EXPERIENCES WITH SERIAL AND PARALLEL ALGORITHMS FOR CHANNEL ROUTING USING SIMULATED ANNEALING

BY

RANDALL JAY BROUWER

B.S., Calvin College, 1985

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

CHAPTER 1

INTRODUCTION

1.1. Motivation

During the past few years, we have seen the complexity of VLSI circuit designs increase rapidly. One reason for the increase in complexity is the technological advances in the area of mask production and fabrication, making it possible to use smaller and smaller devices. Another reason for the increase in complexity is the automation of the design process, through the use of Computer-Aided Design (CAD) tools. Without the aid of computer programs in the design process, the complexity of the design would be far too great for any engineer to handle.

The design process can be divided up primarily into eight stages as follows [1]:

1) System Specification (Architectural Design I)
2) Functional Design (Architectural Design II)
3) Logic Design
4) Circuit Design
5) Circuit Layout
6) Design Verification
7) Test and Debugging
8) Prototype Test and Manufacture

Stage five of the design process includes the placement and routing of components. There are usually three steps distinguished at this stage, namely :

1) Cell Placement
2) Global Routing of Wires
3) Detailed Routing of Wires

A great deal of research has been directed in these three areas over the last few years in an effort to develop algorithms to perform these complex tasks in a reasonable amount of time. All three of these problems are known to be NP-complete, which means that no known algorithm exists which can optimally solve any of these problems in polynomial (nonexponential) time with respect to the size of the problem. For this reason, all the heuristics and algorithms that have been developed are only able to produce near optimal results.

The detailed routing step can be modeled in many different ways. Some of these ways include:

1) River Routing
2) Channel Routing
3) Switchbox Routing

The focus of this thesis is to discuss a new algorithm for channel routing.

1.2. Channel Routing Problem

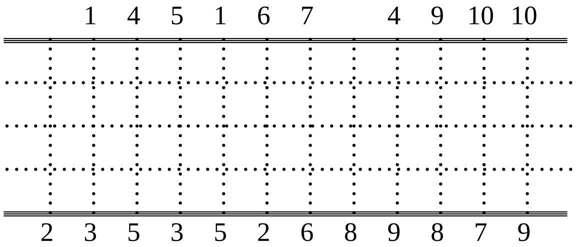
The general channel routing problem deals with placing wires connecting modules of a circuit within a surface area of the chip using the connection layers provided by the given fabrication technology. The surface area can be thought of as a general rectilinear shape, an L shape, a rectangular shape, or any other maskable shape. The wires may be fabricated using any of the connection layers available.

In gate array and standard cell designs, the module placement step determines the positions of cell blocks in predetermined row sites on the chip. Space is provided between the rows of cells to connect terminals of cells to the respective terminals of other cells. These spaces are labeled *channels*. The global routing step then determines which wires to assign to be routed in each of the channels available. Finally, a detailed routing is performed on each channel to select the exact placement of conductors in the channel. These conductors are called *nets*.

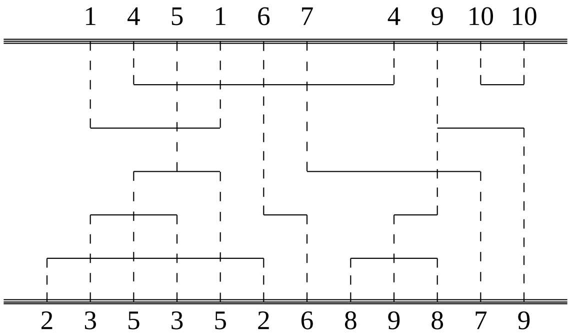
In this thesis, it will be assumed that the channel boundaries form a rectangle, and that the wire terminals are located at uniform spacings (grid based) along the top and bottom edges. Furthermore, only two layers will be used such that all horizontal net segments are routed in one layer and all vertical net segments are routed in the other layer.

Under these assumptions, the goal then is, given a sequence of net terminals along the top and bottom borders of a rectangular channel, to determine a placement of the net segments so as to minimize the size of the channel space and length or resistance of all connections made. An example of a terminal assignment for a channel is shown in Figure 1.1(a). Figure 1.1(b) shows one possible routing of the previous channel. In this figure, horizontal segments are shown in solid lines, vertical segments in dashed lines.

The *channel density* is defined as the theoretical minimum number of tracks required to successfully and completely route a given channel. The density of any column is easily computed by counting the number of nets that must pass through the given column. The *channel density* is then the maximum column density of all



(a) Terminal Assignment

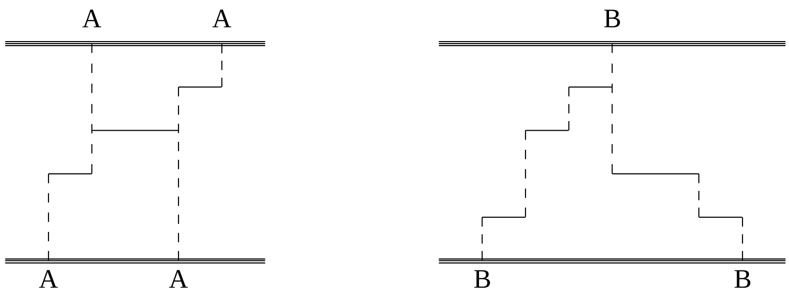


(b) Possible Routing

Figure 1.1. Example Channel with Density 5

columns of the channel. Since this number is channel dependent, it must be calculated for each problem.

Doglegging is a term used to describe nets that occupy two or more tracks of the channel. Each net consists of a set of horizontal segments and a set of vertical segments. There are two forms of doglegging: restricted and unrestricted. *Restricted doglegging* only allows a net to be split into two tracks at a column in which a terminal of the net is found. A simple way to model this is to break nets with more than two terminals (multiterminal nets) into two-terminal subnets. This is shown in Figure 1.2(a). Each subnet is free to occupy any track of the channel, and separations of tracks will automatically occur at the columns in which terminals are found. *Unrestricted doglegging* allows a net to be split so that it occupies two tracks at any point along the channel. This is shown in Figure 1.2(b).



(a) Restricted

(b) Unrestricted

Figure 1.2. Doglegging Examples

One effective graphical technique used to determine relative positions of nets with respect to each other is the *vertical constraint graph (VCG)*. Each net of the channel is represented by a node in the graph. A directed edge from vertex i to vertex j indicates that in column c of the channel a terminal pin for net i is located along the top of the channel and a terminal pin for net j is located along the bottom of the channel. In order to avoid overlap between the vertical segments of nets i and j, the track selected for net i must lie above the track selected for net j. Figure 1.3 shows the vertical constraint graph for the example channel of Figure 1.3.

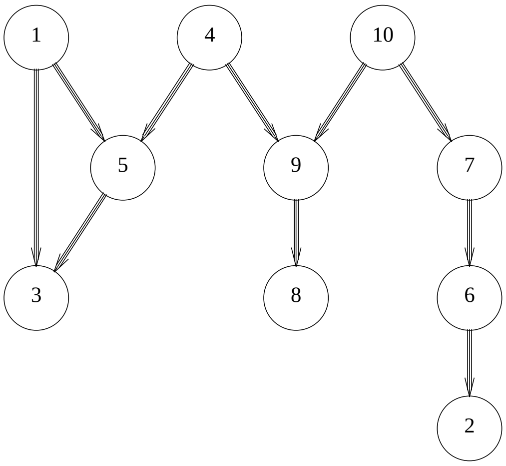
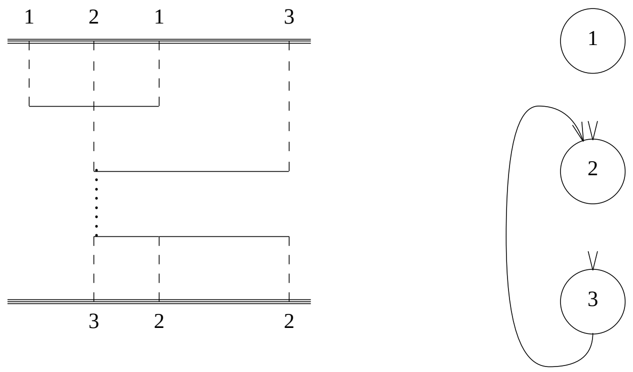


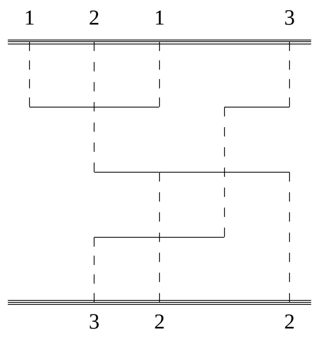
Figure 1.3. Vertical Constraint Graph

If a cycle exists in the vertical constraint graph, then it is impossible to successfully route the channel without allowing unrestricted doglegging. Figure 1.4(a) shows a channel example in which there is a cycle in the VCG, and Figure 1.4(b) shows how unrestricted doglegging is used to avoid the cyclic constraints.

The channel routing problem described above has been proven to be N-P complete.



(a) Example



(b) Solution

Figure 1.4. Cyclic Constraint Problem

1.3. Previous Work

The channel routing problem, in all of its various formulations, has been a focus of much research interest for the past 15 to 20 years. Most of the earlier work was directed toward wire routing of multilayered printed circuit boards. After the introduction of LSI and VLSI fabrication methods, research intensity increased, with many new ideas presented.

One of the first algorithms presented was *Lee's Maze Router* [2]. Lee based his algorithm on the idea of wavefront expansion of a single net and selection of the shortest path found between the source and sink terminals. Some of the problems with this method include the large amount of memory required, the often inadequate wiring of the last nets to be placed, and the tendency to use excessive numbers of vias. Originally, the algorithm was intended for PCB routing, but was easily applied to rectangular channels in integrated circuits.

The next major contribution was nearly ten years later when Hashimoto and Stevens [3] introduced the *Left-Edge Algorithm*. This algorithm routes one track at a time, trying to maximize the use of the space in the current track. Nets are placed in a left-to-right fashion until the track is filled. The algorithm's performance is strongly dependent on the order in which nets are placed and the presence of vertical constraints in the channel routing problem.

Deutsch made several improvements to the Left-Edge Algorithm in his *Dogleg Channel Router* [4], most notably being his inclusion of doglegging. Through an effective use of doglegging and other improvements, he was able to achieve better results than with the simpler Left-Edge Algorithm.

A new approach taken by Yoshimura and Kuh [5] derived routing heuristics from graph theory concepts. Nets are first grouped according to the vertical constraint graph and an interval graph based on horizontal constraints. Next, merging takes place between groups of nets to minimize the longest path in the modified vertical constraint graph. Their results demonstrated a large improvement over the Dogleg and Left-Edge Algorithms, especially in the minimum number of tracks required and overall processing time.

Around the same time, another heuristic-based router was developed by Rivest and Fidducia [6] called the *greedy channel router*. This router applies the same principles as the Left-Edge and Dogleg routers do; however, the channel is scanned on a column-by-column basis instead of track-by-track methods of the former. Unrestricted doglegging is allowed; however, it may be necessary to add extra columns on the end of the channel to complete the wiring.

Another approach, which combines aspects of both track-by-track and column-by-column routers, was presented by Sangiovanni-Vincentelli and Santomauro, called *YACR2*, for "Yet Another Channel Router 2" [7]. Instead of requiring extra columns at the end of the channel, this router may require extra columns in the middle of the channel.

