A Parallel Genetic Algorithm for Physical Mapping of Chromosomes

Suchendra M. Bhandarkar and **Jinling Huang**
Department of Computer Science
The University of Georgia
Athens, Georgia 30602–7404, USA
suchi@cs.uga.edu, jihuang@uga.edu

Jonathan Arnold
Department of Genetics
The University of Georgia
Athens, Georgia 30602–7223, USA
arnold@uga.edu

Abstract

Physical map reconstruction in the presence of errors is a central problem in genetics of high computational complexity. A parallel genetic algorithm for a maximum likelihood estimation-based approach to physical map reconstruction is presented. The estimation procedure entails gradient descent search for determining the optimal spacings between probes for a given probe ordering. The optimal probe ordering is determined using a genetic algorithm. A two-tier parallelization strategy is proposed wherein the gradient descent search is parallelized at the lower level and the genetic algorithm is simultaneously parallelized at the higher level. Implementation and experimental results on a network of shared-memory symmetric multiprocessors (SMPs) are presented. The genetic algorithm is seen to result in physical maps with fewer contig breaks when compared to simulated Monte Carlo algorithms such as simulated annealing and the large-step Markov chain algorithm.

1 Introduction

In this paper we present the design and implementation of a parallel Genetic Algorithm (GA) for the physical mapping of chromosomes. The physical mapping protocol is based on *sampling without replacement* [3] where a maximal set $\mathcal{P}$ of non-overlapping equal-length clones from a library is selected as the probe set. The remaining clones $\mathcal{C}$ in the library are hybridized to the probe set resulting in a digital hybridization signature for each clone. The clone-probe overlap pattern is represented by a binary hybridization matrix H where $H_{ij} = 1$ if the ith clone hybridizes to the jth probe and $H_{ij} = 0$ otherwise. If the probes in $\mathcal{P}$ are ordered with respect to their position along a chromosome, then by selecting from H a common overlapping clone for each pair of adjacent probes, a minimal set of clones and probes that covers the entire chromosome (i.e., a minimal tiling) can be obtained. The minimal tiling in conjunction with the sequencing of each individual clone/probe in the tiling and a sequence assembly procedure that determines the overlaps between successive sequenced clones/probes in the tiling [9] can then be used to reconstruct the DNA sequence of the entire chromosome. In reality, H could be expected to contain false positives and false negatives. In the case of a false positive, $H_{ij} = 1$ when in fact $H_{ij} = 0$. Conversely, a false negative occurs if $H_{ij} = 0$ when in fact $H_{ij} = 1$. In this paper, we confine ourselves to errors in the form of false positives and false negatives that are statistically independent of one another. Hybridization errors due to DNA repeats are not explicitly modeled, but are treated as multiple isolated cases of false positives.

We briefly describe a maximum likelihood (ML) estimator [3, 10] which determines the optimal ordering of probes in the probe set $\mathcal{P}$ and the optimal inter-probe spacings under a probabilistic model of hybridization errors consisting of false positives and false negatives. The optimal probe ordering and optimal inter-probe spacings are deemed to be the ones that maximize the likelihood or probability of occurrence of the observed hybridization matrix H. The estimation procedure involves a combination of continuous and discrete (i.e., combinatorial) optimization where determining the optimal probe ordering entails discrete optimization whereas determining the optimal inter-probe spacings for a particular probe ordering entails continuous optimization. We propose a two-tier parallelization strategy for efficient implementation of the above estimator. The upper level comprises of parallel discrete optimization using a GA whereas the lower level comprises of a parallel Conjugate Gradient Descent (CGD) procedure for continuous optimization. The resulting physical mapping algorithm is implemented on a network of shared-memory symmetric multiprocessors (SMPs) using a combination of the Message Passing Interface (MPI) environment [14] and multithreaded programming [1]. Convergence, speedup and scalability characteristics of the parallel GA are discussed.

