A Parallel Shortest Path Algorithm Based on Relaxed Heap for VLSI Wiring

Xiaoyang Lian
Department of Computer Science and Engineering
Harbin Institute of Technology
150001 Harbin, China
Lianxiaoyang.happy@163.com

Zhenzhou Ji
Department of Computer Science and Engineering
Harbin Institute of Technology
150001 Harbin, China
jzz@pact518.hit.edu.cn

Xiong Su
Department of Computer Science and Engineering
Harbin Institute of Technology
150001 Harbin, China
hit_suxiong@163.com

***Abstract*—Buffer insertion is an efficient technique in interconnect optimization. By inserting numbers of buffers, a wire can be turned into several nodes, so the total delay can be reduced. Buffer insertion has become a hot area of research in recent years. By constructing the optimal buffer insertion model, we can change the problem of buffer insertion into a shortest path problem in a directed graph. In this paper, we will introduce buffer insertion technique under one-dimensional line model and make a detailed analysis of a graph division algorithm under this model. This paper also presents a parallel shortest algorithm based on relaxed heap to solve the SSP problem of the divided graph. Experiments show that our shortest path algorithm can greatly reduce the time of finding optimal buffer insertion position, and with the increase of the number of buffers the speedup becomes more obvious.**

Keywords-relaxed heap; VLSI; buffer insertion; parallel shortest path; graph division

I. INTRODUCTION

As the VLSI design develops into ULSI stage, the attachment of the parasitic effect can't be ignored. The proportion of wiring delay is always increasing and it has replaced the gate delay as the main factor of the total delay. Therefore interconnect optimization technique becomes the key step of high-performance IC design. In nano-scale design, wiring delay has become the main delay of total delay. The delay is proportional to the square of the line length and it will be reduced after inserting buffers which make a line into several nodes. An algorithm has been proposed by Liang Deng according to the basic ideas of buffer insertion[1]. This algorithm changes the basic model into a directed graph by node splitting according to mean and variance of each node to the source node. Then the problem becomes the shortest path problem of the directed graph. But the directed graph is so large that there should be an efficient algorithm to solve the SSP problem. Relaxed heap is a kind of priority queue structure designed by Driscoll, Gabow, Shrairman and Tarjan. It is different from Fibonacci heap[2]. There are two kinds of relaxed heaps, the first one achieves the same amortized bounds as Fibonacci heaps and the second one achieves the same bounds in the worst case rather than amortized case. Relaxed heap is more suitable for parallel algorithm.

This paper presents a parallel shortest path algorithm based on relaxed heap and sequential shortest path algorithm. Based on the theory of buffer insertion and graph division algorithm, the position problem of buffer insertion can be quickly solved.

II. BASIC THEORIES

A. Buffer Insertion Technique

Buffer insertion technique in the interconnection lines is to make the respective insertion position equivalent to a node and then the optimal number of buffer insertion and location in the interconnection lines are determined according to a certain cost function. According to the current research, in the technology below 65nm, the delay parameters of interconnection lines are not decreased by equal proportion with the size of the device[1]. Therefore when the delay is considered, it is a random variable according to the probability distribution. So under the model of buffer insertion, the weight of graph edge will not be a fixed value and a conversion algorithm will be needed. This paper only studies the delay of one-dimensional line model.

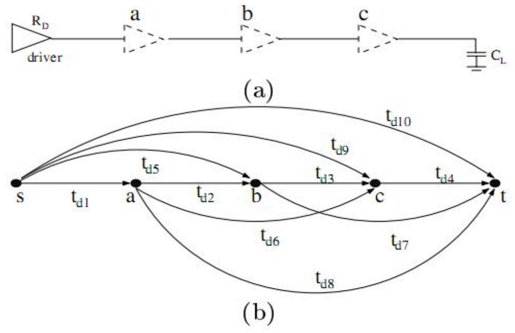

Figure 1. Buffer Insertion Model

Considering the directed graph G, Figure 1, where buffers have been equivalent to nodes, we need to find a path P from the source node s to the end node t, L_P is its length, μ_P and σ_P^2 are its mean and variance, the cost function is $\mu_P+k\sigma_P$. This path allows the minimum cost function. For an arbitrary edge e of G, assume Xe its distribution function, μ_e and σ_e^2 are its mean and variance. The distributions of all the edges are assumed to be independent of each other. We know that the two independent random variables X and Y, if $Z = X + Y$, then the mean and variance of Z can be obtained by adding the mean and variance of the two variables. Considering a path P in G, by adding the mean and variance, we can obtain:

$$\mu_P = \sum \mu_i \;, \quad \sigma_P^2 = \sum \sigma_i^2$$

If the cost function is $\mu_P + k\sigma_P^2$, the problem can be solved by judging the sum of cost function of each edge. But the cost function is $\mu_P + k\sigma_P$ and the standard deviation does not have a feature of linear sum, a new approach will be required. A lemma will be introduced first:

If the variance of any path from the source node to the end node is less than or equal to B, then the variance of any sub-path of G is smaller than B[3].

Now we can use the node splitting technique to solve the earlier problems. Assuming that the variance of each edge is an integer (if not, a nearby integer can be used), then we will use the algorithm proposed in this paper to solve the shortest path problem. According to the lemma, we obtain that the variance of each s-u path are integers between 1-B. The node u_i of G'(after splitting) represents the end node whose variance is i of the path s-u. For u_i, we call i the variance index. By the lemma, we only need to consider the path whose variance is less than B in graph G. Therefore, in graph G, except the source node s, the remaining nodes simply need to split into B nodes after converting G to G'. We can reduce the total number of nodes by merging the nodes with the same variance in the same topology layer. Since we assume that all nodes in G are uniformly distributed, then, the mean and variance of two nodes in the G' depend on the number of layers between the two nodes. Then the number of nodes created in each layer is

$$P_k(i) = \sum_{r=1}^{k} P_r(n-k) \;,$$

the number of total nodes is

$$P(i) = \sum_{k=1}^{i} P_k(i) \;.$$

The entire structure of graph transform algorithm is shown in Figure 2, the source node only needs a point s0 in G' to represent, nodes in the remaining layers should be split and finally you need to create an additional node to represent the endpoint.

```
Create 0th and 1th layer
for  2th to (n-1)th layer
     foreach  $\mu_i$ in G'
          if  $i+\sigma_{uv}^2$ don't exit
               Create new node $v_j$,  variance $j= i+\sigma_{uv}^2$
               Create arc $(\mu_i,v_i)$,  arc weight $\mu_{uv}$
          else $v_i$ already exit, Create arc $(\mu_i,v_i)$,  arc
weight $\mu_{uv}$
Create end node t',  set it the nth layer
foreach node $t_i$ in the (n-1)th layer
     Create arc (t_i,t'), weight  $\Phi(i)$
```

Figure 2. Graph Division Algorithm

B. Relaxed Heap

Relaxed heap is a kind of priority queue data structure and there are two kinds of relaxed heaps, rank relaxed heap and run relaxed heap[2]. Both of them consist of relaxed trees which are based on binomial trees and more structured than the Fibonacci trees[4]. Relaxed tree divides nodes in the trees into two groups: good nodes and bad nodes. Root node and nodes whose parent's key is bigger than its own are all good nodes and other nodes are bad nodes. This is called relaxed character of the relaxed heap. Some nodes in relaxed heap are defined as active and all of the bad nodes are active. In the rank relaxed heap, there is no more than one active node in each layer and each active node must be the last child. Different from the rank relaxed heap, run relaxed heap allow the operation of the bad nodes. There are two main differences between relaxed heap and binomial heap: one is the structure of the tree which is detailed in the preceding text, the other is the core operation of relaxed heap, decrease_key(x,v) which decreases the key of node x to v. The decrease_key operation tries to keep the relaxed character of the heap by rearranging the nodes. After updating the key of special node to smaller one, it will adjust the structure of the heap to keep the number of the active nodes unchanged or make it smaller and then stop or make another adjustment[2]. After a series of adjustments, the relaxed character of relaxed heap will be kept. Figure 3 shows the specific algorithm of relaxed heap.

```
Function decrease_key(x,v)
Begin key(x) = v; promote(x) end
Function promote(x)
Begin
        r = rank(x), p = parent(x), s = sibling(x,r +
1), g = grandparent(x)
        if  key(x) < key(p) then
                if x is the last child of p then
                        if active(r) == NULL then
active(r) = x
                        else if active(r) != x then
pair_transform end
                else if active(r+1) == s then
active_sibling_transform
                        else
good_sibling_transform
        end
end
Function pair_transform
…
Function active_sibling_transform
…
Function good_sibling_transform
```