A new approach, taken by Burstein and Pelavin [8], applies linear and dynamic programming to the channel routing problem which is decomposed hierarchically. The results they have presented show a further improvement over previous channel routers.

A far different approach was proposed by Joobbani and Siewiorek [9]. They have applied principles of artificial intelligence and expert systems to the channel routing problem. The task of channel routing is divided into subtasks which are assigned to subtask experts. The efforts of these experts are then coordinated to produce high quality channel routings.

Shin and Sangiovanni-Vincentelli developed *MIGHTY: A 'Rip-Up and Reroute' Detailed Router* in 1986 [10]. *Mighty* is a very powerful router, able to route channels of various shapes, including switchboxes. *Mighty* is a two-layer router; however, vertical routing is not restricted to a single layer and horizontal routing to the other layer. Heuristics are applied for placing nets one at a time, displacing some nets slightly to make room for blocked nets, and ripping up some nets currently placed to allow other nets to be placed first.

Finally, another approach was taken by Leong, Wong, and Liu[11] through the application of a new optimization technique called simulated annealing. Their routing program produced very good results; however, the program run time was far too long.

The above papers were chosen because they represent the major research contributions and directions taken in channel routing over the past few decades. There have been many other papers published not mentioned that discuss improvements to previous algorithms, theoretical bounds on channel routing, and less restricted problem statements (including gridless and multilayered channel routing). For a good set of references on channel routing, see the introduction to the book by Hu and Kuh [1].

1.4. Thesis Outline

In the remainder of this thesis, research in serial and parallel algorithms for channel routing based on the simulated annealing methodology will be presented and discussed. This thesis is organized as follows. Chapter 2 discusses the simulated annealing methodology and how it is applied to channel routing. First, simulated annealing is presented as a recent approach to solving multivariate optimization problems. Next, a previous application of simulated annealing to channel routing is discussed in detail. Subsequently, our approach to channel routing is presented. Finally, we discuss how to apply this approach to the basic channel routing problem and also to extensions of channel routing which include unrestricted doglegging, obstacle avoidance, and switchbox routing. Chapter 3 will first present the details of the serial implementation of the channel router. After discussing some of the heuristics used, the results of the serial version will be given. Chapter 4 will present the details of the parallel implementation of the channel router. The targeted parallel machine will be described, followed by descriptions of how the problem was partitioned onto the parallel architecture. Some of the heuristics will be discussed, and then the results will be presented. Finally, Chapter 5 will summarize the research accomplished and draw some conclusions from the work.

CHAPTER 2

SIMULATED ANNEALING

2.1. Simulated Annealing Methodology

In 1983, Kirkpatrick, Gelatt, and Vecchi [12] demonstrated the similarities between statistical mechanics and multivariate or combinatorial optimization and proposed a technique for optimization. Their technique, called *simulated annealing*, is analogous to the process of slowly cooling a bar of metal so that large uniform crystalline structures are formed. These crystalline structures represent the lowest possible energy states for that material. The probability of a given state x_i with energy $E(x_i)$ is given by

$$P(x_i) = e^{-E(x_i)/k_b T}$$

where k_b is Boltzman's constant and T is the absolute temperature.

To simulate the annealing process of metals, one must first determine how the state of a system is defined. A methodology for permuting one state into another must be outlined. The selection of components to move can be made in either a purely random fashion or by applying specific heuristics generated for the problem at hand. An approximation to the energy of the states must also be formulated, usually in the form of a cost function that accurately represents the criteria to be minimized. Finally, a simulated temperature range and a schedule for decrementing the temperature must be selected to achieve an optimal cost-to-temperature ratio throughout the annealing process.

In 1953, Metropolis *et al.* [13] outlined an efficient procedure for deciding whether or not a given state will exist at a given temperature. At each step of the annealing process, a pseudo-random move is made and the resulting change in cost ΔC from the previous state to the new state is calculated. Then, the probability of accepting the new state is

$$\begin{cases} 1, & \Delta C \leq 0 \\ e^{-\Delta C/T}, & \Delta C > 0 \end{cases}$$

With this negative exponential function, it is very likely that new states causing a cost increase will be accepted at high temperatures, but not at low temperatures. Figure 2.1 shows the generalized simulated annealing algorithm which can be applied to many different problems.

Set Initial Temperature and State
WHILE (Stopping Criteria Not Satisfied) **DO**
    FOR Inner_Loop_Count = 1 **TO** MAX **DO**
        Select Elements to Move
        Select Move Operation
        Calculate Cost Change
        Evaluate Accept/Reject Based on Temperature and Cost
        IF (Accept) **THEN**
            Update State Information
        Adjust Temperature
    END Inner Loop
END While Stopping Criteria Not Satisfied
Display Final Results

Figure 2.1. General Simulated Annealing Algorithm

It is precisely this aspect of simulated annealing that makes it attractive over other optimization methods. Nearly all of the channel routers presented in the first chapter apply a set of heuristics in solving the problem. The problem with simply using heuristics is that they can easily lead the optimization to a local minimum which could be far from the optimal solution of the problem. Once the local minimum is found, these algorithms are stuck. Simulated annealing allows one to get out of local valleys. Figure 2.2 graphically shows local and global minima in a typical optimization problem.

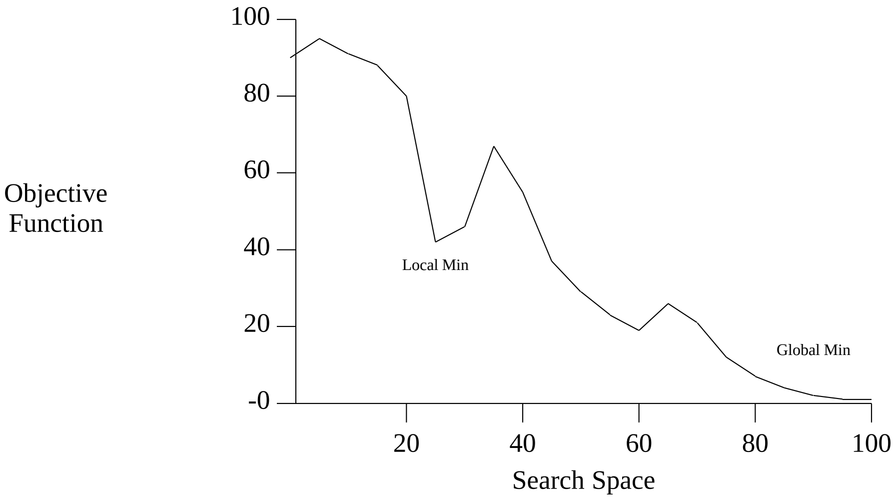


Figure 2.2. Local and Global Minima in an Annealing Cost Function

However, there is a price to be paid. Simulated annealing is basically an iterative trial-and-error algorithm, and calculating each cost change could be expensive in time and computing resources. It is critical, therefore, to determine the cost criteria carefully and efficiently.

In their original paper, Kirkpatrick, Gelatt, and Vecchi showed how to apply simulated annealing to the problem of chip partitioning, cell placement, global wiring, and the classical *Traveling Salesman Problem*. Other researchers have applied simulated annealing to logic minimization [14], cell placement [15, 16], global routing [17], and detailed (channel) routing [11] since then. Furthermore, many of the simulated annealing algorithms have been parallelized with some very interesting results. Some of these include partitioning and routing [18], standard cell placement [19-21], macro cell placement [22], and floorplanning [23].

2.2. Simulated Annealing Applied to Channel Routing

As was noted above, determining the optimal assignment of nets in the tracks of a channel has been proven to be an NP-complete problem. Although many people have reported good results from applying heuristics to the problem, we feel that far better results in general may be attained by applying simulated annealing. Heuristic algorithms are usually greedy algorithms in the sense that only downhill moves are accepted. If a local minimum is encountered anywhere, these algorithms will accept that state as the minimum, even though a better state may exist.

2.2.1. The first simulated annealing channel router

In 1985, Leong, Wong, and Liu presented the first channel routing algorithm based on simulated annealing [11]. Their algorithm borrows ideas from the net merging router of Yoshimura and Kuh [5]. All nets of a given channel are divided into subnets and the vertical constraint graph G is formed. This graph is then partitioned into groups in which subnets in one group represent subnets placed in the same routing track with no horizontal overlap incurred.

One of three different types of operations is then chosen randomly and applied to randomly chosen subnets to create a new channel state. These operations (or moves) include displacing one subnet from one group to another, exchanging two subnets in different groups, and extracting a subnet from a group to form a new group. Further, only legal moves are permitted; at no time will a move that creates overlap between two subnets be allowed. Leong [24] has demonstrated that this set of moves is sufficient to perform all necessary permutations on the state of the channel.

The cost function applied to determine the acceptance or rejection of a move is a combination of three characteristics of the current and new states of the channel. These are

1) Channel Width
2) Longest Path in $\hat{G}$
3) Track Sparsity

The channel width requires no calculation, the longest path is found by searching the modified vertical constraint graph $\hat{G}$, and the sparsity of each individual track must first be calculated to find the overall sparsity of the channel.

Since annealing takes a long time to complete, one option is to parallelize the process. If moves are to be attempted in parallel, some mechanism must be used to prevent two separate moves from causing an illegal channel state. An example of how this might occur is shown in Figure 2.3.

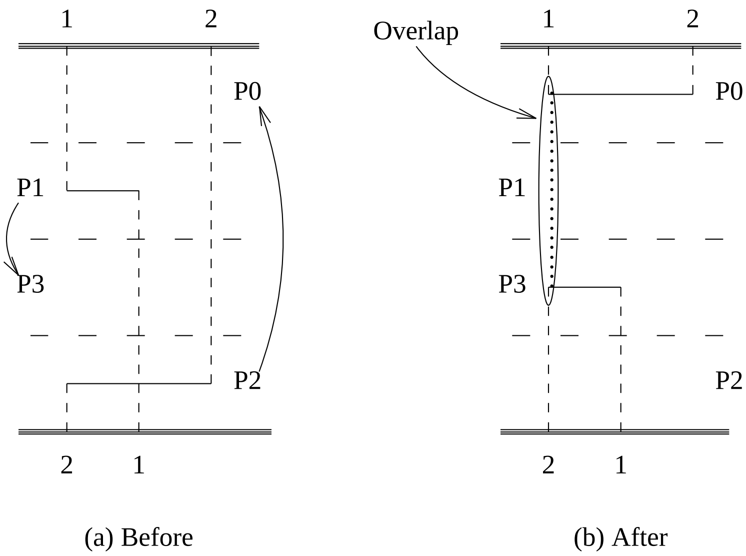


Figure 2.3. Example of Illegal Move

Without shared data, there is no easy way to have parallel selection of subnets and moves. Also, after a set of moves is attempted in parallel, the modified vertical constraint graph, $\hat{G}$, must be modified to reflect each move. This is done before acceptance decisions are made, and if a subset of moves is rejected, reformulation must take place. Furthermore, it is difficult to incorporate avoidance of obstacles, such as power and ground wiring, into the vertical and horizontal constraint checking.

For these reasons, we decided to investigate alternative approaches to applying simulated annealing to the channel routing problem.