2 Mathematical Formulation of the ML Estimator

The ML-based physical mapping problem can be formally stated as follows. Given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of n probes and a set $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ of k clones generated using the sampling-without-replacement protocol and the $k \times n$ clone-probe hybridization matrix H containing both false positives and false negatives with predefined probabilities, reconstruct the ordering $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the probes and also the spacing $Y = (Y_1, Y_2, \ldots, Y_n)$ between the probes such that probability or likelihood of occurrence of the observed hybridization matrix H is maximized.

The likelihood function can be shown to be given by [3, 10]

$$P(H \mid \Pi, Y) = \prod_{i=1}^{k} C_i \{R_i - \sum_{j=1}^{n+1} (a_{i,\pi_j} - 1)(a_{i,\pi_{j-1}} - 1) min(Y_j, M)\} \quad (1)$$

where

$$a_{i,j} = \begin{cases} \frac{\eta}{(1-\rho)} & \text{if } h_{i,j} = 0 \text{ and } j = 1, \ldots, n \\ \frac{(1-\eta)}{\rho} & \text{if } h_{i,j} = 1 \text{ and } j = 1, \ldots, n \\ 0 & \text{otherwise,} \end{cases}$$

$$C_i = \frac{\rho^{p_i}(1-\rho)^{(n-p_i)}}{N - M}$$

and

$$R_i = N - nM + M \sum_{j=1}^{(n-1)} a_{i,\pi_j} a_{i,\pi_{j+1}}$$

given the following parameters
N : Length of the chromosome,
M : Length of a clone/probe,
n : Number of probes,
k : Number of clones,
ρ : Probability of false positive,
η : Probability of false negative,
$H = ((h_{i,j}))_{1 \le i \le k, 1 \le j \le n}$: clone-probe hybridization matrix,
where
$h_{i,j} = \begin{cases} 1 & \text{if clone } C_i \text{ hybridizes with probe } P_j \\ 0 & \text{otherwise,} \end{cases}$
H_i : ith row of the hybridization matrix,
$\Pi = (\pi_1, \ldots, \pi_n)$: permutation of $\{1, 2, \ldots, n\}$ which denotes the probe labels in the ordering when scanned from left to right along the chromosome,
$p_i = \sum_{j=1}^{n} h_{i,j}$: number of 1's in H_i,
$P = \sum_{i=1}^{k} p_i$: total number of 1's in H, and
$Y = (Y_1, Y_2, \ldots, Y_n)$: vector of inter-clone spacings where Y_i is the spacing between the right end of $P_{\pi_{i-1}}$ and the left end of P_{π_i} $(2 \le i \le n)$, and Y_1 is the spacing between the left end of P_{π_1} and the left end of the chromosome, and
$\mathcal{F} \subseteq \mathcal{R}^n$: set of feasible interprobe spacings $Y = \{Y_1, \ldots, Y_n\}$ such that $Y_i \ge 0$, $1 \le i \le n$ and $N - nM - \sum_{i=1}^{n} Y_i \ge 0$.
The goal therefore is to determine Π and Y that maximize $P(H \mid \Pi, Y)$ (equation (1)), that is determine $(\hat{\Pi}, \hat{Y})$ where

$$(\hat{\Pi}, \hat{Y}) = \arg \max_{(\Pi, Y)} P(H \mid \Pi, Y) \quad (2)$$

Alternatively we could consider the negative log-likelihood (NLL) function $f(\Pi, Y)$ given by

$$f(\Pi, Y) = -\ln P(H \mid \Pi, Y) \quad (3)$$

Since $\ln x$ is a monotonically increasing function of x for all $x > 0$, it follows that

$$(\hat{\Pi}, \hat{Y}) = \arg \max_{(\Pi, Y)} P(H \mid \Pi, Y) = \arg \min_{(\Pi, Y)} f(\Pi, Y) \quad (4)$$

