficient, straightforward distance matrix lookup techniques are not scalable in terms of memory since the memory requirements grow quadratically with respect to the clonal data size. Data structures that are scalable in terms of memory and simultaneously allow time-efficient retrieval need to be explored. Also, the clonal data set is typically sparse; characterized by a large number of 0's and very few 1's in the clonal signature. Sparse matrix representation techniques in the context of representation of clonal data need to be explored to allow for more efficient storage and consequently more scalable MPSA algorithms.

Acknowledgements

The support of the National Science Foundation (NSF BIR 94-22896, CCR-8717033 and CDA-8820544) and the research instrumentation grant by MasPar Computer Corporation are gratefully acknowledged.

References

[1] 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. *Lecture Notes in Comp. Sci: Proc. 3rd Ann. Symp. Theo. Aspects of Comp. Sci.* 210, 87–97. Springer-Verlag Inc., Berlin, Germany.

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

[3] Banerjee, P., Jones, M.H., and Sargent J.S. 1990. Parallel Simulated Annealing Algorithms for Cell Placement on the Hypercube Multiprocessor. *IEEE Trans. Par. and Dist. Sys.* 1, 91–106.

[4] Bhandarkar, S.M., and Arnold, J. 1994. Parallel Simulated Annealing on the Hypercube for Chromosome Reconstruction. *Proc. IMACS 14th World Cong. Comp. and Appl. Math.* 3, 1109–1112.

[5] Blank, T. 1990. The MasPar MP-1 Architecture. *Proc. IEEE COMPCON 90.* Feb. 27 - Mar. 2, San Francisco, CA. 20–24.

[6] Casotto, A., Romeo, F., and Sangiovanni–Vincentelli, A. 1987. A Parallel Simulated Annealing Algorithm for the Placement of Macro Cells. *IEEE Trans. Comp. Aided Des.* 9, pp. 838–847.

[7] 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.

[8] Date, S., Kulkarni, R., Kulkarni, B., Kulkarni-Kale, U., and Kolaskar, A.S. 1993. Multiple Alignment of Sequences on Parallel Computers. *CABIOS*. 9, 397–402.

[9] Huang, X., Miller, W., Schwartz, S., and Hardison, R.C. 1992. Parallelization of a Local Similarity Search Algorithm. *CABIOS*. 8, 155–166.

[10] Ishikawa, M., Toya, T., Hoshida, M., Nitta, K., Ogiwara, A., and Kanehisa, M. 1993. Multiple Sequence Alignment by Parallel Simulated Annealing. *CABIOS*. 9, 267–273.

[11] Jayaraman, R., and Rutenbar, R. 1987. Floor Planning by Annealing on a Hypercube Multiprocessor. *Proc. IEEE Intl. Conf. Comp. Aided Des.* 346–349.

[12] Jones, R. 1992. Sequence Pattern Matching on a Massively Parallel Computer. *CABIOS*. 8, 377–384.

[13] Kirkpatrick, S., Gelatt, C. Jr., and Vecchiet, M. 1983. Optimization by Simulated Annealing. *Science*. 220(4598), 498–516.

[14] 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. *CABIOS*. 9, 701–708.

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

[16] 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. *CABIOS*. 8(2), 167–175.

[17] Miller, P.L., Nadkarni, P.M., and Bercovitz, P.A. 1992b. Harnessing Networked Workstations as a Powerful Parallel Computer: A General Paradigm Illustrated Using Three Programs for Genetic Linkage Analysis. *CABIOS*. 8(2), 141–147.

[18] 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.*, 91, 11094–11098.

[19] Witte, E.E., Chamberlain, R.D., and Franklin, M.A. 1991. Parallel Simulated Annealing using Speculative Computation. *IEEE Trans. Par. and Dist. Sys.* 2(4), 483–494.

[20] Wong, C.P., and Fiebrich, R.D. 1987. Simulated Annealing-based Circuit Placement on the Connection Machine System. *Proc. Intl. Conf. Comp. Des.* 78–82.

[21] Xiong, M., Chen, H.J., Prade, R.A., Wang, Y., Griffith, J., Timberlake, W.E., and Arnold, J. 1995. On the Consistency of a Physical Mapping Method to Reconstruct a Chromosome *In Vitro*. *Genetics*. in press.