The 100-Processor Barrier in Parallel Simulated Annealing Algorithms

Zhihao Lou

Nov. 15, 2012
# Contents

1 Introduction .................................................. 1

2 Background ................................................... 3
   2.1 Theoretical Analyses of Simulated Annealing ................. 5
   2.2 Practical Simulated Annealing ............................. 8

3 Performance Measurement ..................................... 10
   3.1 Speedup and Efficiency ..................................... 10
   3.2 Quality Oriented Measurements ............................. 12
   3.3 Test problems ............................................. 13
      3.3.1 Explicit Cost Functions ............................. 14
      3.3.2 Application Oriented Problems ....................... 18

4 Parallel Strategies for Simulated Annealing .................... 19
   4.1 Single Markov Chain Approach ............................. 20
      4.1.1 Branch Prediction .................................... 20
      4.1.2 Simultaneous Moves ................................. 21
   4.2 Multiple Markov Chain Approach .......................... 22
      4.2.1 Search Space Decomposition ......................... 23
      4.2.2 Multiple Independent Runs .......................... 24
      4.2.3 Division Algorithm .................................. 25
1 Introduction

In the last decade, the serial performance of computing hardware has stopped increasing. Instead, the improvements are in the form of embedding more cores into a single processor chip and making more processors work together. Today, a single workstation can have as many as 12 cores and the top supercomputer has more than 1 million cores connected together\cite{43}. That means any computationally demanding work needs to be adapted to the parallel architecture. Fortunately, after active research, parallel solvers for linear systems, ordinary, and partial differential equations which apply to many categories of physical and biological problems are mature enough to achieve satisfactory efficiency. However, there is a category of problems that is equally important but difficult to perform under with massively parallel architecture: the global optimization problem.

The global optimization problem can be described as, without loss of generality,

$$\min_{x \in \mathcal{S}} f(x)$$ (1.1)

where $f$ is a cost function that has a range of $[0, \infty)$ and $\mathcal{S}$ is a search space. In practice, the search space $\mathcal{S}$ can be $\mathbb{R}^n$ or a subset of $\mathbb{R}^n$, integers $\mathbb{Z}^n$, a permutation group $S_n$ or a mix of all of the above. For maximization problems or cost functions with a different range, it is straightforward to transform the problem into the form in (1.1).

To solve the problem in (1.1) for linear or convex cases, there are many mature theories and
methods that will yield satisfactory results. Unfortunately, in practice many problems involve cost functions that are nonlinear and non-convex in a high dimensional search space. The structure of the optimization problems is in general not yet understood. These situations require a “black box” style optimization algorithm that can perform without prior knowledge of the search space. In practice, the simulated annealing algorithm is often used for this purpose and it works surprisingly well for obtaining near-optimal solutions.

The major drawback of using the simulated annealing algorithm is its long running time for acceptable results. Given the current conditions in of the optimization field, parallelizing the simulated annealing algorithm becomes an obvious direction. In this report, the author surveyed the different techniques used in parallelizing, their performance and its measurement, and the testing problems used in literature to date. However, the sad fact is, as this report will show, there seems to exist a barrier such that no algorithm available in the literature is able to surpass a speedup of 100. Consider that it is now commonplace for departments to have clusters with a few hundred nodes while larger universities and institutions to have access to even larger supercomputers, it is indeed a very disappointing situation. In the last part of this report, the author will discuss the possible directions in which to move forward.

When paralleledizing the optimization algorithms, there is a natural class of techniques that parallelize the evaluation of the cost function. This works well with a high dimensional search space on closely coupled systems with a small number of processors, such as SMP machines. For some problems, the requirement of communication at every cost function evaluation may prevent the algorithm from scaling up. More importantly, this technique relies on the problem structure and hence is not a parallelization of the simulated annealing algorithm itself. Based on these reasons, this report only focuses on problem independent methods without prior knowledge of the structure of the cost function. In addition, this report uses “processor” and “processing core” in a loosely interchangeable fashion, along with the more abstract term “processing element (PE)” which are used in some of the literature.
2 Background

The physical annealing process starts from a system in equilibrium at some high temperature $T$. According to statistical mechanics, the probability $P_E$ that such a system in a state with energy $E$ is determined by the Boltzmann distribution of the ensemble at temperature $T$

$$P_E = g(E) \frac{\exp\left(-\frac{E}{k_B T}\right)}{Z(T)}.$$ (2.1)

where $g(E)$ is the number of states having energy $E$ and $Z(T)$ is the partition function. Then, by cooling the temperature infinitesimally slowly, the system undergoes a quasi-equilibrium process and keeps the Boltzmann distribution at each corresponding temperature. When $T$ reaches 0, the Boltzmann distribution collapses to the global minimum or minima.

The simulated annealing algorithm introduced by S. Kirkpatrick is a stochastic optimization algorithm that mimics the physical process of annealing[23]. By introducing an artificial temperature $T$ into the system, this algorithm makes an analogy between the energy of a system and the value of the cost function being optimized. At each temperature, the algorithm runs the Metropolis algorithm repeatedly to perturb the state so that the distribution of the energies of the states will gradually approximate the Boltzmann distribution at that temperature $T$[31]. Thus, after gradually decreasing the temperature, the cost being optimized, or the energy, is expected to converge to its global minima, just as the statistical physics dictates it will in the real annealing process.
begin initialize $T_0$, $x_0$;
$E_0 \leftarrow f(x_0)$;
while not frozen do
\( x_p \leftarrow \text{perturb}(x_n); \)
\( E_p \leftarrow f(x_p); \)
\( \Delta E \leftarrow E_p - E_n; \)
\( \xi \leftarrow \text{uniform}(0, 1); \)
if $\xi < \text{accept}(\Delta E, T_n)$ then
\( x_{n+1} \leftarrow x_p; \)
\( E_{n+1} \leftarrow E_p; \)
else
\( x_{n+1} \leftarrow x_n; \)
\( E_{n+1} \leftarrow E_n; \)
end if
\( T_{n+1} \leftarrow \text{cooling schedule}(T_n); \)
n \leftarrow n + 1;
end while
end

Figure 2.1: Simulated Annealing Algorithm

The algorithm is shown in pseudocode in Figure 2.1. The perturbation in line 5 generates a random proposed trial solution $x_p$ based on the current solution $x_n$. This process is called move generation. For a discrete state space, it is common to pick $x_p$ from in the immediate neighborhood $x_n$. For continuous search space, a predefined distribution, usually Gaussian, centered at $x_n$ is used[16]. In some situation, distributions with unbounded variance like Cauchy distribution may also be used[41]. The proposed state $x_p$ is accepted according to the Metropolis criterion determined by the energy difference $\Delta E = f(x_p) - f(x_n)$ and the current temperature $T[31]$. That is, to accept $x_p$ with probability

$$ \text{accept}(\Delta E, T) = \begin{cases} \exp\left(-\frac{\Delta E}{T}\right), & \Delta E > 0 \\ 1, & \text{otherwise}. \end{cases} \quad (2.2) $$

Lines 5 through 15 are generally known as the Metropolis algorithm. A cooling schedule controls
the pace of the temperature decrease. The system is considered frozen either when it reaches a predefined temperature, usually 0 or a value very close to 0, or when no more moves can be accepted at the temperature.

2.1 Theoretical Analyses of Simulated Annealing

Despite the analogy between the simulated annealing algorithm and the actual physical annealing process, most of the theoretical analyses regarding simulated annealing are based on Markov chains rather than statistical mechanics. These analyses lead to precise cooling schedules and elegant convergence proofs no other meta-heuristics can yet achieve. Unfortunately, these schedules and proofs do not provide adequate directions in practice.