2.2.2. A new simulated annealing algorithm for channel routing

The algorithm presented here is less restrictive during the annealing process than the algorithm of Leong, Wong, and Liu. First, an algorithm for channel routing is presented. Second, extensions of the channel routing algorithm to include unrestricted doglegging, obstacle avoidance, and switchbox routing are described.

2.2.2.1. Channel routing

Four aspects of our channel router will be discussed here. The first is the set of moves permitted to operate on a given channel state. All moves of subnets are legal. We do not distinguish between moving a subnet to an empty area of a track or moving it on top of another subnet, creating overlap that must later be removed. Overlaps of subnets are handled during the evaluation of the cost function. A similar idea was successfully used in the *Timberwolf* cell placement program based on simulated annealing [15].

There are two basic move types used, displacement and exchange. Displacement moves allow a subnet residing in a given track T_i to be moved to track T_j. Track T_j is either an existing track, or a new track. Displacing subnets to existing tracks is the source of the majority of the improvements made to the channel state. It is possible through this move to eliminate tracks completely by moving all subnets in the track to other tracks. If the annealing process gets stuck at a local minimum, displacing subnets to a new track can be used to free up the channel enough to get out of the local minimum.

The second set of moves permitted is exchange moves. These moves are also used to provide more freedom to the annealing process. Although no tracks are ever freed up by this move, exchanging two subnets does provide improvements in cases where a sequence of two moves is necessary. All exchange moves can be subdivided into a sequence of two displacement moves. The first part displaces one subnet into the track of the second subnet, usually causing horizontal overlap between the two subnets. The second part displaces the second subnet to the original track of the first. Since overlap is usually induced momentarily, the first displacement would be accepted with an extremely low probability. Thus in situations exemplified by Figure 2.3, it is far better to use the exchange move than two displacement moves.

The second aspect of our channel router is the cost function used for calculating the cost of a new channel state after randomly selected moves have been applied to the current state. Since the goal of our channel router is to provide a near optimal routing of the given channel, the cost for a given state of the channel is a function of the amount of overlap between unique nets (OL), the length of the nets (NL), the width of the channel (WC), and the fraction of the track not occupied by nets (FU). For each proposed move, the cost change incurred if the move was to be accepted is calculated as follows and used to evaluate move acceptance:

$$\Delta COST = \alpha\times(\Delta OL) + \beta\times(\Delta NL) + \gamma\times(\Delta CW) + \delta\times(\Delta FU)$$

It is necessary to adjust the values of the parameters α, β, γ, and δ to optimize the annealing process. These values should be determined through a study of numerous trials on a variety of problems.

The third aspect of our channel router is the data structure employed. Since overlap is an important part of the cost function and requires the most computation, the design of the data structure should concentrate on providing efficient calculation mechanisms. Each net occupying a given track is given a structure in a linked list that specifies the grid points of the left and right endpoints of the subnet segment found in the horizontal track. Each track list is linked with the list of the preceding track to form a two-dimensional linked data structure. The subnet structures in each track list are also sorted by leftmost gridpoint value so that searches may be terminated early without traversing the entire linked list. Linked list structures are used for the track data because the number of subnets in a track varies greatly from track to track, along with the total number of tracks varying throughout the annealing process.

For the vertical segments of subnets placed in specific columns, there is no need for linked lists (at least not in the case of channel routing) and so dynamically allocated column structure arrays are used. The number of columns is always fixed, and each column has exactly two endpoints where net terminals are located. The only other way to place more nets in a column is by unrestricted doglegging. Since those numbers are very small it is possible to use fixed sized arrays. Figure 2.4 shows the linked list structure used for the track data.

Finally, the fourth aspect of our channel router is the annealing schedule used. Many researchers have investigated optimal and efficient cooling schedules for annealing processes. The cooling algorithm can be modeled by Markov-Chains. One method has been developed to approximate the optimal cooling schedule by analyzing fixed-length Markov-Chains in polynomial time [25]. Another method attempts to control convergence by adjusting the temperature so that the average cost decrease is uniform [26].

Initially we decided to take a simplified approach by applying a predefined temperature adjustment schedule. The annealing temperature T is adjusted based on the following schedule:

$$T_{i+1} = ALPHA(T_i) \times T_i$$

in which the function $ALPHA(T)$ ranges from 0.8 for large values of T to 0.95 for small values of T. This schedule allows more permutations at low annealing temperatures to make many small improvements. To

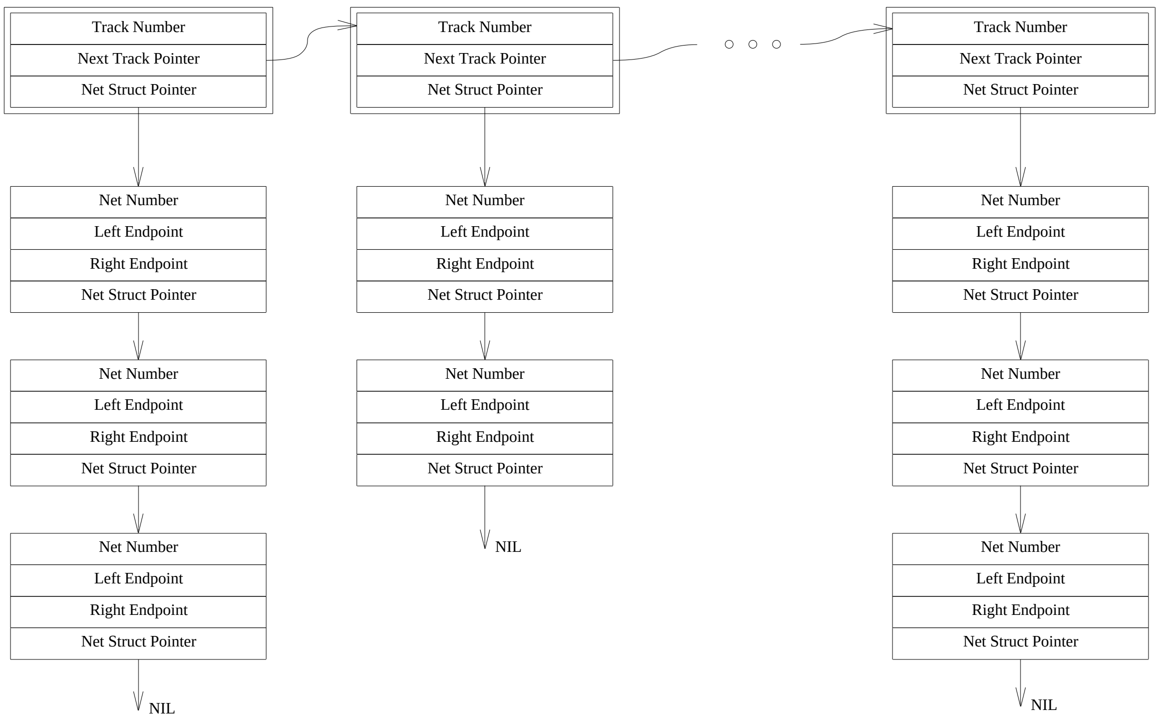


Figure 2.4. Track Data Linked List Structure

determine the initial temperature, 100 random moves with a positive cost change are evaluated without accepting any of them. The average cost change $\Delta COST_{AVE}$ for those moves is then calculated and T_{INIT} is solved for as follows:

$$T_{INIT} = -\frac{\Delta COST_{AVE}}{ln(0.9)}$$

The value 0.90 is used because at the initial temperature we would like to accept 90% of all moves attempted.

2.2.2.2. Extensions to the channel routing algorithm

The algorithm presented above can easily be extended to include unrestricted doglegging, obstacle avoidance, and switchbox routing.

To allow unrestricted doglegging it is necessary to add two more move types to the set already used, one to split a selected subnet into two different tracks at a selected column, and one to restore a separated subnet back into a single track. Furthermore, a penalty or cost should be assessed to any move that creates unrestricted doglegs because of the additional vias required. In cases where cycles are found in vertical constraint graphs, it is necessary to allow unrestricted doglegging.

Since overlaps are allowed during the annealing process, the algorithm is also well suited for extending to include obstacle avoidance. Obstacle avoidance is important to consider if some sections of the routing area could be used for power or ground routing or any other element of the chip that must be placed there. By applying a very high cost to any subnet occupying those areas it is possible to retain the necessary freedom for the subnets at high temperatures to be placed almost anywhere, and then as the temperature is reduced, those interferences can gradually be eliminated.

Switchbox routing is similar to channel routing, except nets are given terminals on all sides of the rectangle instead of just two sides. Although this problem is much more difficult than the channel routing problem, it is not as difficult to extend our algorithm to handle switchboxes. Since there are many more constraints on the placement of subnets, it is even more important to allow the subnets to overlap during high temperature annealing. In some sense, at high temperatures it appears that each subnet is being placed in the best location independent of all other nets around it, and as the temperature is reduced, the effect of the other nets is slowly increased. The linked list representation of the track data could easily be replaced with a representation similar to that used for the column data. Unrestricted doglegging would have to be included to successfully route nearly all switch-box examples.

CHAPTER 3

SERIAL IMPLEMENTATION

3.1. Implementation Details

The algorithm presented in the previous two chapters was implemented as a serial version with approximately 7,000 lines of C code and was executed on a Sun Microsystems 3/50 workstation under Sun UNIX 4.2, Release 3.4.

3.2. Heuristics

In the following we will discuss various heuristics used for different characteristics of the annealing algorithm. After a simple initial implementation of the simulated annealing algorithm, it was clear that many more improvements on the algorithm would have to be made. The initial placement of nets and selection of moves, nets, and tracks for displacement were originally made in a purely random fashion. It is necessary to include some intelligent heuristics to all of the selection processes in order to achieve convergence within a reasonable amount of time. In the following pages, we will attempt to describe those heuristics that were applied to the uniprocessor implementation.

3.2.1. Initial placement

One simple heuristic was used for the initial placement of the nets into tracks. First, nets with all terminals on the top border of the channel are placed in unique tracks. No horizontal overlap is created because subnets in the same track always belong to the same net. Next, all nets that have terminals along the top and bottom borders of the channel are placed, one per track. Finally, all nets that have all terminals along only the bottom border are placed, one per track.

3.2.2. Move selection

Sechen [15] reported that for a simulated annealing algorithm for standard cell placement, the number of displacement moves should outweigh the number of exchange moves. The ratio used was 5:1 in favor of displacements. After a series of tests, we found that for channel routing, a ratio between 15:1 and 20:1 produced better results.

After further analysis of the moves selected at low temperatures, it was decided that exchange type moves should be eliminated for temperatures below a given threshold. The cost function used is able to accurately predict overlap for a given subnet displacement, but due to the complexity of the calculations, the overlap between exchanging subnets is only approximated. Because of this, overlap could mistakenly be created at low temperatures, not allowing enough time for the annealing to gradually clear it out.

3.2.3. Net selection

Net selection could be one of the most important aspects of the annealing process. If the best placed subnets are always selected to be moved, it will be impossible to make any progress. Originally, the subnet to be selected was drawn at random from the set S of subnets. This approach is analogous to walking a random path in a forest, hoping to find the way out.

One solution to the problem is to apply a weighting to each subnet in the set S, forming $\hat{S}$. Subnets currently incurring some overlap should be weighted much higher than subnets with no overlap. This can be reflected by adding a cost term proportional to the amount of overlap the given subnet has. The subnets with overlap should be selected more often, and overlap should be quickly eliminated. A similar idea was also use by Kling and Banerjee [27] for selecting the queue ordering of modules in an evolution-based standard cell placement (ESP) program.