2.1 Computation of the ML Estimate

Computing the values of $\hat{\Pi}$ and $\hat{Y}$ involves a two stage procedure:
Stage 1: First the optimal spacing $\hat{Y}_\Pi$ for a given probe ordering Π is determined, i.e., $\hat{Y}_\Pi = (\hat{Y}_1, \ldots, \hat{Y}_n)$ such that for a given Π,

$$f(\Pi, \hat{Y}_\Pi) = \min_{Y} f(\Pi, Y) = \min_{Y} f_\Pi(Y) \quad (5)$$

The minimum is taken over all feasible solutions Y that satisfy the constraints $Y_i \ge 0$; $i = 1, \ldots, n$ and $\sum_{i=1}^{n} Y_i \le N - nM$.
Stage 2: $\hat{\Pi}$ is determined for which,

$$f(\hat{\Pi}, \hat{Y}_{\hat{\Pi}}) = \min_{\Pi} f(\Pi, \hat{Y}_\Pi) = \min_{\Pi} f_{\hat{Y}_\Pi}(\Pi) \quad (6)$$

The minimum is taken over all Π where Π is a permutation of $\{1, \ldots, n\}$. The resulting values of $\hat{\Pi}$ and $\hat{Y}_{\hat{\Pi}}$ are termed the ML estimates of the true probe ordering and the inter-probe spacings, respectively.

2.1.1 Computation of $\hat{Y}_\Pi$

It can be shown that $f_\Pi(Y)$ is convex in a finite number of closed regions in $\mathcal{F} \subseteq \mathcal{R}^n$ and therefore possesses a unique local minimum which is also a global minimum [3, 10]. Consequently this minimum can be

reached using continuous local search-based techniques such as the steepest descent search [7]. The steepest descent search is a simple iterative procedure which consists of three steps: (i) Determine the initial value of Y, (ii) Compute the downhill gradient at Y given by $U = -\nabla f(\Pi, Y)$ and (iii) Update the current value of Y using the computed value of the downhill gradient $Y_{new} = Y_{old} + sU$. The optimal value of s, say s^* such that $f(\Pi, Y + s^*U) = \min_s f(\Pi, Y + sU)$ is computed using the bisection procedure. Steps (ii) and (iii) are repeated until the gradient vanishes, or in practice, until the gradient magnitude is less than a prespecified threshold.

In this paper, we have used the conjugate gradient descent (CGD) search instead of the steepest descent search since the former is known to be one of the fastest in the class of gradient descent-based optimization methods [7]. The CGD search is very similar to the steepest descent procedure except that instead of consistently following the local downhill gradient direction, a set of n mutually orthonormal (i.e., conjugate) direction vectors are generated from the downhill gradient vector where n is the dimensionality of the solution space [7]. Unlike the steepest descent algorithm, the CGD algorithm guarantees convergence to a local minimum within n steps.

2.1.2 Computation of $\hat{\Pi}$

Determining the optimal clone ordering $\hat{\Pi}$, entails a combinatorial search through the discrete space of all possible permutations of $\{1, \ldots, n\}$. The problem of coming up with such an optimal ordering is isomorphic to the classical NP-complete *Traveling Salesman Problem* (TSP) for which no polynomial-time algorithm for determining the optimal solution is known [11]. One could use a Monte Carlo search method such as Simulated Annealing (SA) [6] or the Large Step Markov Chain (LSMC) [12] or an evolutionary search method [13], all of which are known to be robust in the presence of local optima in the solution space and give near-optimal solutions in average polynomial time. Our previous work in using SA and LSMC indicated that whereas these methods were robust to the presence of local optima, their parallel versions were not scalable with respect to speedup and efficiency of parallelism with an increasing number of processors [4]. Consequently, we decided to investigate evolutionary search methods and their parallel implementation on a network of shared-memory symmetric multiprocessors (SMPs).

A typical evolutionary method begins with an initial ensemble or population of candidate solutions and iterates through a number of generations before reaching a locally optimal solution. In each iteration or generation, the solutions in the current population are subject to evolutionary operators that mimic their biological counterparts [13]. The Genetic Algorithm (GA) is one of the most widely used evolutionary methods. A typical GA uses the selection, crossover and mutation operations on the candidate solutions in the current population to generate solutions for the following generation.