Hastings in 1970 proposed a way to analyze and also generalize the Metropolis algorithm using the Markov chain approach[20]. Consider the sequence of states \( \{x_n\} \) and the sequence of energies defined on these states \( E_n = f(x_n) \). The Metropolis algorithm implies that the state \( x_{n+1} \) is only dependent on its previous state \( x_n \) and hence \( \{x_n\} \) is a Markov chain. Under a fixed temperature, it is natural to assume the move generation is also fixed. Thus, the process of the Metropolis algorithm can be described using a homogeneous Markov chain. Let

\[
P_T(a, b) = P_T[x_{n+1} = b|x_n = a], \quad a, b \in \mathcal{S}
\]

(2.3)

denote the one step transition probability from state \( a \) to state \( b \) for such a Markov chain \( \{x_n\} \) at temperature \( T \). When the number of states in \( \mathcal{S} \) is finite, \( P_T(a, b) \) form a transition matrix \( P_T \). The transition matrix \( P_T \) can have the Boltzmann distribution as its stationary distribution if the detailed balance equation

\[
\frac{P(a, b)}{P(b, a)} = \frac{\exp[-E(b)/T]}{\exp[-E(a)/T]}.
\]

(2.4)
is satisfied. If an appropriate move generation model is chosen such that \( P_T \) is irreducible, i.e., \( \forall a, b \in \mathcal{S}, \) we can find a sequence \( x_0 = a, x_1, \ldots, x_{k-1}, x_k = b \in \mathcal{S} \) such that
\[
\prod_{i=1}^{k} P_T(x_{i-1}, x_i) > 0,
\] (2.5)
then such a Markov chain is also positive recurrent and its stationary distribution is unique. Furthermore, such a Markov chain will converge to the stationary distribution, in this case the Boltzmann distribution, regardless of the initial distribution.

In addition, each step in the Metropolis algorithm consists of first generating a proposed state and then deciding if that state is to be accepted. Since the acceptance probability is only dependent on the energy of the current and proposed states, the one step transition probability can be represented as a product of \( G(a, b) \), the probability of proposing \( b \) given \( a \), and \( A(a, b) \), the probability of accepting such a proposed move:
\[
P(a, b) = G(a, b)A(a, b).
\] (2.6)
In practice, the move generation \( G(a, b) \) usually only depends on the distance of the proposed state from the current state. That is \( G(a, b) = G(b, a) = G(r) \) where \( r \) is some measure of distance between state \( a \) and \( b \). This reduces (2.4) to
\[
\frac{A(a, b)}{A(b, a)} = \exp\left\{-\frac{E(b)}{T}\right\} \frac{\exp\left\{-\frac{E(a)}{T}\right\}}{\exp\left\{-\frac{E(a)}{T}\right\}}.
\] (2.7)
Assume \( E(b) \geq E(a) \), then the Metropolis criterion in (2.2) for the probability of accepting state \( b \) given state \( a \) at temperature \( T \) is \( A(a, b) = \exp\{-|E(b) - E(a)|/T\} \) while accepting state \( b \) has probability \( A(b, a) = 1 \). It is thus clear the Metropolis criterion is a special case of (2.7).

When \( T = 0 \), moves toward a higher energy state will no longer be permitted, and hence the irreducibility condition no longer holds. The theoretical analyses on the behavior of this system as \( T \) approaches 0 can be categorized into homogeneous and inhomogeneous Markov chain
approaches. The homogeneous approach constructs a Markov chain with $m$ sub-chains. Each sub-chain runs $n$ steps of the Metropolis algorithm at temperature $T_m$, where $T_m \to 0$ monotonically as $m \to \infty$. It can be proved that the Markov chain converges to a global minimum when $n \to \infty$ and $T_m \to 0(m \to \infty)$. To implement this algorithm based on the homogeneous approach, a given temperature has to be held long enough for the energy distribution to become stationary. Doing so is generally infeasible in practice, but the inhomogeneous approach provides a way to cool the temperature every step which still guarantees convergence to the minimum state. Geman and Geman pointed out, and later Gidas showed with much more rigorous analysis, that for a finite number of states, with a symmetrical and irreducible move generation, a cooling schedule that is slow enough to satisfy

$$T(n) \geq \frac{c}{\log(1+n)},$$

(2.8)

where $c$ is a problem specific constant, will make the algorithm converge in distribution\cite{16,17}. Hajek later proved that (2.8) is sufficient and necessary for the convergence\cite{19}.

From the Markov chain point of view, the convergence is a property of the transition matrices, hence it depend not only on the cooling schedule but on move generation as well. Szu and Hartley exploited this property and assert that, on a continuous state space, any bounded variance distribution as the move generation model can cool no faster than (2.8). In contrast, they demonstrated that using the Cauchy distribution, which has infinite variance, for the move generation model allows a cooling schedule that is inversely linear given by

$$T(n) \geq \frac{c}{1+n},$$

(2.9)

which converges to the global minimum\cite{41}. 

2.2 Practical Simulated Annealing

Although the cooling schedules derived in the previous section provide theoretical assurance, in practice these methods are too slow and the geometric schedule to be discussed below is usually employed. To see how slow they are, consider a specific example used in *Drosophila* embryo pattern formation models[38]. A typical simulated annealing runs in that paper starts at temperature $T_1 = 1000$ and ends at $T_n = 0.0202905$ in $n = 17737100$ steps. To cover the same temperature range using (2.8) requires about $10^{21403}$ steps. Consider the age of universe is only in the order of $10^{17}$ seconds, it is impossible for the logarithmic schedule to be deployed in practice. To address this problem, various schedules without a proven convergence are proposed and used in application with promising results.

The single most simple and widely used schedule is the geometric or exponential cooling schedule

$$T_{n+1} = \alpha T_n$$

with $0 < \alpha < 1$. Called simulated quenching by Ingber[22], this schedule gives up the proved convergence property in exchange for rapid execution and the results are largely acceptable. It is so commonly used that in the rest of the paper, all the cooling schedules used in the algorithms studied are geometric unless otherwise specified.

For more difficult problems, exponential cooling schedule sometimes will suffer from premature freeze and produces less then ideal results. This is a result of overly rapid cooling. Schedules that can overcome this problem generally take advantage of past statistics to dynamically determine the cooling pace.

Aarts and van Laarhoven introduced one such adaptive cooling schedule [3] in the form of

$$s_{n+1} = s_n + \frac{\lambda}{\sigma(s)}$$

(2.11)
where $s = 1/T$ and $\sigma(s)$ denotes the standard deviation of the distribution of values of the cost function of the points in the Markov chain at $s$.

Lam gave another adaptive efficient cooling schedule based on the quasi-equilibrium condition

$$
|\bar{X}(s_n) - \mu(s_n)| \leq \lambda \sigma(s_n),
$$

(2.12)

where $\bar{X}(s_n)$ is the average energy at inverse temperature $s_n$ and $\mu(s_n)$ is the mean energy at $s_n$ [24, 25]. In his work, he derives the optimal cooling schedule as

$$
s_{n+1} = s_n + \left[ \frac{\lambda}{\sigma(s_n)} \right] \left[ \frac{1}{s_n^2 \sigma^2(s_n)} \right] \left[ \frac{4\rho_0(1 - \rho_0)^2}{(2 - \rho_0)^2} \right],
$$

(2.13)

where $s_n$ is the inverse of temperature at step $n$, $\sigma(s_n)$ is the variance of the energy at inverse temperature $s_n$ and $\rho_0$ is the acceptance ratio.