Figure 3. Relaxed Heap Algorithms

The relaxed heap can achieve lower time complexity by keeping its relaxed character. The amortize time complexity of the following operations of rank relaxed heap, MAKE-HEAP, INSERT, MINIMUM, UNION and DECREASE-KEY, are O(1) and other operations' amortize time complexity is O(lg n). But run relaxed heap achieves the same time bounds in the worst case.

C. *Shortest Path Algorithm*

Considering directed graph G(V,E) where V is the set of vertexes and E is the set of edges. We assume s is the source node, d[v] is the distance from v to the source node s, c(u,v) is the weight of node u and node v and Q is the minimum priority queue. The serial shortest path algorithm of directed graph is shown in Figure 4.

```
( a ) Initial
    foreach v in V do
        d[v] = MAX
    end
        d[s] = 0
        Creat(Q,V)
        while IsEmpty(Q) != NULL do
         ( b ) ExtractMin(Q)
                u in Q and d[u] = min{d[v], v in Q}
        ( c ) Relax
                foreach (u,v) in E do
                    if d[u] + c(u,v) < d[v] then
                        d[v] = d[u] + c(u,v)
                            Update(Q,v)
                    endif
                end
end
```

Figure 4. Serial Shortest Path Algorithm

The parallel shortest path algorithm can optimize the above a, b and c parts. The idea is that, first, we divide nodes of directed graph into different groups. In order to balance the loads, each processor node should share almost the same number of nodes. Each processor node maintains a minimum priority queue which is related to the set of vertexes it cares for and all of the minimum priority queues are based on the relaxed heap. At the beginning, all processor nodes do the initial operation. Then each processor does find-min on its priority queue. Then the smallest of all the priority queue minima is found by doing a parallel minimum computation. Then we mark the node with the minimum value and the node number and the minimum value are broadcast to all processor nodes. The processor node containing the minimum value will do decrease operation on its priority queue. Each processor node will do relaxation operation. The above steps will be repeated to the end. The main framework is shown in Figure 5.

We now analyze the time complexity. Supposed p is the number of processors, n is the number of vertexes, and m is the number of edges. First is initial operation, the time of computing all the value of array d is O(n) in the worst case which means s connects to all vertexes and the time of creating relaxed heaps is O(p log (n/p)). Second, we need two steps to analyze the time complexity of the main loop operation. First, we consider the operations not involving the priority queues. The processor communication time for n minimum computations and n broadcasts is O(n log p). Take the relaxation operation into consideration, all edges must be traversed. So the time complexity is O(m/p) on average when each processor gets almost the same number of edges. So the total time complexity is O(n log p + m/p). Now consider the time for priority queue operations, we need n executions of find_min, delete and decrease. All of these operations use time O(n log n + n log (n/p)). Considering the non-priority queues operations, the parallel shortest path algorithm need total time O(m/p + n log n).