Another possible factor that could be included in the selection of subnets is the current position of the subnet with respect to the best possible placement of that subnet taken individually. This idea applies primarily to "n" and "u" shaped subnets as shown in Figure 3.1, or in other words, subnets having either both terminals along the top border of the channel or both terminals along the bottom border. There are two reasons for wanting these types of subnets drawn to their respective borders. The first is that it frees up the central tracks so that other subnets having both top and bottom terminals may use those. The second reason, more importantly, is that it shortens the length of the conducting wires of those subnets, reducing the resistance and propagation delay. We decided to add another term to the approximated subnet cost to reflect the excess length that is proportional to that length. During the high temperature ranges of the annealing process, the effect of the length is much less than the effect of the overlap, so to save computation time, the length computation is only added below a given

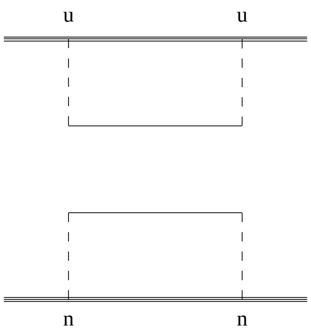


Figure 3.1. "n" and "u" Shaped Subnets

temperature threshold.

One other subnet selection biasing technique is to increase the probability of selecting subnets in nearly vacant tracks. If it is possible to displace a subnet out of an almost vacant track, then it might be possible at the same time to eliminate that track and decrease the channel width.

3.2.4. Track selection

Selecting the track to displace a subnet to is also a very important decision. Purely random selection is simply not enough to secure improvements quickly, especially at high temperatures when bad track selections are accepted equally well as good track selections. Note, however, that it is important not to eliminate "bad" moves because they are an integral part of the annealing process.

The first method for biasing track selection to consider is to increase the probability of selection of a track based on track vacancy. Since subnets are selected to vacate nearly empty tracks, the track selected for displacement to should be reasonably full, or few gains are made.

Another heuristic applied to "n" and "u" shaped subnets is to bias the displacement track selection toward those tracks on the inside of the subnet. This approach should encourage such subnets to move toward the border tracks to reduce wire length and congestion. Again, it is important not to over-bias the selection because it is absolutely necessary that some subnets move away so that better subnets can be moved inward.

In physical annealing, during very low temperatures, molecular movement is usually limited to a very small area around the molecule's current position. This same idea has been applied by many in standard and macro-cell placement by simulated annealing [20, 22, 28]. The idea can take on two forms: One, a fixed sized window enabled for temperatures below a threshold, and two, a variable sized window proportional to the temperature. The first is the easiest to implement, but the second is better suited to annealing because of its gradual changes. A thorough testing was not done to determine the feasibility of either approach; further research in this area is necessary.

A more accurate way of determining which track to choose is to evaluate the anticipated overlap and wire length changes that would occur for each track under consideration. This estimated cost is then used to find the weighted probability of selection for each track. Although this is one of the better heuristics, it is also very costly in computation time.

3.3. Results

Due to the large number of variations possible in heuristics, a thorough testing of each heuristic independently was impossible. Many trial runs were performed combining many of the heuristics together and adjusting the parameters and heuristics by analyzing the output of each run. Instead of listing the results of every trial, this section will present the results of applying some of the "better" heuristics to one channel example in particular.

The ratio of exchange to displacement moves to exchange moves was 15:1. The threshold temperature for cutoff of exchange moves and including net length in the cost calculations was 20.0 At each temperature 500 iterations were performed. The density of the channel was 12, and there were 21 nets broken up into 39 subnets.

The weighting applied to each subnet was a function of the overlap, the current track vacancy, and if below the threshold, the excess length of the subnet. Subnets for displacement and the first subnet for exchanging were selected randomly biased by the calculated weighting. The second subnet selected for exchanging was biased by precalculating the resulting overlap for each eligible subnet.

The tracks for displacement were biased by calculating the expected overlap if the track was selected and adding a constant factor to bias "n" and "u" shaped nets toward the appropriate border. No windowing was used in selecting either the tracks or subnets.

Figure 3.2 shows the final solution for the 12 track example. Figure 3.3 shows the annealing cost with respect to temperature for that example. Figure 3.4 shows the average overlap with respect to temperature. Finally, Figure 3.5 shows the average number of tracks with respect to the temperature.