The appearance of $\rho_0$ in (2.13) couples the Lam schedule to move generation. This is natural, because we have already seen that the properties of the transition matrix are central to the Markov chain model of simulated annealing, and moreover that the convergence properties of a particular schedule are intimately related to the move generation scheme used. What is less natural, and indeed quite surprising, is that the Lam schedule is unique among serial annealing algorithms in taking move generation into account. The schedule and the move control together give Lam’s algorithm very high performance, especially in terms of quality of the results. To see this clearly, we need to define the performance more carefully before we come back to discuss the parallel algorithms.
3 Performance Measurement

Any discussion of parallel algorithms would be misguided without a well defined parallel performance measurement. Measuring the performance of parallel simulated annealing algorithm is a rather controversial topic. There exist well established measurements like speedup and parallel efficiency. In deterministic algorithms these work very well. In stochastic optimizations like simulated annealing however, empirical data shows the quality of the result also has a direct impact on the performance. In addition, the difficulty of the test problem used may affect the performance because of scalability considerations. In this chapter, different classes of performance measurements are reviewed together with the test problems used.

3.1 Speedup and Efficiency

The most fundamental question in parallel computing is always how fast is the algorithm. Speedup is one such way of measuring the parallel performance. It is defined as

$$S_P = \frac{T_1}{T_P}$$

(3.1)

where $T_1$ and $T_P$ are the execution times of the serial and parallel versions of the algorithm on $P$ processors, respectively [15]. Sometimes, execution on a single processor may not be available
for certain architectures, or the timing is not representative due to memory or algorithmic constraints. In these situations, a modified speedup

$$S_P = \frac{P_0 T_{P_0}}{T_P}$$  \hspace{1cm} (3.2)

based on an appropriate minimal number of processors $P_0$ may be used instead.

In addition to the speedup, how efficient it is to achieve speedup is also a central concern. The parallel efficiency, defined as

$$E_P = \frac{S_P}{P},$$  \hspace{1cm} (3.3)

can be used when discussing the scalability of the algorithm over a wide range of processors used. Generally, an algorithm with linear speedup

$$S_P \geq cP,$$  \hspace{1cm} or  \hspace{1cm} $$E_P \geq c$$  \hspace{1cm} (3.4)

can be called strong scalable if equation (3.4) holds for some $c > 0$ over a wide range of $P$ on a fixed size of problem. If (3.4) is achievable over problems whose size is increasing with $P$, then it is considered weak scalable. Linear speedup with $E_P = 1$ is the ideal case. Cases with $E_P > 1$ are called superlinear speedup. They rarely happen and should be put under strict scrutiny. As Eager et al. pointed out, speedup and efficiency are inherently in conflict [14]. The general aim of a parallel algorithm is usually to reach a speedup as high as possible while still having reasonable efficiency.

Since speedup and efficiency are two widely used measurements, many authors simply use them without second thoughts [7, 9, 21, 26, 36, 37, 40]. However, some works exhibit suspicious near perfect or even superlinear speedup [9, 21, 37]. Among these, Ram et al. are able to show their algorithm has comparable quality of results for 3 nodes with almost perfect speedup[37]. But the absence of quality data from 3 nodes up to 9 nodes in that work inspires little confidence. As Greening pointed out, people often observe superlinear speedup in parallel simulated algorithms, which might be caused by either a change of algorithmic behavior that the serial
algorithm can also benefit from, or alternatively a parallel algorithm that stops prematurely [18]. The inaccuracy caused by changes of algorithmic behavior can be easily compensated by mimicking the parallel behavior in serial. However, to compensate for premature stopping of the optimization one needs quality adjusted measurements.

3.2 Quality Oriented Measurements

Conceptually, the ideal speedup should be measured using the best serial timing against parallel ones with comparable quality of results. In practice, neither one is easy to obtain. Researchers who are willing to measure the quality of their algorithms usually use a form of approximation instead.

One of the common methods used is to measure the quality of the results over a fixed number of iterations [42, 48]. After the same number of total iterations in serial and parallel algorithms, for cases with comparable or better quality, this automatically leads to linear speedup in terms of iterations. In work which is more concerned about the results of the optimizations, sometimes the quality of the results alone might be used [39].

As early as 1987, Darema et al. proposed [12], among other things, a measurement that used the frequency of successful convergence in addition to the running time in parallel compared with the serial algorithm. Unfortunately, the algorithm reported was not run on an actual parallel machine, but was simulated on virtual machines, which hence reduces its importance. Another careful measurement was proposed by Chu et al. in 1999 [11]. That paper empirically measured the trade off between number of iterations and the quality of the results of the problem of interest in serial under different parameters. The authors find a log-log relation of the two, and use that to compensate the parallel performance. The major drawback of this method is the tradeoff relation has to be evaluated empirically for each problem tested, which means it is
not feasible to evaluate for some real problem that is too big to run in serial. In addition, the paper only reports performance in terms of number of iterations, and no speedup in time was reported.

Other quality adjusted speedup measurements includes a measure of the speedup which is based on the claim of having the same average quality of solutions [8], though no tuning of serial algorithms was mentioned. In addition, Chen et al. [10] reported performance in the form of curves in the processors-iterations plot that each refer to a certain range close to the optimal, which can also be considered as some variant of quality adjusted speedup. No serial performance was reported in that paper because of hardware restrictions.

### 3.3 Test problems

The choice of test problems used in research on parallel simulated annealing are often overlooked. As Wolpert and Macready pointed out in their famous “No Free Lunch” paper [46], the average performance of any pair of algorithms across all possible problems is identical. For parallel algorithms, Wolpert’s work indicates that two algorithms which performs with widely different efficiencies on one problem may actually have similar overall efficiency. Though it is impossible to test the algorithms on all problems, caution should still be taken to make sure the test problems are as far apart as possible from each other to get a proper evaluation of the algorithm. The ubiquity of optimization problems creates an undesired side effect of having too many problems for any single study to cover. In this section, we go through the test problems commonly used in the literature.

Another issue related with the choice of test problems is the size of the problem that may have an impact on the performance of the algorithm. As Amdahl’s Law puts it [5], the theoretical
limit of the speedup of any parallel algorithm is

$$S_P \leq \frac{P}{1 + f(P - 1)}$$  \hspace{1cm} (3.5)

where $P$ is the number of processors and $f$ is the fraction of the time the algorithm needs to spend on the serial portion. It is easy to see that the upper bound of the speedup for any problem is $1/f$, but when put into a different perspective, the $f$ or the upper bond is not only algorithm dependent, but also problem dependent. Empirically, $f$ goes smaller when the problem grows, in terms of dimension or difficulty. This indicates that when testing the performance of any algorithm, caution must be used to separate the speedup increase due to the advancement in the algorithm itself from the simple fact of choosing a intrinsically more scalable problem.

3.3.1 Explicit Cost Functions

Explicit cost functions are the most commonly used test problems. They are fast to evaluate and, when constructed carefully, can have many local minima but only one or a couple of global minima. Some even have variable number of dimensions to help calibrate the scalability. Though it is arguable that they do not represent any real world applications, their properties make them very popular as test problems.

Traditional Functions

Dekkers and Aarts has collected a few functions that are suitable for testing simulated annealing algorithms [13]. Those functions include

First, there is Goldstein and Price’s function, given by

$$f(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$

$$[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)],$$  \hspace{1cm} (3.6)
where \( x_i \in [-2, 2] \). There are four local minima and \( f(0, -1) = 3 \) is the global minimum.

Another example is **Branin’s function**

\[
f(x_1, x_2) = \left( x_2 - 5.1x_1^2/4\pi^2 + 5x_1/\pi - 6 \right)^2 + 10(1 - 1/(8\pi)) \cos x_1 + 10 \tag{3.7}
\]

where \( x_1 \in [-5, 10] \) and \( x_2 \in [0, 15] \). There are 3 global minima at \( x_{\text{min}} = (-\pi, 12.275) \), \((\pi, 2.275)\) and \((3\pi, 2.475)\) where \( f(x_{\text{min}}) = 5/4\pi \). There are no more minima in the domain.