The selection operator uses a *roulette-wheel* procedure to select candidate solutions with probability in direct proportion to their fitness values [13]. The fitness function is the negative of the NLL objective function in equation (3) so that solutions with lower objective function values have higher fitness values and conversely. During the crossover operation the solutions selected by the roulette-wheel procedure are treated as parental chromosomes, and child chromosomes are generated by exchanges of parental chromosomal segments. The crossover operation enables large-scale exploration of the search space [13]. We have used an improvised version of the heuristic crossover operator originally proposed by Jog *et al.* [8] in their GA for the TSP. In a typical GA for the TSP, the mutation operator is a random local 2-opt perturbation [11] where the ordering within a randomly chosen block of probes in the current probe ordering is reversed. The GA is deemed to have converged when the best solution in the population has not improved within a certain number of successive generations.

Our previous experience with the GA has shown that incorporation of a *stochastic* hill-climbing search mechanism greatly improves the asymptotic convergence of the GA to a near-globally optimal solution [2]. As a consequence, the GA was enhanced with the incorporation of a stochastic hill-climbing search similar to that of the LSMC algorithm [12]. The stochastic hill-climbing search in the LSMC algorithm incorporates an exhaustive local search using the 2-opt perturbation coupled with the Metropolis or Boltzmann decision function [4]. The GA-LSMC hybrid algorithm is outlined in Figure 1. Note that the members of the population in each generation are locally optimal solutions under the 2-opt perturbation. This is ensured by a mutation operator which is a combination of the non-local *double-bridge* perturbation [12] and an exhaustive local search using the 2-opt perturbation.

3 Parallel Computation of the ML Estimator

We propose a two-tier parallel computation of the ML estimator corresponding to the two stages of optimization.

The GA-LSMC Hybrid Algorithm with Stochastic Population Replacement:

1. Create an initial population of locally optimal solutions using the *double-bridge* perturbation and an exhaustive local 2-opt search; (This ensures that all the members in the initial population are locally optimal solutions)
2. While not converged do
   (a) Select two parents using the roulette wheel selection procedure;
   (b) Apply the heuristic crossover to the selected parents to create an offspring S;
   (c) Perform exhaustive local 2-opt search on S to yield a new locally optimum solution S';
   (d) With probability p_m perform a mutation on S' using a double-bridge perturbation followed by exhaustive local 2-opt search to yield a new locally optimum solution S^*;
   (e) Evaluate the NLL objective function at S^* (if mutation is performed) or at S' (if mutation is not performed) using conjugate gradient descent search;
   (f) Compute $\Delta f = f(S^*) - f(P)$ (if mutation is performed) or $\Delta f = f(S') - f(P)$ (if mutation is not performed) where P is the less fit of the two parents;
   (g) Replace P with S^* (if mutation is performed) or S' (if mutation is not performed) with probability p_r computed using the Boltzmann function $p_r = \frac{1}{1+\exp\left(\frac{\Delta f}{T}\right)}$;
   (h) Update p_m;
   (i) Update temperature using the annealing function $T = A(T)$;
   (j) Check for convergence;
3. Output the best solution in the population as the final solution;

Figure 1: Outline of the GA-LSMC hybrid algorithm with stochastic replacement

Level 1: Parallel computation of the optimal inter-probe spacing $\hat{Y}_\Pi$ for a given probe ordering Π that minimizes $f(\Pi, \hat{Y}_\Pi)$. This entails parallelization of the gradient descent search procedure for constrained continuous optimization.
Level 2: Parallel computation of the optimal probe ordering $\hat{\Pi}$ for which $f(\hat{\Pi}, \hat{Y}_{\hat{\Pi}})$ is minimum. This entails parallelization of the GA-LSMC hybrid algorithm for discrete optimization.
The parallel algorithms were implemented on a cluster of SMPs using a combination of MPI and multithreaded programming.