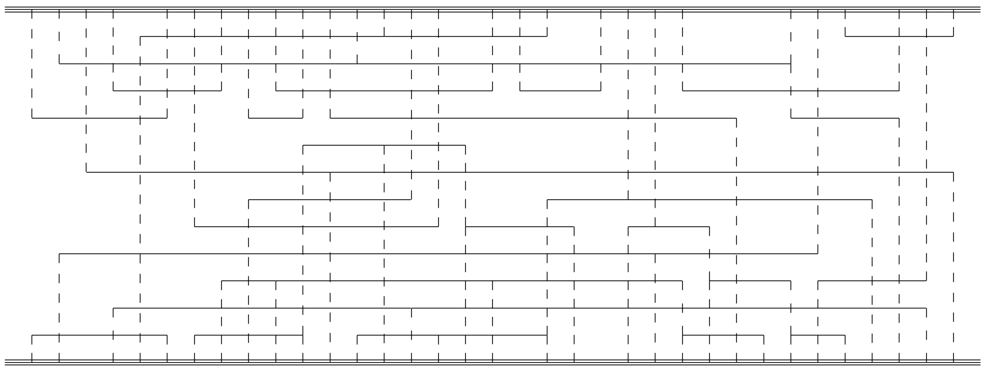


Figure 3.2. Final 12 Track Solution -- Serial

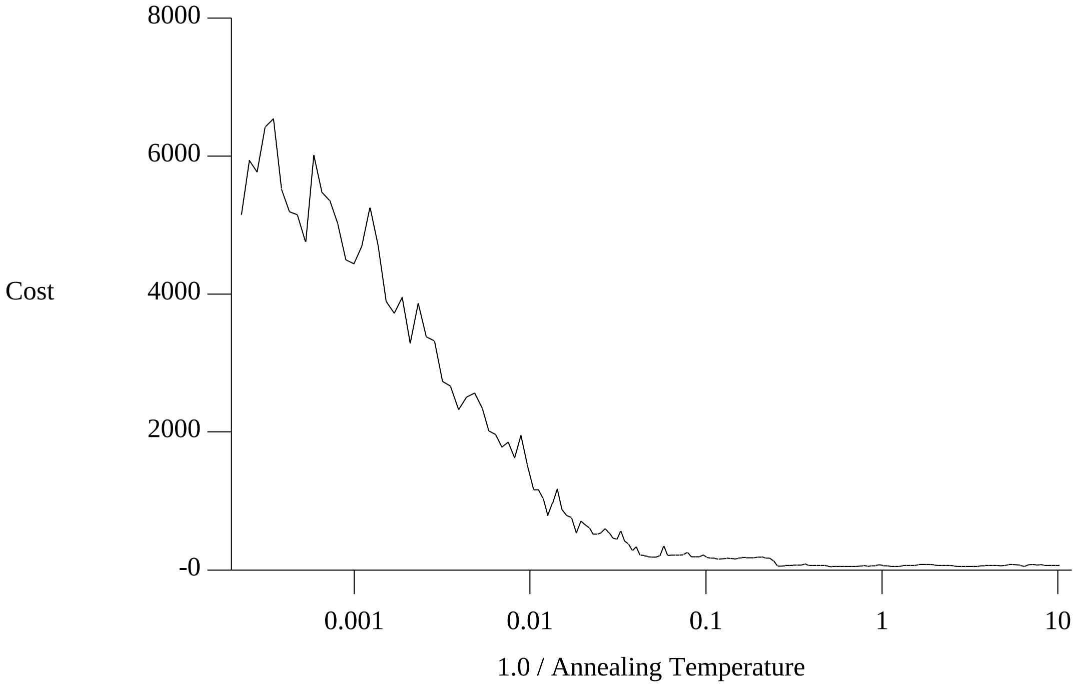


Figure 3.3. Annealing Cost vs. Temperature -- Serial

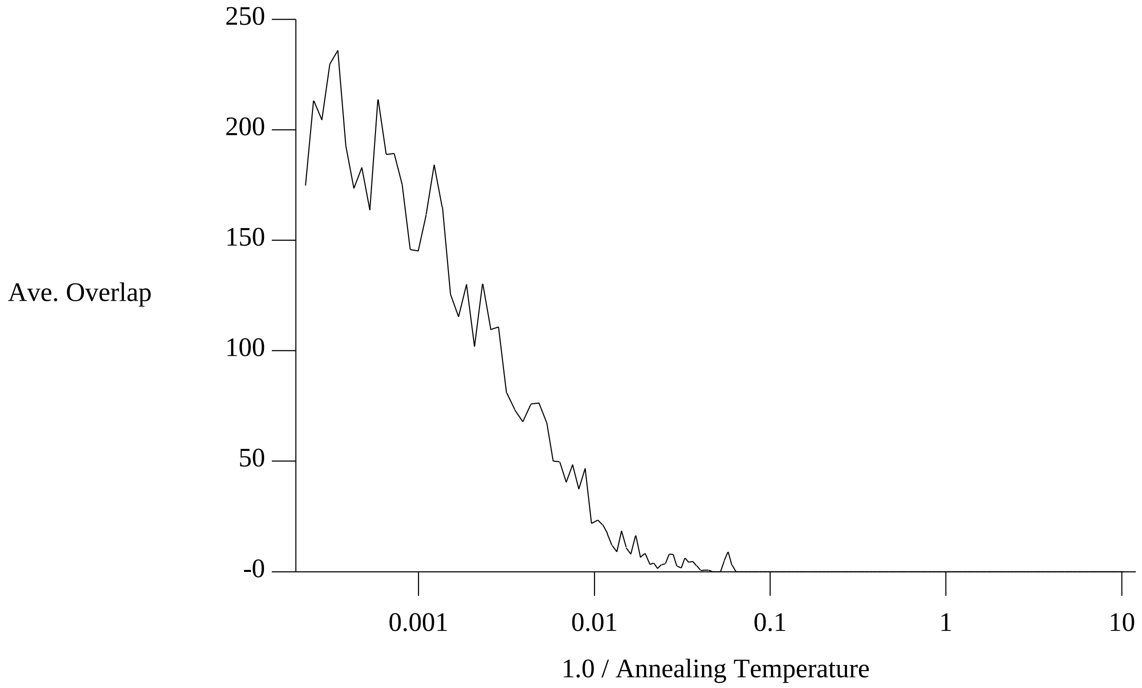


Figure 3.4. Subnet Overlap vs. Temperature -- Serial

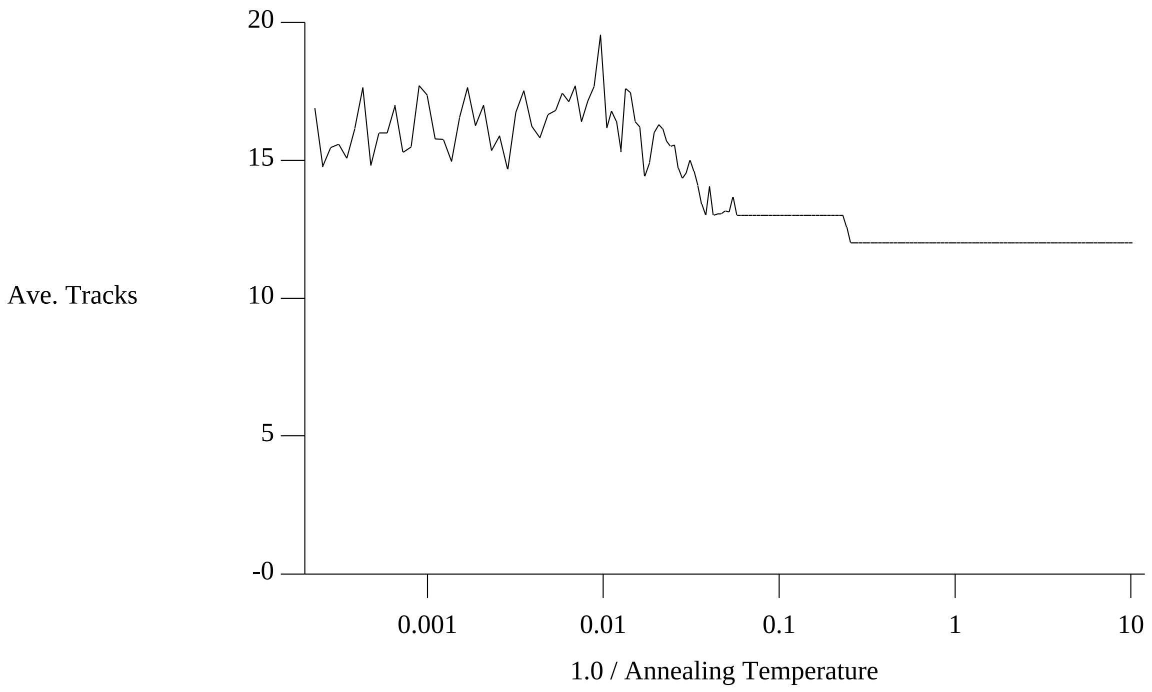


Figure 3.5. Average Number of Tracks vs. Temperature -- Serial

CHAPTER 4

PARALLEL IMPLEMENTATION

Once we demonstrated the viability of our simulated annealing approach to solve the channel routing problem, we decided to implement a parallel version which was the original intent of this thesis. The parallel algorithm would serve to cut down the run time of the algorithm. The machine for which the parallel version is targeted is the Intel iPSC Hypercube. The iPSC was chosen because one machine is readily available for use here at the University of Illinois for testing. However, even though a system was available, due to lack of time and resources, no testing could be performed on it. Instead, simulations were carried out using the Intel Hypercube Simulator, version 3.0 running on a Sun Microsystems 3/50 workstation.

4.1. Hypercube Architecture

A hypercube computer is a collection of $P = 2^N$ processor nodes interconnected by a binary N-cube topology. Each node of the hypercube is a self-contained computer with a cpu, memory, and communication hardware. Each node can communicate directly with exactly N neighbors through communications channels connecting adjacent nodes. Figure 4.1 illustrates a four-dimensional (16 node) hypercube, showing the nodes and communication channels between them. Each node is labeled with a unique N-bit binary number so that adjacent node numbers differ in exactly one bit position.

The *diameter* of a network is defined as the maximum number of hops required to send a message between any two nodes, and the node *connectivity* is the maximum number of communication lines required for any single node. For the hypercube, the diameter and node connectivity are both $\log_2 P$. The hypercube offers a good balance between node connectivity and communication diameter. Furthermore, the topology of the hypercube allows a user to embed many different communication mappings such as meshes, trees, linear arrays, and smaller dimensional cubes. Each of the sixteen nodes of the available iPSC contains an Intel 80286 cpu, an Intel 80287 numeric coprocessor, 4.5 MBytes RAM, and communication hardware based on the Intel 82586 Ethernet Controller Chip. It is possible to have up to a seven-dimensional iPSC hypercube; however, such an array is difficult to draw and harder to visualize.

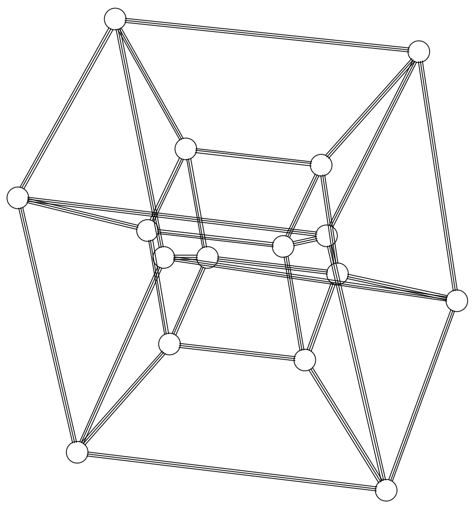


Figure 4.1. Four-Dimensional Hypercube

The iPSC cube nodes are connected to a System Manager computer through which a user can interface. The cube manager is made up of a monitor, hard and floppy disk drives, and ethernet ports for connecting to both the cube and other computers on a local area network.

4.2. Hypercube Software

A typical program to be run on the iPSC is made up of two separate executable parts. One part, called the host program, is executed by the *host* or System Manager provides the user interface, file access, and downloading of the node program to each node. The second part, called the node program, is executed by each node in parallel. Since the hypercube is a message-passing based architecture, special constructs and functions are used to establish communication for the node with the host and for the node with other nodes.

The functions used for sending and receiving messages have the form:

send(ci, type, buf, len, node, pid);
recv(ci, type, buf, len, &cnt, &node, &pid);

where

ci = Channel identifier for the channel to transmit the message on

type= Type of message being sent or waiting to be received

buf = Starting address of buffer to read message from or to write message into

len = Number of bytes to send or the size of the receive buffer

node= Number of node to send to or number of node received from

pid = Process id of process sending message

cnt = Number of bytes actually received

Other functions are available for reading the clock, checking the status of a channel, writing to a logfile, and some diagnostic functions.

4.3. Intel Hypercube Simulator

The Intel Hypercube Simulator is a tool distributed by Intel to provide the user with an environment for developing and debugging programs written for the hypercube. The simulator simulates the actual hypercube by forking a UNIX process for each node. Communication between nodes is simulated by using UNIX pipes and signals. Aside from a few minor limitations, Intel claims that programs successfully run on the simulator will run on the hypercube with few to no changes.

The material for the preceding sections of this chapter was taken from [29-34].

4.4. Implementation Details

Before developing an algorithm for implementation on a hypercube, one should consider first the number of processors required, how the problem can best be partitioned, how to map that partition onto the hypercube, and what data structures would be most efficient for such an implementation [35]. Given a highly parallelizable problem like matrix multiplication, choosing the right partitioning, mapping, and data structure could greatly affect the performance of the implementation. For example, partitioning the data of a matrix according to the back diagonals of the matrix would not make any sense. For this reason, care must be taken in developing the parallel algorithm and implementation. The parallel algorithm implementation for channel routing is outlined in Figure 4.2, and will be discussed in more detail in the following sections.

4.4.1. Selected topology

Since the hypercube topology can be used to embed many other topologies, we choose to map the processors into a linear array as shown in Figure 4.3. The lines and arcs on the figure show the communication chan-

```
Determine Initial Annealing Temperature and Parameters
Make Initial Track Assignment to Each Processor of Hypercube
WHILE (Temperature > ε) DO
    FOR Inner_Loop_Count = 1 TO MAX_I DO
        FOR Cube_Dimension = 0 TO log(P) − 1 DO
            Randomly Select One Subnet in Each Processor in Parallel
            Randomly Select P/2 Moves For Node Pairs of Cube_Dimension in Parallel
            Evaluate Cost Change for Each Move Between Pairs of Nodes in Parallel
            Evaluate Accept/Reject Based on Temperature and Cost in Parallel
            IF (Accept) THEN
                Update Local State Information
            Broadcast Updates to All Other Nodes
        END Dimensions of Cube
        Adjust Temperature in each Node
    END Inner Loop
END While Stopping Criteria Not Met
Display Final Results
```

Figure 4.2. Parallel Algorithm for Channel Routing

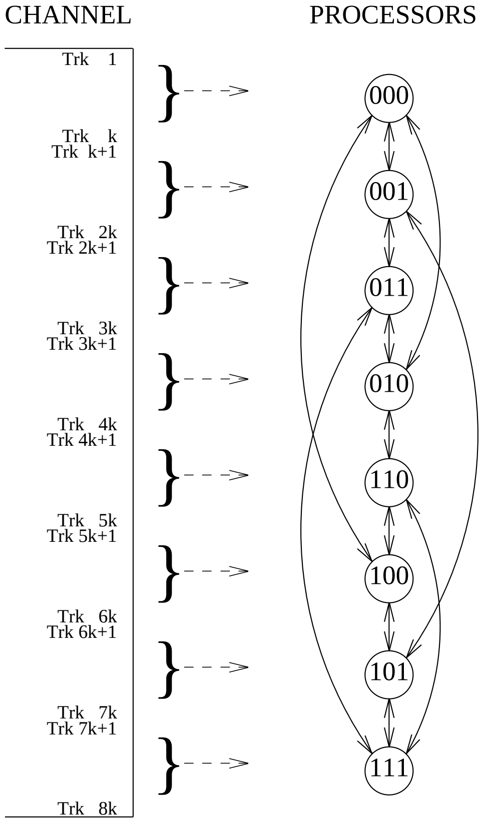


Figure 4.3. Domain Map for Three-Dimensional Hypercube

nels for the three-dimensional hypercube as it is embedded into a line. Adjacent processors in the array are chosen to be adjacent nodes of the hypercube, following a pseudo-gray code. The pattern is not a true gray code since we chose not to have the topmost and bottommost processors adjacent. This distributes the long range connections more evenly.

4.4.2. Data partitioning

After the initial placement of the subnets into tracks (similar to the serial implementation), sets of adjacent track are assigned to corresponding nodes in the linear array of Figure 4.3. The tracks are distributed as evenly as possible so that the work load of each node is as uniform as possible. The channel area is divided into strips of tracks because the algorithm used assumes that subnets are displaced or exchanges between different tracks.

Each node is given information about the horizontal space used by each subnet in each track assigned to it. In other words, each node receives $1/P^{th}$ of the corresponding serial linked list structure for the tracks. It is unnecessary for each node to know what sections of its neighbor's tracks are occupied or not. However, a copy of the entire column data array is maintained in each node because of faster accessing and the small amount of updating required for the column data.

4.4.3. Parallel moves

The moves used to transform one channel state into another will still be based on the displacement and exchange moves of the serial algorithm. In this case, however, two nodes cooperate together as a pair to perform the desired transformation. During the evaluation of a move, one processor of the pair acts like a master, and the other a slave. The following moves can then be identified:

MOVE 0: Intra-Displace -- each node of a pair performs a displacement move within its own sets of subnets and tracks

MOVE 1: Inter-Displace -- master node displaces a subnets from its domain to a track within the domain of the slave node.

MOVE 2: Intra-Exchange -- each node of a pair performs an exchange move within its own sets of subnets and tracks

MOVE 3: Inter-Exchange -- master and slave nodes each select a subnet to exchange with each other

By applying the Inter-processor moves, it is possible to utilize the connections to nodes not adjacent on the linear array to move a subnet a large distance up or down the channel in a single move.