The **Hartmann family of functions** are in the form of

\[
f(x) = -\sum_{i=1}^{m} c_i \exp \left( -\sum_{j=1}^{n} a_{ij} (x_i - p_{ij})^2 \right) \tag{3.8}
\]

where \( x_i \in [0, 1] \). Hartmann3 is the 3-D version with \( n = 3 \) and \( m = 4 \) and Hartmann6 is the 6-D version with \( n = 6 \) and \( m = 4 \). Both have 4 local minima and one of them is the global minimum. Specific value of \( a_{ij}, c_i \) and \( p_{ij} \) which are commonly used can be found in [13].

**Shekel’s family of functions** are in the form of

\[
f(x) = -\sum_{i=1}^{m} \left[ (x - a_i)^T(x - a_i) + c_i \right]^{-1} \tag{3.9}
\]

where \( x = (x_1, \ldots, x_n) \), \( x_i \in [0, 1] \), \( n = 4 \) and \( m = 5, 7, 10 \) for functions with 5, 7 and 10 local minima, respectively. Common values for \( a_i \) and \( c_i \) can be found in [13].

The **Penalized Schubert’s family of functions** all have a penalty terms in the form of

\[
u(x_i, a, k, m) = \begin{cases} 
  k(x_i - a)^m, & x_i > a, \\
  0, & -a \leq x_i \leq a, \\
  k(-x_i - a)^m, & x_i < -a.
\end{cases} \tag{3.10}
\]
The 2-D Schubert’s function is
\[
f(x_1, x_2) = \left\{ \sum_{i=1}^{5} i \cos[(i + 1)x_1 + 1] \right\} \left\{ \sum_{i=1}^{5} i \cos[(i + 1)x_2 + 1] \right\} 
+ u(x_1, 10, 100, 2) + u(x_2, 10, 100, 2)
\] (3.11)

where \( x_i \in [-10, 10] \). It has 760 local minima, 18 of which are global.

The 3-D Schubert function is
\[
f(x) = \frac{\pi}{n} \left\{ k_1 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - k_2)^2[1 + k_1 \sin^2(\pi y_{i+1})] + (y_n - k_2)^2 \right\} 
+ \sum_{i=1}^{n} u(x_i, 10, 100, 4)
\] (3.12)

where \( n = 3, y_1 = 1 + \frac{1}{3}(x_i + 1), k_1 = 10, k_2 = 1 \) and \( x_i \in [-10, 10] \). This function has roughly \( 5^3 \) local minima and one global minimum \( f(1, 1, 1) = 0 \).

The 5-D Schubert function is
\[
f(x) = k_3 \left\{ \sin^2(\pi k_4 x_1) + \sum_{i=1}^{n-1} (x_i - k_5)[1 + k_6 \sin^2(\pi k_4 x_{i+1})] \right\} 
+ (x_n - k_5)^2[1 + k_6 \sin^2(\pi k_7 x_n)] \right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)
\] (3.13)

where \( k_3 = 0.1, k_4 = 3, k_5 = 1, k_6 = 1, k_7 = 2 \) and \( x_i \in [-5, 5] \). This function has roughly \( 15^5 \) local minima and one global minimum \( f(1, 1, 1, 1, 1) = 0 \).

These commonly used functions vary widely in terms of difficulty. Thus, these can be roughly considered as far apart enough to sufficiently test the performance of the algorithm. Indeed, many studies in the field use all or some of above mentioned function as a test function\([9, 36, 39, 42, 44, 48, 49]\).
Variable Dimensional Functions

A major disadvantage of using above mentioned functions is they are of limited dimensions. When being tested on large scale supercomputers, the search space may be too small for the parallel algorithms to run efficiently. Thus, test functions with variable number of dimensions have been proposed and are now widely used. In the literature, three such functions collected by Mühlenbein are of particular interest. They are all smooth functions with one global minimum but in which the number of local minima grows exponentially with the number of dimensions $n$. These are, the **Rastrigin function**, given by

$$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10 \cos(2\pi x_i)], \quad (3.14)$$

where $x_i \in [-5.12, 5.12]$. It has a global minimum at $f(0, \ldots, 0) = 0$. The **Schewefel function**, given by

$$f(x) = \sum_{i=1}^{n} -x_i \sin \sqrt{|x_i|}, \quad (3.15)$$

where $x_i \in [-500, 500]$. It has a global minimum at $x_i = 420.9687, i = 1, \ldots, n$. Finally, there is the **Griewank function**, given by

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1, \quad (3.16)$$

where $x_i \in [-600, 600]$. It has a global minimum at $f(0, \ldots, 0) = 0$.

Multiple papers have used all three with the dimension $n$ ranging from 10 to 50, and Chen et al. in 2007 used a modified version of (3.14) with $n = 2$ and (3.16) with $n = 2, 4, 6$ and 8.
3.3.2 Application Oriented Problems

Besides those predefined functions, many authors test their algorithms on more application-oriented problems. Classic combinatorial optimization problems are often used either for testing purposes or as an actual problem of concern. Sometimes less studied problems or even applied problems are used for testing.

The traveling salesman problem (TSP) is such a widely used test problem for all kinds of algorithms. The problem is formulated as to minimize the distance traveled on a route which traverses all cities in a list exactly once, returning to the starting point. Though in theory, the TSP can have cities on any metric space, in practice, it is usually makes use of either Manhattan distance or Euclidean distance. The advantage of using the TSP is that it is known to be NP-hard while at the same time has many sets of cities with proved optimal solutions that the solution quality can be checked against \[1\]. Furthermore, the number of cities in the problem can also be adjusted, making the test more flexible. These advantages make the TSP an attractive test problem which is used by many researchers, with the number of cities ranging from 50 to 500 \[7, 10, 37, 40\].

Other application oriented problems seen in literature include chip placement \[12\], graph partitioning problem \[26, 32\], job-shop scheduling \[37\] and the design of error correcting code \[10\]. Real world problems like the Drosophila pattern formation model \[11\], biomechanical pedaling simulation \[21\] and band selection \[8\] are also used as test problems.

After discussing both the theoretical basis of the simulated annealing algorithm and the measurement of its performance, we are ready to consider the different methods which have been used to parallelize it together with their performance.
4 Parallel Strategies for Simulated Annealing

As simulated annealing is such a powerful optimization algorithm, it is unsurprising to see a large number of studies devoted to its parallelization. Despite its name, the parallelized algorithm does not necessarily follow the exact behavior of the original algorithm. This is a typical situation for parallelization in general. For example, in cases like iterative methods for a linear system or PDE solver, outdated boundary data are routinely used for faster execution. On the flip side, keeping the algorithmic framework intact in simulated annealing means all the theoretical analyses, especially regarding convergence can be applied without modification given a proper move generation and cooling schedule. Since the Markov chain analysis in section 2.1 indicates an inherently sequential process at the heart of the simulated annealing algorithm, all parallelization schemes must face a fundamental question of how closely should the parallelized algorithm follow the serial framework. Too close to the serial approach will certainly limit the speedup, while straying too far will lose the good attributes and result in a degradation of quality.

This chapter classifies the parallelization schemes into three large categories according to their closeness to the serial algorithm: single Markov chain, multiple Markov chain and population-based approaches. While both Markov chain approaches are more or less direct parallelizations of the simulated annealing algorithm, population-based approaches are hybrid schemes which are more distant from serial behavior.
4.1 Single Markov Chain Approach

Many early efforts in parallel simulated annealing algorithm to take the form of a single Markov chain, attempting to keep the same theoretical properties as the serial algorithm. Although paralleling a serial process without too much alteration is generally a hard problem, there are many techniques like branch prediction and domain decomposition that can be borrowed from other algorithms.