3.1 Parallel Evolutionary Search

Our approach to parallelizing the GA is based on partitioning the population amongst the available processors [5]. Each processor is responsible for searching for the best solution within its subpopulation. This is tantamount to performing multiple concurrent searches within the search space [5]. The parallel GA (pGA) was implemented on a distributed-memory platform comprising of a network of SMPs using MPI and the master-slave computation model. The master process runs on one of the SMPs within the SMP cluster. The master process reads in the problem data, creates the initial population, spawns slave processes on all the SMPs (including its own) and divides the initial population amongst the slave processes. Each slave process runs a serial version of the GA-LSMC hybrid algorithm (Figure 1) on its subpopulation concurrently with the other slave processes. The slave processes periodically send the solutions within their subpopulation to the master process. The master process on receipt of the solutions from all the slave processes, checks for convergence, mixes the solutions at random and redistributes the population amongst the slave processes. The periodic mixing and redistribution of the population prevents a slave process from premature convergence to a local optimum after having exhausted all the genetic variation within its subpopulation. The master process deems the pGA to have converged if the best solution in the overall population has not changed over a certain number of successive generations.

3.2 Parallel Gradient Descent Search

Due to its inherent sequential nature, we deemed data parallelism to be appropriate for the parallel CGD algorithm. The Y and U vectors are distributed amongst the different processors within an single SMP and each processor performs the gradient vector computation and updates to the inter-probe spacing vector using its local subvectors Y_{loc} and U_{loc} concurrently with the other processors within the SMP. Here, $|Y_{loc}| = |Y|/N_P$ and $|U_{loc}| = |U|/N_P$ where N_P is the number of processors within each SMP. A multithreaded programming approach [1] was used with a single thread running on a single processor within the SMP. Since the individual subvectors have to be periodically distributed amongst the processors and also periodically gathered to compute a global value for s during the bisection procedure, the threads have to be periodically synchronized using a barrier. Updates to global values in shared memory (such as the global sum of the vector components) are controlled using mutex (mutual exclusion) locks.

3.3 Two-tier Parallelization

In order to ensure a scalable implementation, two tiers of parallelism were incorporated in the computation of the ML estimator. The finer or lower level

of parallelism pertains to the computation of $\hat{Y}$ for a given probe ordering Π using the parallel multithreaded CGD algorithm for continuous optimization. The coarser or upper level of parallelization pertains to the computation of $\hat{\Pi}$ using the pGA for discrete optimization. The multithreaded CGD algorithm is embedded within each of the pGAs. The parallelization of the CGD algorithm at the finer level is transparent to the pGA at the coarser level. When the parallel CGD procedure is invoked from within the master or slave pGA process, a new set of slave CGD processes (i.e., threads) is spawned on the available processors within an SMP. The master CGD thread runs on the same processor as the pGA process (master or slave). The master and slave CGD threads cooperate to evaluate and minimize the value of $f(\Pi, \hat{Y}_\Pi)$. Once $f(\Pi, \hat{Y}_\Pi)$ is minimized, the slave CGD threads terminate and the corresponding processors within an SMP are available for future computation.

4 Experimental Results

The parallel algorithms were implemented on a dedicated cluster comprising of 8 nodes where each node is a shared-memory symmetric multiprocessor (SMP) running SUN Solaris-x86. Each SMP comprises of 4 700MHz Pentium III Xeon processors with 1 MB cache per processor and 1GB of shared memory. The programs were tested with simulated clone-probe hybridization data [3, 10] as well as real data from *cosmid2* ($n = 109$, $k = 2046$) and *cosmid3* ($n = 111$, $k = 1937$) of the fungal genome *Neurospora crassa* being currently mapped in the Department of Genetics at the University of Georgia.

The parallel (i.e., multithreaded) versions of the CGD algorithm and exhaustive local search algorithm were tested on simulated chromosomal data sets with a varying number of probes and clones $(n, k) = (50,300)$, $(200,1300)$ and $(500,3250)$ and on real data from *cosmid2*. The performance of these parallel algorithms is detailed in [4]. For both algorithms, the payoff in parallelization was observed to be better realized for larger values of n (i.e., larger problem sizes). However, in the case of the CGD algorithm, the speedup was observed to saturate much faster with increasing n compared to the exhaustive local search algorithm. This is on account of the higher overhead of inter-thread synchronization in the case of the multithreaded CGD algorithm due to frequent synchronization between master and slave threads. In contrast, in the case of the parallel exhaustive local search algorithm, there is very infrequent synchronization amongst the slave threads or between the master thread and the slave threads. Since each slave thread performs an independent search of the space of 2-opt perturbations, the only synchronization needed is at the beginning and end of the search process.

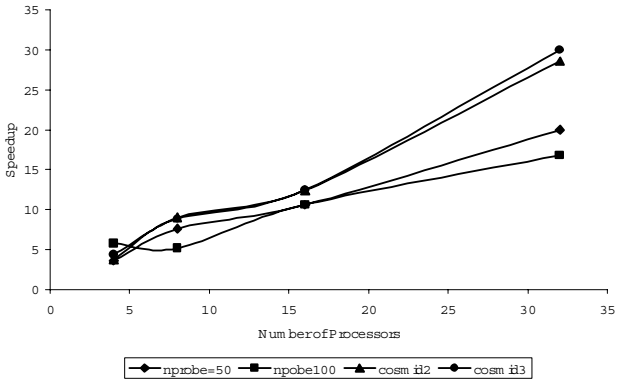
The pGA was implemented with the following parameters: the population size $N_{pop} = 40$, the initial temperature $T_{init} = 1$, the annealing factor $\alpha = 0.95$ in the geometric annealing schedule $T_{next} = \alpha \cdot T_{prev}$, the maximum number of trials before the population is mixed $max\_trials = N_{pop}$ and the convergence criterion used was the fact that no member in the population was replaced for two successive generations. The mutation probability p_m was set dynamically based on the population replacement rate. The value of p_m is kept low when the population replacement rate is sufficiently high and vice versa.

The results of the serial GA-LSMC hybrid algorithm were compared with those of the serial SA and serial LSMC algorithm. The serial GA-LSMC hybrid algorithm was seen to consistently yield lower NLL values compared to the SA and LSMC algorithms on both, artificial and real data. The number of probe suborderings (i.e., contigs) recovered by the GA-LSMC, SA and LSMC algorithms on the synthetic data sets was also compared. The GA-LSMC algorithm was seen to consistently yield fewer and longer contigs than the SA and LSMC algorithms suggesting that the GA-LSMC algorithm is capable of yielding solutions of higher quality.

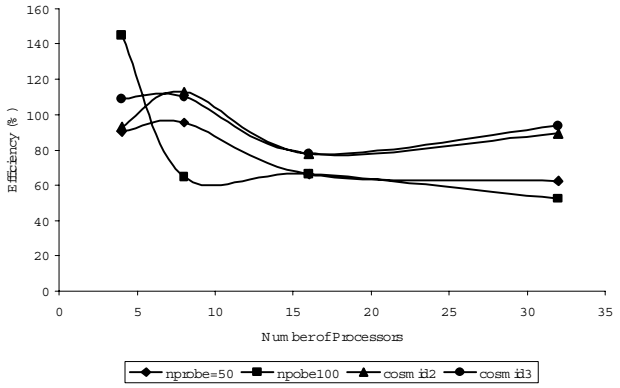
The performance of the pGA is summarized in Figure 2. The pGA was seen to exhibit superlinear speedup in some instances. This can be attributed in part to the fact that with a larger number of processors, the population per processor decreases to the point where it can reside entirely in cache. For a small number of processors, this caching effect can overcome the inter-processor communication and synchronization overhead resulting in super-linear speedup. Also, the efficiency values are observed to be higher for larger problem instances (larger values of n and k) for a given value of N_{proc} (the total number of processors). The efficiency values also exhibit an overall declining trend for increasing values of N_{proc} for a given problem instance implying the dominance of the inter-processor communication and synchronization overhead in the overall execution time. These observations are in conformity with the general expectations regarding the performance of the pGA.