It is important to select which node should be the master and which node should be the slave for a given iteration. The node numbers of the two nodes of a pair always differ in exactly one bit position. An algorithm specifying that the node with a one in the bit position should be the master and the other, the slave, would not work because then sooner or later, all of the slave's subnets would get displaced to the master. Instead, the mastership of a pair of nodes should alternate after each iteration.

The selection of the move is performed at the beginning of an iteration by the master processor. The ratio of intraprocessor to interprocessor moves is 1:1. Intraprocessor moves improve the performance and speedup, but interprocessor moves are equally necessary to be able to move the subnets throughout the channel. The ratio of displacement moves to exchange moves ranges between 15:1 and 20:1, the same as in the serial implementation.

Since the hypercube has no shared memory, it is necessary for the nodes of a pair to communicate through messages while evaluating each current move. Figure 4.4 illustrates the communication requirements for each of the four types of moves discussed earlier, and Figure 4.5 lists the steps performed by the master and slave processors in evaluating the move. Note that for MOVE 3, Inter-Exchange, it is possible to overlap the first message sent by the master with the first message sent by the slave to gain some parallelism. Furthermore, some calculations can be performed by each processor during the transmission of the first message.

4.4.4. Parallel updating

The data defining all aspects of a subnet are split up into two separate structures, the track linked lists and the column data array. If a move is evaluated favorably (using the same cost and acceptance evaluation functions as the serial implementation), the information in the data structures must be updated to reflect these changes. It should be noted, however, that with evaluating moves in parallel, the information is, for all purposes, out of date. Processor pair (i,j) evaluates their move(s), tacitly assuming the rest of the data on the channel is constant, while at the same time processor pair (k,l) is doing likewise. It is very possible that the moves may offset each other and result in the state of the channel being worse than expected.

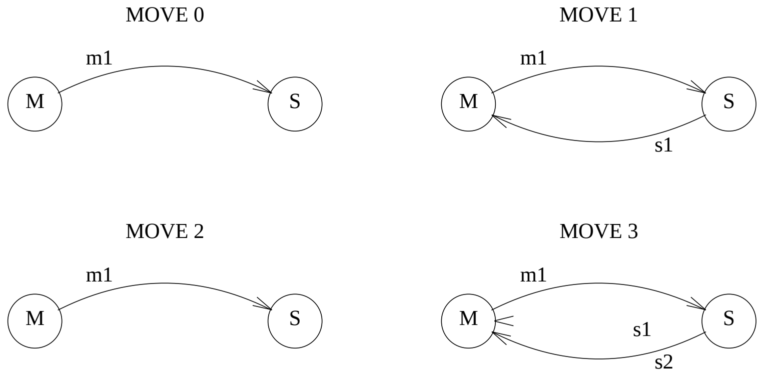


Figure 4.4. Move Communication Requirements

MASTER

```
Select Subnet mn1
Select Move
Calculate Cost Change for Removing mn1
Send Subnet mn1 Data to Slave (m1)
...
Case (MOVE)
   0:  Select New Track mt1
       Calculate Cost for Placing mn1 in mt1
       If (Accept(Cost Change))
           Update mn1
       End Case 0

   1:  Wait For Cost Data From Slave (s2)
       ...
       ...
       Receive Cost Data From Slave (s2)
       If (Accept(Cost Change))
           Update mn1
       End Case 1

   2:  Select Subnet mn2 for Exchange With mn1
       Calculate Cost for Placing mn1 in mt2
       Calculate Cost for Placing mn2 in mt1
       if (Accept(Cost Change))
           Update mn1 and mn2
       End Case 2

   3:  Wait for Subnet sn1 From Slave (s1)
       Receive Subnet sn1 Data From Slave (s1)
       Calculate Cost of Placing sn1 in mt1
       Wait for Cost Change Data From Slave (s2)
       Receive Cost Change Data From Slave (s2)
       If (Accept(Cost Change))
           Update mn1 and sn1
       End Case 3

   End Case(MOVE)
Broadcast Updates
```

SLAVE

```
Select Subnet sn1
Calculate Cost Change for Removing sn1
Wait for Subnet mn1 Data From Master (m1)
...
Receive Subnet mn1 Data From Master (m1)
Case (MOVE)
   0:  Select New Track st1
       Calculate Cost for Placing sn1 in st1
       If (Accept(Cost Change))
           Update sn1
       End Case 0

   1:  Select New Track st1
       Calculate Cost for Placing mn1 in st1
       Send Cost Change to Master (s2)
       End Case 1

   2:  Select Subnet sn2 for Exchange With sn1
       Calculate Cost for Placing sn1 in st2
       Calculate Cost for Placing sn2 in st1
       If (Accept(Cost Change))
           Update sn1 and sn2
       End Case 2

   3:  Send Subnet sn1 Data to Master (s1)
       Calculate Cost of Placing mn1 in st1
       Send Cost Change Data to Master (s2)
       End Case 3

   End Case(MOVE)
Broadcast Updates
```

Figure 4.5. Master/Slave Move Evaluation Steps

Jones and Banerjee [20] have found that the convergence properties were nearly maintained despite the use of parallel moves similar to what we propose. Furthermore, they were able to apply those results to a uniprocessor or serial implementation of their placement algorithm [36]. They applied what was termed *pseudo-parallel moves* in which a series of moves would be evaluated and accepted before performing an update on the data structure. By doing this, they were able to maintain the convergence of their algorithm while decreasing the computation time dramatically. It is possible to apply *pseudo-parallel moves* because of the non-deterministic behavior of annealing. An offshoot of that idea, shown by Grover [16], is to use approximate cost calculations to save computation time. As others have pointed out [22, 37], it is important to carefully control parallel or approximated moves, especially at low temperatures.

As shown in Figure 4.2, updating of all data structures is performed after each pair of processors has executed moves in parallel. Every node must receive the updated information, so some method must be used to broadcast the information across the network. The simplest method, provided the hypercube network could support the function, is to have every node broadcast the information to every other node. Unfortunately, the iPSC hypercube does not have appropriate hardware for global communication; instead, a global send is performed by sending a copy of the message out on a tree embedding [38]. Another possible method is to embed a ring network into the hypercube and transmit update information around the network until it returns to the originating node. The tree broadcast scheme is $O(log\ P)$, but contention and congestion on the channels will likely slow the performance down considerably. On the other hand, the ring network scheme is $0(P)$, but will not have problems with contention and congestion because of the uniform uni-directional flow of data around the network.

4.5. Heuristics

In general, nearly all of the heuristics discussed in the previous chapter for the serial implementation were also applied in the parallel implementation, modified slightly for the different moves and data structures. One problem faced by using the Intel Hypercube Simulator was the extremely long time needed for each trial run. It became important then to find additional heuristics to speed up the convergence of the algorithm.

The first idea was to improve the selection of the second subnet of an intraprocessor exchange move. Assuming that the first subnet i is already chosen, it is possible to evaluate the overlap that would occur for each subnet of the set of possible subnets to be chosen. This overlap can then be used to bias the selection of the subnet in favor of the subnets causing the least damage, and which will likely provide some improvement.

The next heuristic applied was to adjust the constant factor γ for the change in channel width of the cost function to reflect whether or not tracks should be removed, added, or neither. For example, if the channel density for a problem is D, and if along the annealing process the current channel width is $D + 2$, then it is favorable to increase the chances of removing a track and decrease the chances of adding a track. If, however, the current channel width is D, then it is usually better not to create new tracks or remove any of the current tracks.

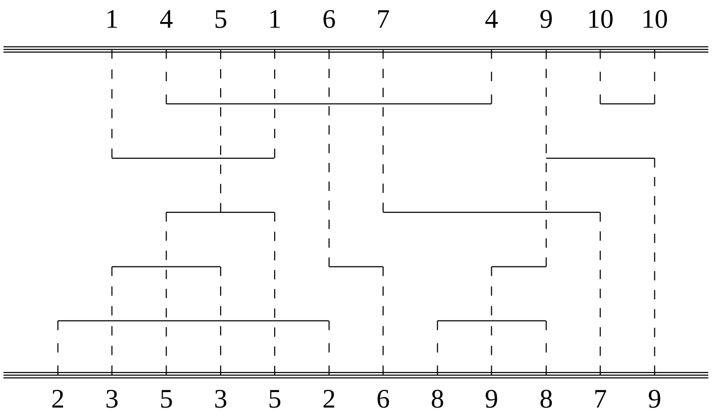
The final heuristic used is to approximate the vertical constraint graph G created from the original problem statement and estimate track positions based on the structure of the graph. Figure 4.6 and Table 4.1 show the solution to a simple channel routing problem, the vertical constraint graph for the problem, and the data resulting from approximating the vertical constraint graph. Source nodes of G are nodes which only have directed arcs pointed away from the node. Sink nodes are nodes which only have directed arcs pointed into the node. Assuming for now that there are no cycles, for each node of the graph one can find a path through that node which starts at a source node and ends at a sink node. Let ρ_i be the longest path from source to sink passing through node i. For node i the final track placement can be approximated by the position of the node along ρ_i. For channel width w, net number 7 of the example would be assigned to a zone ranging from track $0.25 \times w$ to track $0.50 \times w$.

This approximation can then be applied to the cost evaluation at several points. One possible use is for subnet selection. Subnets not placed in the tracks of their zones can be biased for selection higher than those inside their zone. Another way to apply this is in the selection of tracks for displacement. Tracks to be selected can be biased inversely proportional to the distance from the subnet's zone.

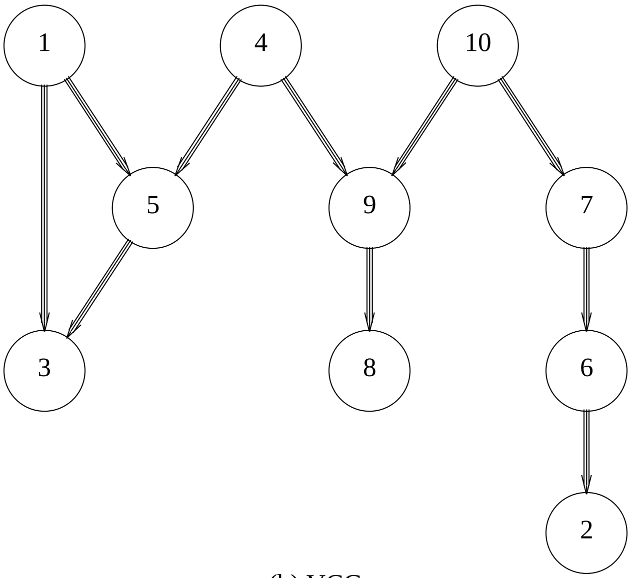
4.6. Algorithm Results

For reasons similar to those in Chapter 3, we will be presenting a summary of the results of one channel routing problem for the set of heuristics that arrived at the best results. The same channel routing problem with channel density twelve was used for the testing.