4.1.1 Branch Prediction

The simulated annealing algorithm has a perfect structure for branch prediction. At every step, the algorithm needs to decide whether to accept a proposed move. For most problems it is much faster to propose a move than to actually evaluate the cost difference of that move and make a decision. When step $n$ has been proposed, it is feasible to propose moves at step $n + 1$ on other processors based on both the acceptance and rejection cases of step $n$. The same can be said for both proposed moves at step $n + 1$ and so forth, up to $s$ steps in advance at step $n + s$, as shown in Figure 4.1. Thus, when all the steps are decided, the whole process is still a single Markov chain in retrospect and retains all the key properties of the serial simulated annealing algorithm. This balanced approach needs $2^{s+1} - 1$ processors for every $s$ steps ahead. After step $n$ has been evaluated and decided, one whole branch of the tree needs to be abolished, along with step $n$ itself, which comes to exactly $2^s$ nodes that can be used as step $n + s + 1$. Obviously, this method requires a number of processors which grows exponentially with $s$. In addition, even if the number of processors is infinite, the speedup can be no more than the depth of the tree the algorithm can construct during the time period of a cost function evaluation, i.e., the speedup is limited by the ratio of the times required for cost function evaluation to move generation.

To improve performance, Witte et al. proposed a speculative algorithm that takes advantage
of the fact that, under fixed move generation, the annealing process tends to accept most of
the moves when the temperature is high, and rejects most of the moves when temperature
is low. Thus, instead of constructing a balanced tree every time, this method constructs an
unbalanced tree. The shape of the tree is dependent on the estimated acceptance probability.
When the temperature is high, the tree has more nodes on the accepting branch, and when
the temperature is low, more on the rejecting branch. Thus, more depth can be achieved when
either the temperature is very high or very low. In their paper, Witte et al. report that they
are able to achieve a speedup of 4 with 16 processors on a 100-task 4-processor assignment
problem. Wong and Constantinides [47] further improve the performance by introducing a
threshold function using previous data to help predict the acceptance probability. That method
is able to get speedup of more than 15 using 100 processors on a 1024-city TSP.

4.1.2 Simultaneous Moves

Simultaneous moves is a method based on the observation that, without move control, most
of the proposed moves will be rejected when temperature is low. As the name indicates, this
method allows all the processors to propose and evaluate their own moves based on the common
state. Each step will have only one state being accepted into the main Markov chain. By doing
This method is used in the “Approach 2” in the work of Chen et al. in 2007 [9], in which every processor generates local moves according to a randomly assigned step size. At the end of the step, the best move is selected as the new global state. However, this method fails to show appealing results. Sohn proposed [40] a generalized speculative computation method based on $n$-array speculative computation that combines the concepts of both decision tree and simultaneous moves. Instead of constructing a tree directly, it proposes multiple moves simultaneously. All these moves are ordered along a single Markov chain with appropriate temperature. After the first accepted move, all the rest of the moves are invalid to be on the same chain, and hence are dropped. That accepted state is then broadcast to all other processors for the next iteration. In that paper, Sohn was able to report a speedup around 20 using 100 processors on TSPs ranging from 100 to 500 cities.

### 4.2 Multiple Markov Chain Approach

Although the idea of using a single Markov chain has many advantages, such a framework has a fundamentally serial quality which does not mesh well with parallel computing. Among the methods reviewed above, the best implementation is only able to report a speedup of 20 using 100 processors or 20% efficiency. One major caveat of this approach is it requires communication after every cost function evaluation so to keep the algorithm following the single Markov chain. These communications can be reduced or eliminated if the requirement of following a single Markov chain is loosened. Thus, each processor can follow its own chain for a certain length before any communication has to happen. This approach is called the multiple Markov chain approach.
The general idea of multiple Markov chain approach is simply slicing the Markov chain of length $L$ into $P$ sub-chains of length $L/P$ on which $P$ processors runs independently. When being implemented, many choices have to be made and hence lead to a whole family of algorithms. The iterations within $L$ are commonly called a segment, within which there is no communication between processors. The segment length $L$ can be preset or determined dynamically. When $L$ is equal to the total number of iterations, the algorithm is called “multiple independent runs” (MIR). At the end of its own $L/P$ iterations, each processor communicates synchronously or asynchronously with other processors, to decide which state or states to be chosen as the starting point of the next segment. Other information may also be exchanged at that time.

4.2.1 Search Space Decomposition

Decomposition is a common technique among parallel algorithms. For simulated annealing algorithms in particular, search space decomposition is a good candidate as it does not require excessive decomposition in evaluating the cost function. However, it does require the search space to satisfy

$$f(x + \Delta x) - f(x) \approx \sum_{i} [f(x + \Delta x_i) - f(x)],$$

(4.1)

where $\Delta x = \sum_{i=1}^{n} \Delta x_i$, $\Delta x_i \in S_i$ and $S_i$, $i = 1, \ldots, n$ is a partition of the search space $S$. That is, a move is approximately the sum of the moves in each decomposed sub-space. Thus, the parallel algorithm can maintain a global configuration while allow each processor to generate and evaluate its own local moves. The inaccuracy introduced by the approximation is considered noise and may or may not be corrected periodically depend on the variant used. In practice, many but not all problems exhibit the property in (4.1) and make the method feasible in some cases.

One early such effort by Darema et al. parallelizes the simulated annealing for chip placement problem [12]. This problem has a natural way to decompose the search space according to the
different chips considered. Moving of a single chip can be generally considered independent from
others. Thus, each processor picks and makes its own moves while it can still maintain a global
configuration. Among different conditions including whether the approximation be corrected
periodically and how strict the locking is, the authors report to achieve a theoretical speedup
of 14.1 on 32 processors with correction and less strict locking. Unfortunately, no realtime data
has been reported as the study was done on a virtual machine.

However, for problems which \((4.1)\) is not exact, this method may break down beyond certain
number of processors. Macready et al. demonstrated \([28]\) that on both generalized spin-glass
model problem and TSP, the quality of the results exhibit an abrupt degradation after certain
decrease in accuracy of approximation in \((4.1)\). Such accuracy of approximation is determined
by the degree of parallelism \(\tau\) used in the paper which can roughly be seen as \(\tau = P/N\), the ratio
of parallel processors used \((P)\) to the size of the problem \((N)\). The transition between good
and bad quality of results is sudden and the quality differences are very large. This suggests a
phase transition-like behavior with a critical value of \(\tau\). The exact value of critical \(\tau\) is problem
dependent. The existence of this behavior prevents the inexact search space decomposition
method to have wider application and larger speedup.

4.2.2 Multiple Independent Runs

Among all possible variances of the algorithm, MIR is arguably the simplest parallelization
method. After the initial states being generated, there is no communication until the very
end. Naturally, only the best result needs to be reported. In theory, MIR should have better
performance than the division algorithm to be introduced later, as Azencott pointed out \([6]\)
that “\(P\)-simultaneous periodically interacting annealing scheme is asymptotically less efficient
than the \(P\) simultaneous independent annealing scheme.” However, the finite-step experiments
does not necessary align with the asymptotic analysis.
Lee and Lee have reported an implementation of MIR which they call the noninteracting scheme that has better speedup than single Markov chain approaches but worse than other more sophisticated MMC approaches [26]. The best performance reported using this method is 13.4 using 16 processors on a 1200-node graph partition problem, with some decrease in the quality of the result.

A similar implementation called the asynchronous approach (AS) was studied among other methods in Onbaşıoğlu and Özdamar’s work [36]. The best performance of this method reported in this paper shows an average speedup of 6.86 using 8 processors on a set of functions.

4.2.3 Division Algorithm

The division algorithm was originally proposed by Aarts et al. [2]. It chooses the best of the states at the end of each segment and every processor starts the next segment with that best state. The length of the segments are predefined. The book shows that this method can scale perfectly up to 8 processor on the 100-city TSP without substantial decrease of quality.

The periodic exchange scheme in Lee and Lee’s paper is essentially the same method [26]. The paper reported a speedup of 14.8 using 16 processor on a 118-node graph partition problem. Onbaşıoğlu and Özdamar tested an almost identical method called the synchronous approach with occasional enforcement of best solution-fixed intervals (SOEB-F) on different classes of test functions and reported a speedup of 6.39 using 8 processors in the best scenario [36].

In addition, Higginson et al. proposed an algorithm called simulated parallel annealing within a neighborhood (SPAN) that is very similar to the algorithms described above, but each processor is allowed to perform moves with a different search radius [21]. The paper reported near linear speedup for fitting of 50-dimensional parabola and simulation of pedaling in up to 32
processors “with an error comparable to the solution found using a serial simulated annealing algorithm”.

4.2.4 Resample of States

A potential drawback of the plain division algorithm is that it is likely to be trapped into some local minima. By selecting only the best state at the end of every segment, the algorithm actually samples less search space than it otherwise could. To compensate this, variants of the division algorithm have been proposed that instead of choosing the best state, re-sample the states randomly from the pool of states in the end of each segment.

This method was first mentioned in Lee and Lee, but these authors presented no details [26]. A sophisticated algorithm proposed by Chu et al. [11] is an example of this type. The algorithm resamples the state at the end of a segment with temperature $T$ such that the probability for a processor to accept the state from processor $i$ is

$$e^{-E_i T \sum_{j=1}^{P} e^{-E_j T}}.$$  \hspace{1cm} (4.2)

In addition, it is based on the Lam’s schedule discussed in section 2.2. The statistics used for that schedule are also exchanged among processors at the end of each segment [11]. The algorithm is able to achieve a linear theoretical speedup up to 50 processors and 80 for 100 processors on a embryonic pattern formation model. However, these number are in number of iterations only, and do not count the overhead of communication.

A similar approach proposed by Chang et al. is called parallel simulated annealing band selection (PSABS) [8]. At the end of each segment, the algorithm finds and broadcasts the best state to all the processors. That best state is treated like another proposed state such that each processor decides to adopt that state with a probability given by the Metropolis criterion [2.2]. This
PSABS was tested, as its name suggests, on a band selection problem. It is shown to scale nearly perfectly up to 8 nodes with some decrease of quality in results.

4.2.5 Variable Interval Scheme

For implementations with fixed move generation schemes, the observation that the acceptance rate decreases with temperature discussed in section 4.1.2 can still apply to the division algorithm. When considering this observation, it becomes more desirable for the division algorithm to have longer segments when the temperature is high, and shorter ones when low.

Ram et al. [37] proposed, among other things, a cluster algorithm (CA) for simulated annealing that features segments which linearly decrease in length at a predefined pace. At the end of a segment, the best state is chosen as in the division algorithm. The paper reported this method to have a speedup of 9.01 using 9 nodes on a 10 × 15 job scheduling problem.

The dynamic exchange scheme in Lee and Lee’s paper is a modification of this scheme [26]. The interval of segments in their method is dynamically determined by the acceptance rate, in addition to the minimum length that is needed for removal of correlation. As in all other schemes in that paper, only the best state among all processors is passed to the next segment. The authors reported a speedup of 15.8 using 16 processors on a 118-node graph partition problem.

Onbaşoğlu and Özdamar [36] have also presented a synchronous approach with occasional enforcement of best solution-varying intervals (SOEB-V) that relies on the temperature rather than acceptance rate to determine the segment length. The authors reported a poor speedup of 3.86 on 8 processors, which is the best result among optimization of different sets of functions.
Li et al. also used the dynamic exchange intervals approach in solving their 3D layout design problem [27], though the detailed synchronization condition was not given. The paper did not report speedup directly but it can be measured to be roughly 3.28 using 4 processors.

4.2.6 Asynchronous Approach

All the methods described above require periodic synchronization at fixed or non-fixed intervals. For many cost functions, the time of evaluation varies from state to state. Thus, those processors which finish faster have to wait for the slower ones and that causes overhead which will reduce the performance of the parallel algorithm. Asynchronous communication is a common technique that may reduce that kind of overhead. This technique requires a global state that all processors can access asynchronously. Thus, instead of all processors waiting for each other for synchronization, each processor can access the global state at its own pace.

Lee and Lee’s paper [26] tested an implementation of this asynchronous approach and got a possibly suspicious speedup of 25.1 using 16 processors on a 118-node graph partition problem. The quality of the result exhibits a substantial decrease. Chang et al. presented a similar asynchronous implementation for solving band selection problems [8]. Unfortunately, the performance data in that paper did not separate synchronous and asynchronous implementations.

4.2.7 Sample Sort Simulated Annealing

All previously discussed multiple Markov chain methods keep the temperature the same across the processors. Sample-sort simulated annealing introduced by Thompson and Bilbro [42] maintains a static temperature gradient across the processor while allow the states to move. In this method, on \( P \) processors, the authors make the temperature \( T_k \) on each processor \( k \) be monotonically increasing with \( k \). Then periodically, in addition to the annealing steps, each processor
i communicate with its neighbors $j \in [i - N, i + N]$ and adopt the neighboring state according to the probability

$$A(x_i, x_j) = \min \left\{ 1, \exp \left[ -(E_j - E_i) \left( \frac{1}{T_i} - \frac{1}{T_j} \right) \right] \right\}.$$  \hspace{1cm} (4.3)

In experiments, unfortunately, the method shows a significant decrease of convergence rate compared to a serial algorithm with the same total number of iterations both in 10-processor and 100-processor cases. The situation improves somewhat when more iterations in each temperature step.

### 4.2.8 Coupled Simulated Annealing

Coupled simulated annealing\cite{48} revisits the detailed balance principle in equation (2.7) and derives new acceptance criteria in the form of

$$A_\Theta(x_i, y_i; \gamma)$$  \hspace{1cm} (4.4)

where $i$ is the index over the set of processors $\Theta$, and $\gamma$ is a coupling factor that is a function of energies in $\Theta$. The idea is to normalize the acceptance probability so as to on one hand hold more knowledge of low-energy regions of the cost function, and on the other hand to explore unknown regions better. The paper only reported the quality improvement for a fixed number of iterations over 10 and 100 processors. No speedup data was reported.

### 4.3 Population Based Parallel Simulated Annealing

Aside from being used as a self contained algorithm for optimization problems, simulated annealing and its parallel methods are often being used together with other optimization algorithms, in particular with the family of evolutionary algorithms, to produce hybrid algorithms. The
family of evolutionary algorithms, including evolution strategies, evolutionary programming and genetic algorithm, is sketched in Figure 4.2.

```
1 begin
2    initialize P_0;
3    E_0 ← f(P_0);
4 while not stop-condition do
5        P'_n ← variation(P_n);
6        E'_n ← f(P'_n);
7        P_{n+1}; E_{n+1} ← select(P_n, P'_n; E_n, E'_n);
8    n ← n + 1;
9 end while
10 end
```

Figure 4.2: Evolutionary Algorithm

The algorithm maintains a population of states \( P = [x_1, x_2, \ldots, x_p] \) where \( p \) is the population size, and its corresponding cost or fitness \( E \). Using ideas from population genetics, each state of the search space \( x_i \) is considered to be the genotype of an individual \( i \), where the phenotype is simply the fitness or cost \( E_i \). The evaluation of the population in the evolutionary algorithms is a naturally parallel process. At each step, a variation operation generates a temporary population \( P' \) based on the current population. Following classic population genetics, such variation can be either a mutation or a crossover. A mutation generates a new individual from a perturbation of a existing individual. A crossover, however, requires two existing individuals. The temporary population, together with the original population, then undergoes a selection process that determines the population for the next step based on their fitness. The algorithm loops through these steps until it reaches some predefined stop condition [4].