5 Conclusions and Future Directions

In this paper we presented a GA-based physical mapping algorithm for maximum likelihood (ML) estimation-based physical map reconstruction. The



(a)



(b)

Figure 2: Parallel GA with stochastic replacement: (a) Speedup curves, (b) Efficiency curves

ML estimate reconstructs the optimal probe ordering and optimal inter-probe spacings. The estimation procedure was shown to entail continuous optimization for determining the optimal inter-probe spacings for a given probe ordering and combinatorial optimization for determining the optimal probe ordering. A two-tier parallelization strategy was proposed wherein the CGD search algorithm for continuous optimization is parallelized at the lower level and the GA-LSMC hybrid algorithm for combinatorial optimization is simultaneously parallelized at the higher level. The parallel ML estimation algorithm was shown to amenable to efficient implementation on a network of SMPs where the CGD search is parallelized on a single SMP using shared-memory, multithreaded programming whereas the GA-LSMC algorithm is parallelized on the SMP network using the distributed-memory, message-passing-based programming paradigm within the MPI environment. Future research will investigate extensions of the maximum likelihood function that also encapsulate errors due to repeat DNA sequences in addition to false positives and false negatives. Future research will explore the parallelization of the ML estimator on a heterogeneous platform such as a network of SMPs that differ in processing speeds and memory capacity, since that is a scenario that is more likely to be encountered in the real world.

Acknowledgments: This research was supported in part by an NRICGP grant by the US Department of Agriculture and a Research Instrumentation grant by the National Science Foundation.

References

[1] G. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison Wesley Pub. Co., Reading, MA, 2000.

[2] S.M. Bhandarkar and H. Zhang, Image Segmentation using Evolutionary Computation, *IEEE Trans. Evolutionary Computation*, Vol. 3, No. 1, pp. 1–21, April 1999.

[3] S.M. Bhandarkar, S.A. Machaka, S.S. Shete and R.N. Kota, Parallel Computation of a Maximum Likelihood Estimator of a Physical Map, *Genetics*, Vol. 157, No. 3, pp. 1021–1043, March 2001.

[4] S.M. Bhandarkar, J. Huang and J. Arnold, Parallel Monte Carlo Methods for Physical Mapping of Chromosomes, *Proc. IEEE Bioinformatics Conference*, Stanford University, Palo Alto, CA, pp. 64–75, August 2002.

[5] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers, Boston, MA, November 2000.

[6] S. Geman and D. Geman, Stochastic Relaxation, Gibbs Distribution and the Bayesian Restoration of Images, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721–741, 1984.

[7] M. Hestenes and E. Stiefel, Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, Vol. 49, pp. 409–436, 1980.

[8] P. Jog, J.Y. Suh, and D. Van Gucht, The Effects of Population Size Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem, *Proc. Intl. Conf. Genetic Algorithms*, Fairfax, VA, pp. 110–115, June 1989.

[9] J.D. Kececioglu and E.W. Myers, Combinatorial Algorithms for DNA Sequence Assembly, *Algorithmica*, Vol. 13, pp. 7–51, 1995.

[10] J.D. Kececioglu, S.S. Shete and J. Arnold, Reconstructing Distances in Physical Maps of Chromosomes With Nonoverlapping Probes, *Proc. ACM Conf. RECOMB*, Tokyo, Japan, pp. 183–192, April, 2000.

[11] S. Lin and B. Kernighan, An Effective Heuristic Search Algorithm for the Traveling Salesman Problem, *Operations Research*, Vol. 21, pp. 498–516, 1973.

[12] O. Martin, S.W. Otto and E.W. Felten, Large-Step Markov Chains for the Traveling Salesman Problem, *Complex Systems*, Vo. 5, No. 3, pp. 299–326, 1991.

[13] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.

[14] P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, San Francisco, CA, 1996.