The ratio of displacement-to-exchange moves was 15:1. At the same time, the temperature was decreased according to the annealing schedule, and the number of iterations at that temperature was increased by 45% over



(a) Problem



(b) VCG

Figure 4.6. Vertical Constraint Graph Example

Table 4.1. Approximated VCG Data

Subnet	Position in Path	Total Path Length	Percentage of Total	Approximate Track
1	1	3	0.25	2
2	4	4	0.80	5
3	3	3	0.75	5
4	1	3	0.25	2
5a	2	3	0.50	3
5b	2	3	0.50	3
6	3	4	0.60	4
7	2	4	0.40	2
8	3	3	0.75	5
9a	2	3	0.50	3
9b	2	3	0.50	3
10	1	4	0.20	1

the number of iterations at the previous temperature. The cost factor γ was dynamically changed to reflect the need to add or remove tracks in the channel. Finally, the VCG was approximated and the information was used to bias the selection of subnets by weighting subnets not found in their expected track range with a higher probability. Furthermore, the approximated track position was used to bias the selection of tracks for displacement.

The final routing solution for the twelve-track example is shown in Figure 4.7. The plots of Cost vs. Temperature, Average Overlap vs. Temperature, and Average Number of Tracks vs. Temperature are found in Figures 4.8, 4.9, and 4.10, respectively.

4.7. Performance Analysis

Although the parallel implementation used the Intel Hypercube Simulator, we did perform an analysis of the expected speedup of the algorithm when run on the Intel iPSC Hypercube.

4.7.1. Computation costs

The amount of time spent on processing each new move was measured by applying the **CLOCK()** function of the simulator to random moves repeated thousands of times. Since the simulator was run on a Motorola 68020 CPU and the hypercube uses Intel 80286 processors, some adjustment must be made to account for the difference in processing speeds. The 68020 is rated at 2.7 MIPS, while the 80286 is rated at 0.78 MIPS. This is

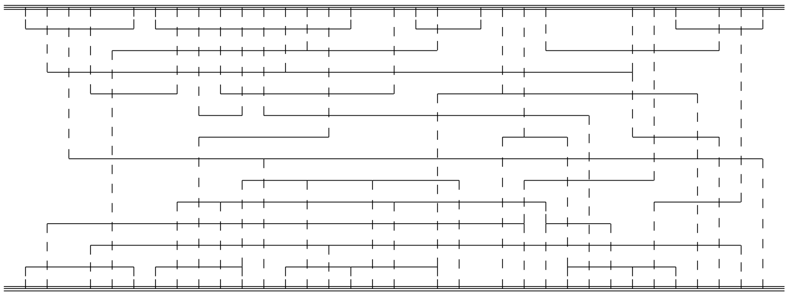


Figure 4.7. Final 12 Track Solution -- Parallel

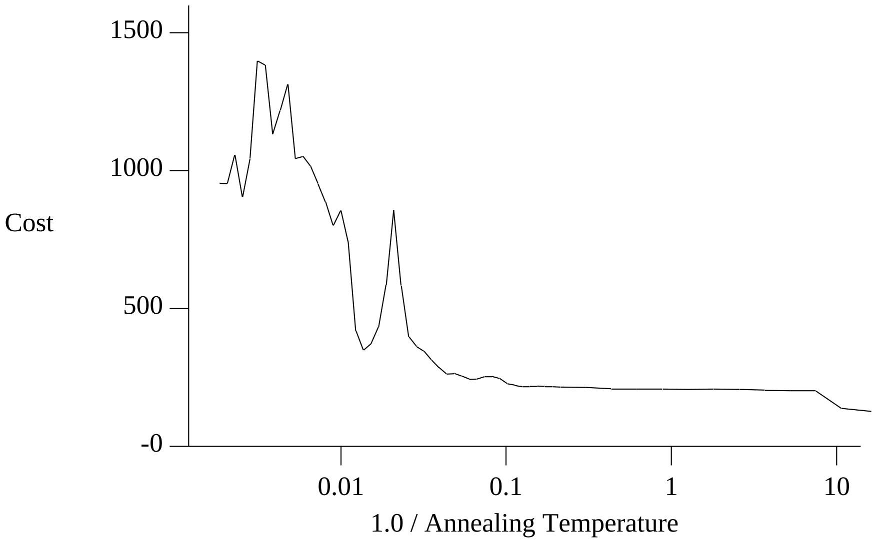


Figure 4.8. Annealing Cost vs. Temperature -- Parallel

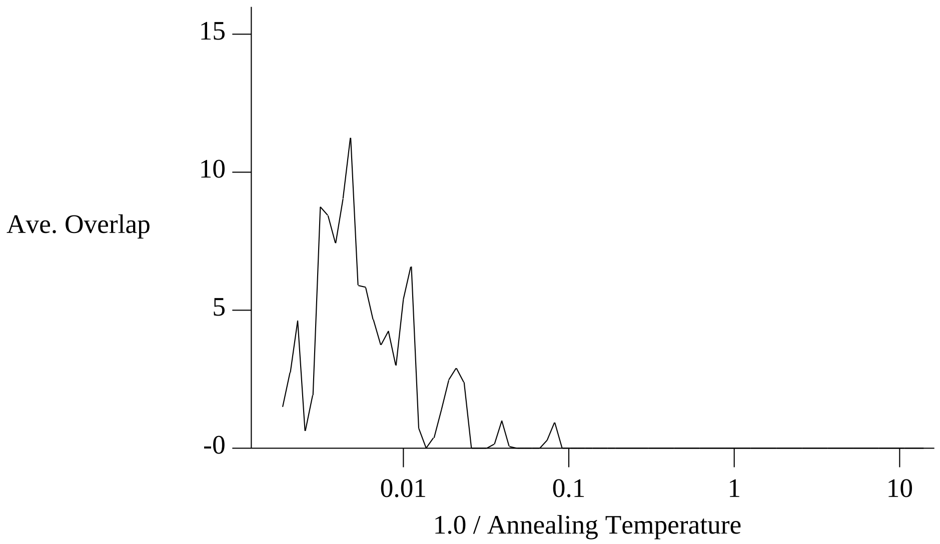


Figure 4.9. Subnet Overlap vs. Temperature -- Parallel

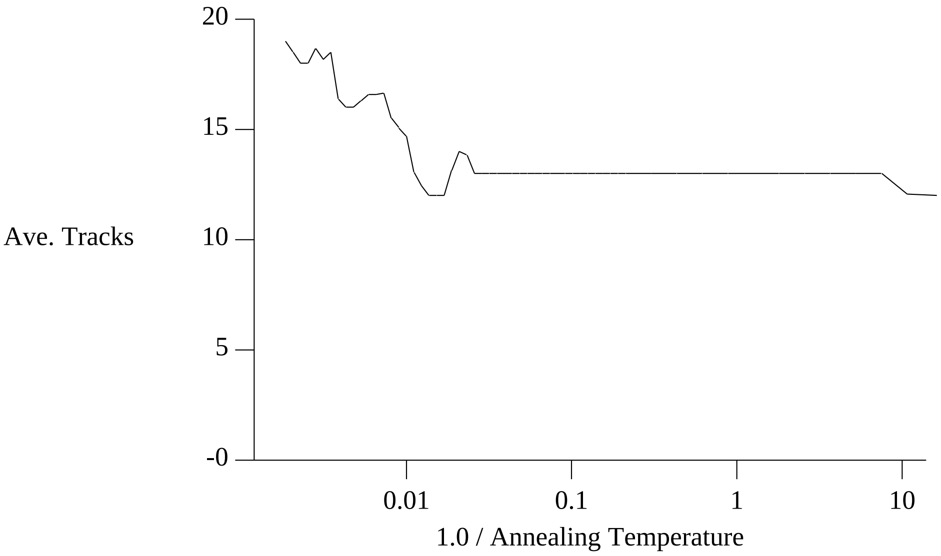


Figure 4.10. Average Number of Tracks vs. Temperature -- Parallel

a difference of approximately 3.5. Table 4.2 gives the computation times for the master and slave nodes for each of the four types of moves for both the 68020 and the 80286. The computation costs for updating subnets after each move are also given in Table 4.2.

Table 4.2. Computation Timing (msec)

Operation	MC68020 CPU (Measured)		80286 CPU (Projected)	
	Master	Slave	Master	Slave
MOVE 0	17.0	15.8	59.5	55.3
MOVE 1	8.4	6.3	29.4	22.0
MOVE 2	22.0	17.1	77.0	59.9
MOVE 3	5.8	3.5	33.3	25.9
Update	5.1	5.1	18.0	18.0

4.7.2. Communication costs

The simulator does not provide any mechanism for estimating the amount of time needed to send a message from one node to one of its neighbors, so we will use timing information reported in the literature for sending one-hop messages on the Intel iPSC Hypercube [39]. Table 4.3 summarizes the message timing for the different types of messages used in our implementation. The messages $m1$, $s1$, and $s2$ are from Figure 4.4. The update message is the packet transmitted around the broadcast ring for updating subnets after a move. There are two other types of messages used, one is for sending the original net data from the host to the nodes and the other is for sending the final routing data from the nodes back to the host. These, however, do not affect the speedup of the algorithm and are not discussed here.

4.7.3. Speedup calculations

Assuming a 16 node hypercube, the ratio of moves is 15:15:1:1, respectively, for MOVE 0, MOVE 1, MOVE 2, and MOVE 3, which means that during every iteration in which pairs of nodes are evaluating a move,

Table 4.3. Message Transmission Timing (msec)

	m1	s1	s2	update
Operation	48 bytes	48 bytes	16 bytes	48 bytes
MOVE 0	1.83	-	-	-
MOVE 1	1.83	-	1.74	-
MOVE 2	1.83	-	-	-
MOVE 3	1.83	1.83	1.74	-
Update	-	-	-	1.83

at least one pair will be performing either MOVE 0 or MOVE 1. MOVE 0 and Move 2 are bottleneck moves because of the computation required. The average time for the bottleneck moves then can be found by weighing each move by its probability of occurrence. The average move computation time per iteration is then 60.6 ms. Since there is an equal probalility of selecting interprocessor moves and intraprocessor moves, approximately half of the node pairs will evaluate a single move and the other half will evaluate parallel moves. Thus there are usually $0.75 \times P$ moves at once, or for the sixteen-node case, twelve parallel moves. Including the communication costs for messages $m1$, $s1$, and $s2$ gives a worst-case move time of 62.5 $msec$ for one iteration. Using a tree-based broadcast strategy, the communication time is $\log P \times 1.83\ msec$. The update computation time is 18.0 ms, giving a total time of

$$(4 \times 1.83) + 18.0 + 62.5 = 87.8\ msec$$

For the uniprocessor case, twelve moves would require

$$(11.25 \times 59.5) + (0.75 \times 77.0) + 14.0 \ = \ 745.6\ msec$$

to complete, resulting in an overall speedup of 8.4 on a 16 processor hypercube.

CHAPTER 5

CONCLUSIONS

5.1. Summary of Results

In this thesis, we have presented serial and parallel algorithms for channel routing using simulated annealing. Simulated annealing is a powerful optimization tool and we have demonstrated its use in a new uniprocessor channel routing algorithm. This algorithm permits maximal freedom to the nets in the channel being assigned to achieve near-optimal results.

The algorithm has been parallelized for implementation on a hypercube computer. The channel is partitioned horizontally by tracks, and adjacent nodes of the hypercube cooperate in parallel to gradually improve the state of the routing. The data have been partitioned to try to minimize the overhead of message passing between pairs and complete updates.