The main loop of the evolutionary algorithms have a similar structure to that of the simulated annealing algorithm. Rudolph pointed out [39] that when the population size \( p = 1 \), the variation operator reduces to a perturbation on the current state, and the EA algorithm can be
formulated in terms of Markov chain as

\[ x_p = x_n + z_n \] (4.5)

\[ x_{n+1} = x_p \cdot a(x_n, x_{p+1}) + x_n \cdot (1 - a(x_n, x_{p+1})) \] (4.6)

where \( z_n \) is the perturbation and \( a(x_n, y_{n+1}) \) is the acceptance function that determines the selection process. For convergence, a selection process that controlled by some monotonically decreasing parameter \( T \) similar to the Metropolis criterion (2.2) is needed, together with an appropriate cooling schedule. When satisfying these conditions, these becomes the same equations to describe the simulated annealing algorithm. From another point of view, the evolutionary algorithms are generalization of the simulated annealing algorithm. All these suggest that a population based parallel simulated annealing algorithm might show good performance. It is worth noting that, due to the structural similarities, most of the multiple Markov chain approaches can be described using this population based framework, with the absent of crossover, but only the explicitly hybrid schemes will be discussed in this section.

### 4.3.1 Two-stage Algorithm

Despite the emphasis in the convergence theory of the simulated annealing algorithm on the ergodicity of the Markov chain, a good initial solution may still benefit the annealing process in practice. A two-stage algorithm takes the advantage of this empirical observation. It runs an evolutionary algorithm for certain number of steps and then selects the \( P \) best states from the resulting population to be the initial states for a multiple Markov chain style simulated annealing, where \( P \) is the number of processors involved.

The genetic clustering algorithm (GCA) proposed by Ram et al. is one such algorithm [37]. It runs genetic algorithm with population size \( P \), the number of processors used, on one processor for a fixed number of iterations before distributes the states to other processors for independent
simulated annealing runs. The authors reported a speedup of 7.07 using 9 processors on a 150-city TSP. Chen et al. tested a similar approach identified as “approach 4” in that paper [9]. Instead of independent runs after initialization, they ran a division algorithm with fixed segment length. That paper reported a significant loss of quality compared to the best approach tested in the same paper and there was little detailed analysis.

4.3.2 Periodic Evolution

For parallel simulated annealing with multiple Markov chains, such as the division algorithm, the communication phase at the end of each segment is a good place for an evolutionary algorithm to act. The states from different chains can naturally be considered as a population. Selecting the best state policy or the resampling discussed in section 4.2.4 can both be seen as special cases of selection in the perspective of an evolutionary algorithm. To generalize this idea, selection that considers both the current and previous population or the introduction of crossover both potentially help in the exploration of the search space.

Yong et al. presented an algorithm called the annealing evolution algorithm [49] that is based on the division algorithm with fixed segment length. At the end of each segment, their algorithm pools the states at the beginning of the current segment as well as those at the end of the segment and selects from the pool with probabilities proportional to the cost of each state. The paper compared this algorithm with Mühlenbein’s parallel genetic algorithm [34], and showed a significant reduction of number of function evaluations under same population size on a set of test functions. Unfortunately, the tests in this paper were run on a single processor machine and no realtime data were available.

The synchronous approach with occasional solution exchanges (SOS) discussed by Onbaşoğlu and Özdamar in contrast allows crossover between segments. The states at the end of each segment are treated as parents. For each individual state $x_p$, the algorithm randomly chooses
an \( x_q \) for it to mate with such that \( p \neq q \). The offspring \( x_c \) differs from \( x_p \) by one dimension \( i \) which is randomly picked such that

\[
x_c^{(i)} = (1 - w)x_p^{(i)} + wx_q^{(i)}
\]  
(4.7)

where the weight \( w = E(x_p) / [E(x_p) + E(x_q)] \). The resulting new population is then directly used as the starting point for next segment, without further selection. The paper reported the best speedup to be 6.62 using 8 processors among different sets of test functions.

The “Approach 5” of Chen et al.’s work goes one step further and runs a full genetic algorithm between fixed length segments [9]. In the genetic algorithm part, for two randomly selected parent states \( x_p = (x_1, \ldots, x_d) \) and \( y_p = (y_1, \ldots, y_d) \) where \( d \) is the number of dimensions of the state, the algorithm generates two random integers \( \mu \in [1, d] \) and \( \nu \in [0, d/2] \) such that the child states are in the form

\[
x_c = (x_1, \ldots, x_{\mu-1}, y_\mu, \ldots, y_{\mu+\nu-1}, x_{\mu+\nu}, \ldots, x_d)
\]

and

\[
y_c = (y_1, \ldots, y_{\mu-1}, x_\mu, \ldots, x_{\mu+\nu-1}, y_{\mu+\nu}, \ldots, y_d),
\]

i.e., two parent state switch \( \nu \) variables starting from the \( \mu \)th one. The selection method used is called tournament competition strategy. The paper reported a better convergence rate than all other methods investigated by the authors, namely domain decomposition, multiple simultaneous moves, the division algorithm and the two-stage algorithm. The author reported a best relative speedup of 26.47 using 40 processors. The relative speedup used here is defined by \( S_P^{\text{relative}} = T_2 / T_P \), and hence can be roughly seen as to be \( S_P = 2S_P^{\text{relative}} \). Thus, the reported result is equivalent to a speedup of 52.94 on 40 processors in terms of (3.1). Because the algorithm requires the population to be the same as the number of processors, such speedup actually reflects the effect of both increasing the population size and parallel performance.
4.3.3 Evolutionary Annealing

The evolutionary annealing algorithms discussed here is a class of algorithms that use evolutionary style mutation and crossover for proposing new states, but still retain a Metropolis style selection and cooling schedule. In this manner, the algorithm can sample the search space much more thoroughly while satisfying all the requirements for convergence, given the proper cooling schedule.

In 1992, Mahfoud and Goldberg proposed a parallel recombinative simulated annealing (PRSA) algorithm that is the pioneering work in this category [29]. In each step, the algorithm randomly pairs each member of the population with another. For each pair, it generates two children using both crossover and mutation. The probability of accepting the child state with cost $E_c$ given the cost of parent state being $E_p$ is

$$e^{-E_c} \over e^{-E_c} + e^{-E_p}, \tag{4.8}$$

which can be verified to satisfy the detailed balance condition (2.7). The algorithm was tested on a test function with 8 subfunctions arranged with 2 different ordering of variables designated as tight and loose problems. The paper reported a huge reduction of evaluations required to reach a certain convergence rate as the population size grew. No real parallel data were reported in the paper. In a later paper [30] with a slightly altered algorithm with random mating instead of pairing, these two authors were able to report a speedup of 1009.85 using 32 processors with population size $n = 64$ on the tight problem when compared with the plain serial simulated annealing algorithm. Noticing that the case with population $n = 2$ on a single processor can have a “speedup” of 2.69, this startlingly high speedup is a combined effect of the increased size of population and parallel execution. A more appropriate way of measuring the speedup in this situation is to compare the parallel algorithm to the serial version of same optimal size of population. The resulting calibrated speedup should yield a smaller but more accurate number.
Rudolph [39] presented a neighborhood algorithm that was “architecture aware” in the sense that each processor only performed recombination with its neighbors in the network, and selection is also done by accepting stochastically the best state in the neighborhood. The algorithm was tested on a set of test functions using a $64 \times 256$ node torus with both von Neumann and Moore style neighborhood. The paper only showed that the algorithm is viable, while no performance comparison with other number of processors or other methods was available. Chen et al. proposed a similar algorithm designed for a $128 \times 128$ toroidal SIMD grid [10] that selects mates by randomly determined direction and distance. In addition, its selection process takes the advantage of the deterministic simulated annealing criterion [33]. Although no speedup was reported because no single processor run was feasible, the paper shows, in terms of iterations, an almost linear performance for TSP problems from 256 to 16K processors while achieving about 5 times fewer iterations for ECC design problem on the processor ranges. No wall clock time performance has reported.

The parallel genetic simulated annealing algorithm by Wang et al. is a hybrid algorithm which features sub-population islands [44]. Each processor works on a sub-population rather than a single individual. Periodically, the algorithm selects an individual from each sub-population and migrates it to another sub-population. Within the sub-population, this algorithm employs random mating, Metropolis style selection and geometry cooling. This method is reported to consistently converge faster than the Mühlenbein’s parallel genetic algorithm [34] on a set of test functions using 8 or 16 processors. It also reported that the algorithm is able to gain speedup up to 24 processors on a particular test problem, though no actual speedup number was reported.

Among the papers discussed in this chapter, a few of them studied more than one method of parallelization [9, 26, 36, 37]. These studies provide a great opportunity for horizontal comparison of the different classes of approaches. In terms of the bigger categories, the class of evolutionary annealing algorithm performs the best. It balances the execution time and
the quality of results. In addition, this class of methods can often outperform classic simulated
annealing in even in serial cases due to the introduction of the population [29]. When examining
the individual implementations, Chu et al.'s algorithm offers good quality of results based on its
move control capability. To develop a better algorithm that truly scales beyond the boundary,
all these features needs to be examined in the bigger picture.
5 Discussion

With the level of computing resources available today, applications like linear algebra seem to be able to scale infinitely. The LINPACK efficiency on the top machines of the TOP500 list can routinely exceed 90\% \[43\]. In comparison, despite the large amount of papers and efforts in this field, the performance of the parallel simulated annealing algorithms is hardly satisfactory. There seems to be a barrier that prevents the parallel simulated annealing algorithms from scaling beyond 100 processors. This chapter will discuss the existence and cause of this barrier, and suggest some possible methods with which to break it.

5.1 The Barrier in the Existing Algorithms

Among the parallelization methods reviewed in the previous chapter, many only have been tested for small systems under 32 processors \[2, 8, 12, 29, 27, 29, 39, 37, 44, 44, 43, 49\], though some of those are fairly recent \[8, 27, 44\] and should have access to larger systems. For those which have been tested on moderately large system of between 32 and 100 processors, when ruling out 2 papers which do not distinguish between the algorithmic improvement and parallel improvement \[9, 30\], mediocre speedup is common \[40, 42, 47, 48\]. Even for the two publications reporting the best results \[11, 21\], the performance beyond 100 processors is untested. It is also worth noting that the only two papers which consider truly large scale systems are fairly old
and the results are not solid either. Research on parallel simulated annealing using modern massively parallel supercomputers is surprisingly absent.

Most of the single Markov chain approaches suffer from algorithmic limitations. By definition, these algorithms follow a single Markov chain at every step. Any moves not incorporated into the main chain will be discarded. That will cause a significant waste of computation [9, 40, 45, 47].

The multiple Markov chain approaches, especially the division algorithm and its variants, generally guarantee that the iterations per processor will be inversely proportional to the number of processors involved. That produces implementations with fairly good efficiency [2, 8, 11, 21, 26]. However, the quality of the result becomes a major concern in this class of parallelization. Faster cooling schedules and the reduction of number of states tested because of choosing the best state are both disadvantageous for getting a high quality result. For domain decomposition methods, the chaotic behavior in convergence demonstrated by Macready et al. [28] destroys confidence for the application of that particular method to general problem. A resample in place for selecting the best state as demonstrated by Chu et al. can avoid the quality problem [11], but the quadratic communication requirement will cause overhead when scaling up. On the other side, implementations like the one presented by Higginson et al. with perfect speedup and quality of result may be the result of excessively slow cooling schedule in serial. To reduce such an effect, the serial algorithm should be optimally tuned before being compared by the parallel versions. Among the studies in this particular approach, only the work of Chu et al. has made efforts to address the problem.

Among three big categories of parallelization schemes, population based methods are balanced in terms of the speed of execution and the quality of results. The comparative study conducted by Chen et al. clearly shows the population based method has a much higher convergence rate than the division algorithm [41]. However, the speedup data of this category of parallel algorithms needs closer inspection. The usual way of implementation sets the population size equal or in
proportion to the number of processors used. Thus, improvement over serial annealing algorithm can be caused by a combination of increasing population and increasing number of processors. A better way to measure the scalability is to find an optimal population size for the test problem and fix that size for the parallel runs. But in this way, the speedup is clearly bounded by the optimal size of population of the test problem, thus limiting the scalability of the whole category of algorithms.

5.2 The Road Beyond the Barrier

In order to design a scalable parallel simulated annealing algorithm for optimization problems, the architecture of the target machines must be considered. The computing resources a researcher is likely to have access to today roughly fit into two classes: clusters with a fat tree or star topology network or massively parallel supercomputers with a multi-dimensional torus network. For both architectures, when the number of nodes used become large, the time for a single node to collect data from all other nodes becomes prohibitive. That means a master slave scheme or even a quadratic communication scheme such as all to all communication will cause significant overhead. In this situation, a decentralized algorithm that each only communicates with a small subset of “neighbors” or an asynchronous one will beneficial.

In terms of the algorithmic concerns, the Markov chain analysis as discussed in section 2.1 suggests that convergence is a propriety of the transition matrices as a whole. The cooling schedule and move control are equally important. Szu and Hartley’s work [41] proposed that the increases in the variability of the distribution of moves is able to speed up the cooling schedule while keeping the same theoretical proprieties. However, parallel methods disproportionately focus on the cooling side. Among the studies reviewed, Xavier-de-Souza is one of the few who recognized the importance of move control, but did not actually has take the advantage of it [48]. Only Lam’s algorithm makes explicit use of move control based on past acceptance
ratio to maximize the variance of the energy distribution of the proposed moves[24][25]. Chu’s parallelization scheme, built on Lam’s algorithm, also makes the move control an inherent part of the algorithm and implements a minimal parallel move control method by simply pooling the acceptance ratio. In addition, its implementation choice of using a fixed total segment length summing across the processors to stay close to the serial case puts an artificial limitation on the scalability of the algorithm. A more scalable version should incorporate a more general and dynamic move control scheme.

On the other side, population based approaches introduce the binary crossover operation from evolutionary algorithms. From the perspective of the simulated annealing algorithm, it can be considered to be as a binary move generation scheme that generates larger moves than the traditional unary move generation, and hence addresses the move generation problem in another direction. It is then natural to generalize the move generation scheme to be an operation on multiple states. For better parallel performance, a more accurate parallel move control scheme also needs to be proposed that able to achieve large variance while still maintaining a reasonable acceptance ratio across all processors. That could involve interpolation of statistics based on the search space states explored by every processor.

It could be concluded from the work reported here that the prospects of obtaining scalable parallel simulated annealing algorithms are dismal indeed, and that the method, appearing so promising in the 1980s, has stagnated. Nevertheless, there are two reasons for hope. First, the parallel simulated annealing algorithm is in practice one of two preferred methods for solving large applied optimization problems in systems of 100 processors or fewer. With regard to the 100-processor barrier, it is worth noting that the Metropolis algorithm was first devised at a constant temperature. For the sampling of a Boltzmann distribution at constant temperature, there is no 100-processor barrier. Subject to initial equilibration, the process of generating samples is embarrassingly parallel. What is needed is to require minimal communication between processors while lowering the temperature and staying close to equilibrium. An element of hope
is provided by the physics metaphor itself. Natural systems which are annealed are inherently highly parallel. Capturing the essence of that physical process on a large parallel computer may well show the way forward to truly scalable parallel annealing.
Bibliography


capabilities. In Proceedings of the April 18-20, 1967, spring joint computer conference,


and Lena Chang. A parallel simulated annealing approach to band selection for high-


[43] TOP500.ORG. TOP500 List - June 2012. [http://top500.org/list/2012/06/100](http://top500.org/list/2012/06/100)