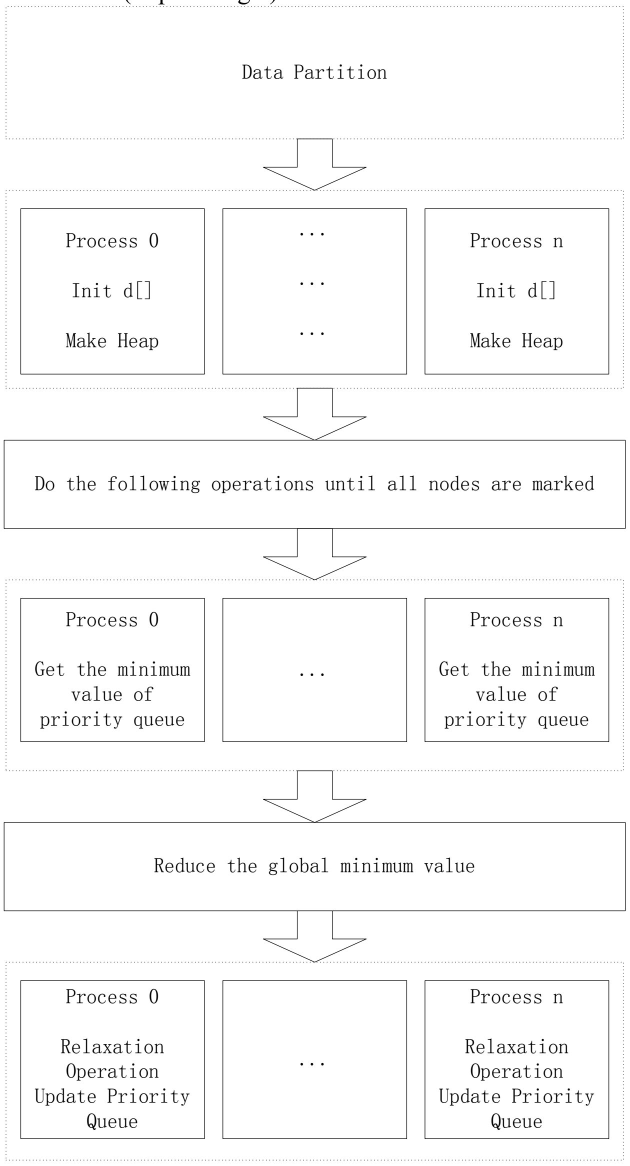

Figure 5. Framework of Parallel Shortest Path Algorithm

III. RESULTS

We need two steps to do the experiments. First we create an original directed graph under the one-dimensional line model, then create a new directed graph using the graph division algorithm and store the new graph into file. These steps don't involve parallel computing. Second we use our parallel shortest path algorithm to get the shortest path of graph which is stored in file. The parallel environment is based on MPI under Intel(R) Core(TM)2 Quad CPU Q8300 @ 2.5GHz with 2G storage. The results are shown in Table 1.

TABLE I. RESULTS

Buffer Num	After Graph Division		Serial Time(s)	Parallel Time(s)	Speed-up
	Vertex Num	Edge Num			
50	9324	95814	0.055	0.04	1.375
100	146675	2843186	1.21	0.63	1.92
110	217141	4611721	2.02	0.94	2.14
120	310943	7186600	3.14	1.48	2.18
130	432960	10822831	4.89	2.16	2.26
140	588458	15825762	6.68	2.82	2.36

The results show that our shortest path algorithm is very stable under the 4-core CPU and with the increase of the number of buffers the speedup becomes more obvious.

IV. CONCLUSION

In this paper, we showed a buffer insertion technique under one-dimensional line model and made a detailed analysis of the graph division. We also gave a new data structure called relaxed heap which has a very low time bounds and came up with a parallel shortest path algorithm based on relaxed heap. We made experimental simulation to test our algorithm. The results show our algorithm is very efficient especially under great number of buffers, so this algorithm has a wide range of applications. Our parallel algorithm is based on the serial Dijkstra algorithm, in fact, there are many good parallel algorithms which are not based on Dijkstra. So how to improve the efficiency of parallel algorithm needs to be further discussed [5].

REFERENCES

[1] Liang Deng, Martin D. F. Wong: Buffer Insertion Under Process Variations for Delay Minimization

[2] JAMES R. DRISCOLL, HAROLD N. GABOW, RUTH SHRAIRMAN, and ROBERT E. TARJAN: Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation

[3] Liang Deng, Martin D. F. Wong: An Exact Algorithm for the Statistical Shortest Path Problem

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms

[5] Ulrich Meyer, Peter Sanders: Delta-Stepping: A Parallel Single Source Shortest Path Algorithm