5.2. Convergence Issues

One important issue to consider carefully in the design of any algorithm for simulated annealing is how quickly the algorithm will converge. There is always a tradeoff between the total number of moves attempted and the time taken to evaluate each move. It is an intractable problem to analyze the problem enough to find one move that would solve the whole problem, and it would take extremely large numbers of moves to solve the problem without analyzing any of them. Between those extremes is the optimal point for minimizing total time to converge. To find that point it is necessary to perform many tests on various strategies and parameters for selecting and evaluating moves.

One of the main features of the algorithm presented here is the allowing of channel states at high temperatures that would be unacceptable as the final solution. These states usually include overlap between nonconnected wires. It has been shown that for certain annealing algorithms convergence is guaranteed, but since illegal intermediate states are allowed, there is no longer any guarantee of convergence. For this reason, it is essential to evaluate all aspects of the algorithm carefully.

5.3. Applicability of Simulated Annealing

Part of the issue of applicability of simulated annealing to channel routing involves the convergence question. If convergence is not achieved, nearly all of the time in a reasonable amount of time, then the problem, by nature, may not be well suited for simulated annealing. For the algorithm presented, good convergence was achieved for small cases, especially for the serial version. However, for large examples, the quality of the results dropped off. This may be due in part to improper selection and evaluation heuristics.

Another aspect concerns the nature of the problem itself and how the current model affects it. For simulated annealing, the choice of neighboring spaces is very important. The algorithm of Leong, Wong, and Liu [11] only allowed legitimate solutions to be in the neighboring space of a current channel state. This greatly reduces the number of possible moves.

The algorithm we propose here allows any possible permutation of the current channel state to be in its neighboring space. It is then much harder to determine the best state to select next, so much more computation is needed. There is another tradeoff here between the benefits of the new algorithm's flexibility and the added work to determine the next state. This is an important area of future research.

5.4. Parallelizability of the Channel Routing Algorithm

How to write parallel algorithms has been a lively topic over the past decade. There are many ways to look at the parallelization of the serial simulated annealing algorithm for channel routing, and the method presented in this thesis is the way we determined to be the best suited for the hypercube facilities available. Our approach can be looked at as a parallelization of individual moves, or multiple moves at once. Another approach is to parallelize the computation of a single move, hopefully providing a high enough computation / communication ratio to be effective. Sets of moves can be parallelized in which each node performs serial annealing on the entire channel for a fixed number of moves, then all nodes combine and take the best of the results. This method may be an effective alternative, but it seems to sidestep parallel algorithm designing.

For the algorithm presented, there are a few issues of concern. First, the cost of updating each node after every set of parallel moves is high because of the overhead. Research into partial or delayed broadcasting is necessary. Second, the overall parallelism is limited. Most channels for routing in industry consist of fewer than 100 tracks. It is impractical to distribute those tracks to more than 32 hypercube nodes. The theoretical speedup is limited to 32, assuming linear speedup, which is unlikely. Other ways to partition the problem to allow for higher ranges of speedup should also be looked into.

5.5. Future Research

Throughout this thesis work, a good, solid base algorithm for simulated annealing has been developed. The current results indicate a need for more research into the areas mentioned above. Furthermore, other applications of this algorithm should be studied.

REFERENCES

[1] T. C. Hu and E. S. Kuh, *VLSI Circuit Layout: Theory and Design*. New York, NY: IEEE, Inc., 1985. pp. 3-18.

[2] C. Y. Lee, ‘‘An algorithm for path connection and its applications,’’ *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346-365, 1961.

[3] A. Hashimoto and J. Stevens, ‘‘Wire Routing by Optimizing Channel Assignment,’’ *Proc. 8th Design Automat. Conf.*, pp. 214-224, June 1971.

[4] D. Deutsch, ‘‘A Dogleg Channel Router,’’ *Proc. 13th Design Automat. Conf.*, pp. 425-433, June 1976.

[5] T. Yoshimura and E. S. Kuh, ‘‘Efficient algorithms for channel routing,’’ *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 25-25, Jan. 1982.

[6] R. L. Rivest and C. M. Fiduccia, ‘‘A Greedy Channel Router,’’ *Proc. 19th Design Automat. Conf.*, pp. 418-424, June 1982.

[7] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, ‘‘A new symbolic channel router: YACR2,’’ *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 208-219, July, 1985.

[8] M. Burstein and R. Pelavin, ‘‘Hierarchical Channel Router,’’ *Proc. 20th Design Automat. Conf.*, pp. 591-597, June 1983.

[9] R. Joobbani and D. P. Siewiorek, ‘‘WEAVER: A Knowledge-Based Routing Expert,’’ *IEEE Design & Test*, vol. 3, no. 1, pp. 12-23, Feb. 1986.

[10] H. Shin and A. Sangiovanni-Vincentelli, ‘‘Mighty: A ’Rip-Up and Reroute’ Detailed Router,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 2-5, Nov. 1986.

[11] H. W. Leong, D. F. Wong, and C. L. Liu, ‘‘A Simulated Annealing Channel Router,’’ *Proc. 22nd Design Automat. Conf.*, pp. 226-228, June 1985.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, ‘‘Optimization by simulated annealing,’’ *Science*, vol. 220, pp. 671-680, May 1983.

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, ‘‘Equations of State Calculations by Fast Computing Machines,’’ *J. Chem. Phys.*, vol. 21, pp. 1087-1092, 1953.

[14] J. Lam and J. M. Delosme, ‘‘Logic Minimization Using Simulated Annealing,’’ *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, pp. 348-351, Nov. 1986.

[15] C. Sechen and A. Sangiovanni-Vincentelli, ‘‘TimberWolf3.2: A New Standard Cell Placement and Global Routing Package,’’ *Proc. 23rd Design Automat. Conf.*, pp. 432-439, June 1986.

[16] L. K. Grover, ‘‘A New Simulated Annealing Algorithm for Standard Cell Placement,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 378-380, Nov. 1986.

[17] M. P. Vecchi and S. Kirkpatrick, ‘‘Global wiring by simulated annealing,’’ *IEEE Trans. Comput.*, vol. 7, no. 4, pp. 215-222, Oct. 1983.

[18] M. J. Chung and K. K. Rao, ‘‘Parallel Simulated Annealing for Partitioning and Routing,’’ *Proc. Int. Conf. Computer Design*, pp. 238-242, Oct. 1986.

[19] S. A. Kravitz and R. A. Rutenbar, ‘‘Multiprocesssor-Based Placement by Simulated Annealing,’’ *Proc. 23rd Design Automat. Conf.*, pp. 567-573, June 1986.

[20] M. Jones and P. Banerjee, ‘‘Performance of a parallel algorithm for standard cell placement on the Intel Hypercube,’’ *Proc. 24th Design Automat. Conf.*, pp. 807-813, June 1987.

[21] F. Darema and G. F. Pfister, ‘‘Multipurpose Parallelism for VLSI CAD on the RP3,’’ *IEEE Design Test*, vol. 4, no. 5, pp. 19-27, October 1987.

[22] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, ‘‘A parallel simulated annealing algorithm for the placement of macro-cells,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 30-33, Nov. 1986.

[23] R. Jayaraman and R. A. Rutenbar, ''Floorplanning by annealing on a hypercube multiprocessor,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 346-349, Nov. 1987.

[24] H. W. Leong, *Routing problems in the physical design of integrated circuits*. Ph.D. dissertation, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, January 1986.

[25] E. H. L. Aarts and P. J. M. van Laarhoven, ''A New Polynomial-Time Cooling Schedule,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 206-208, Nov. 1985.

[26] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, ''An Efficient General Cooling Schedule for Simulated Annealing,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 381-384, Nov. 1986.

[27] R. M. Kling and P. Banerjee, ''ESP: A new Standard Cell Placement Package Using Simulated Evolution,'' *Proc. 24th Design Automat. Conf.*, pp. 60-66, June 1987.

[28] L. K. Grover, ''Standard cell placement using simulated sintering,'' *Proc. 24th Design Automat. Conf.*, pp. 56-59, June 1987.

[29] *Hypercube Simulator -- Internal Product Description, Version 3.0*. Intel Corporation, Oct. 1986. pp. 1-15.

[30] J. P. Hayes, T. N. Mudge, Q. F. Stout, S. Colley, and J. Palmer, ''Architecture of a Hypercube Supercomputer,'' *Proc. Int. Conf. Parallel Process.*, pp. 653-660, Aug. 1986.

[31] *iPSC Simulator Manual*. Intel Corporation, Oct. 1986.

[32] *iPSC System Overview*. Intel Corporation, Nov. 1986.

[33] *iPSC/2 Programmer's Reference Guide*. Intel Scientific Computers, Dec. 1987.

[34] M. H. Jones, *A parallel simulated annealing algorithm for standard cell placement on a hypercube computer*. M.S. thesis, Dept. of Electrical and Computer Engineering, Univ. of Illinois at Urbana-Champaign, January, 1987.

[35] L. M. Ni, C. King, and P. Prins, ''Parallel Algorithm Design Considerations for Hypercube Multiprocessors,'' *Proc. Int. Conf. Parallel Process.*, pp. 717-720, Aug. 1987.

[36] M. Jones and P. Banerjee, ''An improved simulated annealing algorithm for standard cell placement,'' *Proc. Int. Conf. Computer Design*, Oct. 1987.

[37] R. A. Rutenbar and S. A. Kravitz, ''Layout by Annealing in a Parallel Environment,'' *Proc. Int. Conf. Computer Design*, pp. 434-437, Oct. 1986.

[38] *iPSC Software Internal Specification*. Intel Corporation, Mar. 1987.

[39] T. H. Dunigan, ''Hypercube Performance,'' *Proc. SIAM 2nd Conf. Hypercube Multiprocessors*, pp. 178-192, 1986.

ABSTRACT

Two algorithms for channel routing using simulated annealing are presented. Many of the channel routers of the past are for the most part based on greedy algorithms in which special heuristics are applied to generate monotonic improvement. These algorithms are called greedy because they suffer from inappropriate selections, getting stuck at suboptimal solutions. Simulated annealing is an optimization methodology which allows the solution process to back up out of local minima that may be encountered by inappropriate selections. By properly controlling the annealing process, it is very likely that the optimal solution to an NP-complete problem such as channel routing may be found. Previous simulated annealing channel routers only permitted transformations which resulted in a routing without overlapping between nonconnected wires. The algorithm presented here proposes very relaxed restrictions on the types of allowable transformations, including overlapping nets. By freeing that restriction and controlling overlap situations with an appropriate cost function, the algorithm becomes very flexible and can be applied to many extensions of channel routing. The selection of the transformation utilizes a number of heuristics, still retaining the pseudorandom nature of simulated annealing.

The algorithm has been implemented as a serial program designed for a workstation, and a parallel program designed for a hypercube computer. The details of the serial implementation are presented, including many of the heuristics used and some of the resulting solutions. A description of the Intel iPSC Hypercube is given, details on how the channel routing problem was partitioned onto the hypercube are discussed, and results for an example and some performance calculations are presented. Finally, some concluding remarks are made concerning the applicability of simulated annealing to the channel routing problem, and some possibilities for future research work are discussed.

ACKNOWLEDGEMENTS

I wish to especially thank Professor Prith Banerjee for his continual encouragement, ideas, and support throughout the development of this work. I would also like to thank my family and my fiancee for their love, encouragement, and support. Finally, I would like to thank all of the members of the Computer Systems Group, past and present, for plenty of fruitful ideas as well as needed distractions.

This work was supported by the National Aeronautics and Space Administration under contract number NAG-1-613.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE