A Bus-Aware Global Router

Pin-Xin Liao and Ting-Chi Wang
Department of Computer Science
National Tsing Hua University
Hsinchu 30013, Taiwan
a25365473@gmail.com and tcwang@cs.nthu.edu.tw

Abstract—In this paper, we present a bus-aware global router that handles the length-matching issue of buses by modifying a well-known global router, NTHU-Route 2.0, with the following enhancements: (1) a new net ordering determination method for rip-up and reroute, and (2) a length-bounded hybrid unilateral monotonic routing method. The experimental results show that our router can successfully solve 9 of 11 test cases without causing any overflow. In particular, for one of the 9 test cases, NTHU-Route 2.0 cannot completely eliminate its total overflow. In addition, our global router can produce a high quality solution in terms of total bus wirelength deviation, while maintaining comparable total wirelength and runtime efficiency.

I. INTRODUCTION

As the feature size shrinks, the interconnects of a chip become more complex. Due to the high complexity, the routing problem is typically divided into two parts: global routing and detailed routing. In the stage of global routing, the wires of nets are mapped to a coarse-grained routing graph. On the other hand, in the stage of detailed routing, a detailed router assigns wires into the exact routing tracks in an uncoarse routing graph by inheriting the routing result from global routing. Consequently, global routing has a great effect on the solution quality of detailed routing.

International Symposium on Physical Design (ISPD) held two global routing contests, and also released two sets of benchmarks in 2007 [1] and 2008 [2] to boost the research and development of global routing techniques. Research groups developed dozens of improved academic global routers during the contests or afterward, with three typical objectives to minimize: (1) the total overflow, (2) the total wirelength, and (3) the computation time. These global routing methods can be categorized into two classes. The first class is three dimensional (3D) routing, while the other class is two dimensional (2D) routing followed by layer assignment. FGR [3], GRIP [4], and, MGR [5] are members of the first class. FGR uses sequential 3D maze routing to find solutions for nets on a 3D routing graph, while GRIP adopts integer linear programming (ILP) to formulate the routing problem. Both FGR and GRIP perform well in minimizing the total overlap and the total wirelength; however, their computation time is much longer. MGR applies a multi-level framework to reroute nets on the 3D grid graph. Nevertheless, some routing information may be hidden in the coarsening stage of MGR, thus degrading routing solution quality. Due to the complexity of modern designs, normally 3D routing takes longer time than the 2D routing followed by layer assignment. Other global routers, such as Archer [6], BoxRouter 2.0 [7], FastRoute 4.0 [8], Maize-Router [9], NCTU-gr2 [10], NTUgr [11], and NTHU-Route 2.0 [12], are members of the second class. They use two stages to generate 3D routing solutions. In the first stage, the original 3D routing space is projected onto the 2D plane, and most of routers such as [6], [8], [9], [10], [11], [12] apply 2D global routing based on a different rip-up and reroute strategy to obtain a 2D routing solution. In the second stage, they all map the 2D solution from the projected plane to the original 3D routing graph by using layer assignment.

In modern ICs, buses are important signals and are widely used in computation-intensive and/or storage-intensive designs. The growing trend of Intellectual Property (IP) integration also increases the amount of buses in a design. A bus transfers signals in parallel from one functional unit to others. For a bus carrying \( n \) signals, it has one source and one or more sinks, where the source and each sink all have \( n \) pins (see Fig. 1). To match the timing among the pins of a sink, the wirelength between each pin in the sink and the corresponding pin in the source is preferred to be as similar as possible. Therefore, if the interconnects of a bus are considered individually as a regular net during the routing stage, it will counteract the reasons for using the bus.

![Bus of n nets](Image)

Fig. 1: The relation between a bus and its nets.

All global routers that we introduced above do not consider the length-matching issue of buses. The bus-aware global routing problem is different from the traditional global routing problem that aims to minimize wirelength without causing overflow. The bus-aware global routing problem additionally needs to deal with the length-matching issue due to buses. A few bus routing algorithms have been studied by various researchers. The work in [13] proposes a bus centric global routing algorithm to identify and re-use the common routing topology of the different signals in a bus. The idea of common routing topology is later adopted by the algorithms in [14] and [15] that also consider bus pin flip-
ping and bus orientation to enlarge solution space. The methods in [14] and [15] route each bus by using a “virtual net”, which is a single net that represents all the nets of a bus and consumes more routing resources (depending on the number of nets in the bus). However, if a virtual net routing fails, they route those nets by using the same strategy as for regular nets. In addition, their test cases are two-layer designs with a small number of nets. Therefore, their bus routing methods may not generate a good routing result for buses in a modern VLSI design.

In this paper, we present a bus-aware global router that handles the length-matching issue of buses by modifying a well-known global router, NTHU-Route 2.0 [12], with the following enhancements: (1) a new net ordering determination method for rip-up and reroute, and (2) a length-bounded hybrid unilateral monotonic routing method. The experimental results show that our router can successfully solve 9 of 11 test cases without causing any overflow. In particular, for one of the 9 test cases, NTHU-Route 2.0 cannot completely eliminate its total overflow. Besides, our global router can produce a high quality solution in terms of total bus wirelength deviation, while maintaining comparable total wirelength and runtime efficiency.

The rest of this paper is organized as follows. In Section II, we give the problem formulation of bus-aware global routing, and briefly review NTHU-Route 2.0 [12] as well as hybrid unilateral monotonic routing [16]. In Section III, we describe our bus-aware global router in detail. In Section IV, we report the experimental results. We conclude this paper in Section V.

II. PRELIMINARIES

In this section, we first describe our global routing problem. After that, we review NTHU-Route 2.0 [12] and the hybrid unilateral monotonic routing method [16], since our router is an extension of NTHU-Route 2.0 and it takes buses into consideration by using the technique of hybrid unilateral monotonic routing.

A. Problem Formulation

Generally, a global routing problem can be modelled as a grid graph $G(V, E)$ shown in Fig. 2(a) and 2(b). The graph in Fig. 2(a) is a 2D routing model that contains a set $V$ of global tiles and a set $E$ of global boundaries. Fig. 2(b) shows a grid graph for a three-layer design example, and the vias are used in the multi-layer design to connect among layers. The global routing problem is to find a tree for each net such that the tree connects each pin in the sink and the corresponding pin in the source is preferred to be as similar as possible in a global routing result. Among these wirelengths of a sink, the length difference between the largest wirelength and each remaining wirelength can be calculated and summed up to get the wirelength deviation for the sink. Similarly, for each bus, by adding up the wirelength deviation of each sink, we get the wirelength deviation for the bus. The total wirelength deviation of all buses (i.e., the total bus wirelength deviation) is therefore defined to be the sum of the wirelength deviation of each bus.

The main objective of our bus-aware global routing problem is to minimize the total overflow and the total bus wirelength deviation, and its second objective is to minimize the total wirelength and runtime.

The flowchart of NTHU-Route 2.0 [12] is shown in Fig. 3. NTHU-Route 2.0 has four routing stages: initial stage, main stage, refinement stage and layer assignment stage.

B. Review of NTHU-Route 2.0

Fig. 3: Flowchart of NTHU-Route 2.0.

In the initial stage, NTHU-Route 2.0 uses five methods shown in Fig. 3 to find an initial routing result. The primary objective of this stage is minimizing the total wirelength. At the beginning, the multi-layer design is projected into a 2D plane. Next,
NTHU-Route 2.0 uses FLUTE [17] to decompose each multi-pin net into a set of two-pin nets and then the initial congestion map is produced by using probabilistic L-shaped pattern routing for each two-pin net. After that, NTHU-Route 2.0 applies the edge shifting technique [18] to reduce the congestions by changing the topology of each multi-pin net. At last, NTHU-Route 2.0 reroutes each two-pin net with an L-shaped pattern.

In the main stage, NTHU-Route 2.0 tries to improve the initial routing solution by using a rip-up and reroute technique based on a history cost function. If a two-pin net passes one or more overflowed edges, we call it an overflowed net. NTHU-Route 2.0 rips up overflowed two-pin nets and reroutes these nets one at a time. The net ordering for rip-up and reroute is determined as follows. NTHU-Route 2.0 calculates the congestion value of each edge \( e \) (i.e., \( d_e/c_e \)) at the subinterval identification step, and defines an interval \( I \) between the maximum congestion value and 1. The interval \( I \) is partitioned into \( m \) subintervals. For example, if the maximum congestion value among all edges is 2 and \( m \) is set to 10, then the subintervals are \([2, 1.9), [1.9, 1.8), \ldots, [1.1, 1)\). After that, every congested edge is assigned to one corresponding subinterval according to its congestion value. Next, NTHU-Route 2.0 picks the congested edges one at a time in a selected subinterval, and then keeps expanding a rectangular region \( r_e \) around each congested edge \( e \) until the average congestion value of the edges in \( r_e \) is less than or equal to the lower bound of the selected subinterval. After identifying \( r_e \), each overflowed two-pin net which has pins located in \( r_e \) is marked, and NTHU-Route 2.0 begins to rip-up and reroute the marked two-pin nets one at a time. The ordering of rip-up and reroute is in a descending order of bounding boxes of two-pin nets, because it thinks that a net with a bigger bounding box size may have more flexibility to find an overflow-free path without inducing more detours than a net with a smaller bounding box size.

Each ripped-up two-pin net is rerouted by monotonic routing [18] first. If monotonic routing cannot get a better path than the original path, the adaptive multi-source multi-sink maze routing method is applied, trying to find an overflow-free result. A history based cost function is adopted to formulate the edge cost during the reroute process. NTHU-Route 2.0 keeps repeating the congestion region identification and rip-up and reroute process until the total overflow is less than a user-defined threshold or the number of iterations reaches a user-defined value.

In the refinement stage, NTHU-Route 2.0 only focuses on finding an overflow-free path for each overflowed two-pin net. Instead of selecting overflowed nets by subinterval identification, NTHU-Route 2.0 selects two-pin overflowed nets directly in this stage, and then rips up and reroutes each two-pin net one by one. The rip-up and reroute ordering of nets is according to the number of overflowed edges that a two-pin net passes through. If there is a tie, a net with a larger bounding box will be selected first. NTHU-Route 2.0 makes the cost function of each edge simpler in this stage. The cost of each overflowed edge is set to 1, whereas the cost of each overflow-free edge is set to 0.

Finally, the layer assignment method [19] is applied for mapping the routing result in the 2D plane to the multi-layer result in the original 3D graph.

C. Review of Hybrid Unilateral Monotonic Routing

There are two types of unilateral monotonic routing introduced in [16]. Horizontally monotonic (HM) and vertically monotonic (VM) routing respectively identify the least-cost routing path from a source pin \( S \) to a target pin \( T \) in a rectangular search region by using minimal horizontal length and minimal vertical length. Fig. 4 shows a VM routing path and a HM routing path, where congested regions are in gray color. A unilateral monotonic routing path can be efficiently obtained by a dynamic programming method.

The hybrid unilateral monotonic (HUM) routing path from \( S \) to \( T \) is obtained as follows. First, for each node \( u \) in the search region, the method respectively finds the VM routing path \( VP(S, u) \) and the HM routing path \( HP(S, u) \), each of which connects \( S \) and \( u \). Fig. 5(a) and 5(c) respectively give examples of \( VP(S, u) \) and \( HP(S, u) \). Next, for each node \( u \) in the search region, this method respectively finds the VM routing path \( VP(T, u) \) and the HM routing path \( HP(T, u) \), each of which connects \( T \) and \( u \). See Fig. 5(b) and 5(d) for examples of \( VP(T, u) \) and \( HP(T, u) \), respectively. Then, for each node \( u \), the HUM routing path from \( S \) to \( T \) that passes through \( u \) is obtained from one of the following four path combinations: \((VP(S, v), VP(T, v)), (VP(S, v), HP(T, v)), (HP(S, v), VP(T, v)), \) and \((HP(S, v), HP(T, v))\), where each path combination concatenates two unilateral monotonic paths to produce a path from \( S \) to \( T \). Among the four combined paths, the one with lease cost is the HUM routing path from \( S \) to \( T \) that pass through \( u \) (see Fig. 5(e)). Finally, the HUM with least cost among all nodes in the search region is chosen to be the HUM routing path from \( S \) to \( T \).

III. Our Global Router

In this section, we describe our enhancements for NTHU-Route 2.0 to consider buses. The first enhancement is a new ordering method for rip-up and reroute. To keep the wirelength of each net of a bus as similar as possible, it is more beneficial to assign routing resources of congested regions to bus nets than non-bus nets because we do not want to see a bus net takes extra length to make a detour around a congested region. Therefore, we change the net ordering of rip-up and rerout in the main stage to achieve this. The next enhancement is a length-bounded hybrid unilateral monotonic routing method. NTHU-Route 2.0 adopts the monotonic routing method and the adaptive multi-source and multi-sink maze routing method to find a path with the least cost. However, we would like to find a path with an ideal length for a bus net, which means the length is closer to each of the other bus nets, the better. Therefore, we replace the monotonic routing method of NTHU-Route 2.0 by a new length-bounded hybrid unilateral monotonic routing method in the main stage.
remaining path combinations, our method keeps the one with the combination is not useful and therefore is abandoned. Among the it exceeds the length bound of the corresponding net, this path only has a cost but also has a length for the combined path. 

like [16]. However, each path combination in our method not like unilateral monotonic routing, each node has 4 path combinations length for each node during unilateral monotonic routing. After each of whose length is not more than the length bound of the up net, our router tries to find the least-cost path among paths, method, which is modified from [16]. Unlike [16], for each ripped-

2.0 by a new length-bounded hybrid unilateral monotonic routing B. Length-Bounded Hybrid Unilateral Monotonic Routing

regions which are partially caused by those bus nets.

bus nets could be guided to detour for avoiding the congested

unilateral monotonic routing path that combines paths in (b) and (c).

Illustration of the hybrid unilateral monotonic routing path Fig. 5:

HP(T, u) which connects T and u. (c) The HM routing path HP(S, u) which connects S and u. (d) The HM routing path HP(T, u) which connects T and u. (e) The hybrid unilateral monotonic routing path that combines paths in (b) and (c).

A. Net Ordering Determination

NTHU-Route 2.0 identifies a set of two-pin overflowed nets for each congestion subinterval at each iteration of the main stage. After that, these overflowed nets are ripped up and rerouted one at a time by a non-increasing order of bounding box size. But to take buses into account, each bus net with a length bound (we will define length bound in Section III.B) needs to keep the routing resources of congested regions and cannot release them to non-bus nets through rerouting when the path of the bus net currently passes through the congested regions. So bus nets need to be given a higher priority during rip-up and reroute.

Before we rip up and reroute those overflowed nets, we divide them into two sets: \( n_{bus} \) and \( n_{non-bus} \), where \( n_{bus} \) contains bus nets and \( n_{non-bus} \) contains non-bus nets. For each congestion subinterval, the bus nets are rerouted first. After that, we reroute the non-bus nets. Because all the bus nets are rerouted first, non-bus nets could be guided to detour for avoiding the congested regions which are partially caused by those bus nets.

B. Length-Bounded Hybrid Unilateral Monotonic Routing

We replace the monotonic routing method of NTHU-Route 2.0 by a new length-bounded hybrid unilateral monotonic routing method, which is modified from [16]. Unlike [16], for each ripped-up net, our router tries to find the least-cost path among paths, each of whose length is not more than the length bound of the net. In our method, we store the path cost as well as the path length for each node during unilateral monotonic routing. After unilateral monotonic routing, each node has 4 path combinations like [16]. However, each path combination in our method not only has a cost but also has a length for the combined path. Our method looks over the length of each combined path, and if it exceeds the length bound of the corresponding net, this path combination is not useful and therefore is abandoned. Among the remaining path combinations, our method keeps the one with the least cost for each node.

Suppose the length bound of the bus net \( n_1 \) is set to 8 in this example. Our router will rip up and reroute bus nets first because we need to give bus nets a higher priority for keeping.

Fig. 6 is an example of checking nodes after unilateral monotonic routing. Fig. 6(a) shows a global routing instance for a net. Suppose the length bound of the net is 12. Fig. 6(b) shows a path combination for node \( u \), and the combined path is overflow-free. However, the path length is \( 11 + 7 = 18 \) and exceeds the length bound, so the combined path is not useful and is abandoned. Fig. 6(c) shows a path combination for another node \( v \), and the combined path has an acceptable length 10 (5+5). Therefore, this combined path has the chance to become the final path chosen by our method, though it passes through a congested region and has a higher cost. Fig. 6(d) shows a path combination for another node \( w \), and its combined path has an acceptable length 12 (6+6), which is equal to the length bound. It has a lower cost because it successfully detours the congested regions, so it has a higher chance than the one in Fig. 6(c) to become the final path.

The rerouting methods for a bus net and a non-bus net are somehow different. Only bus nets are rerouted by the length-bounded hybrid unilateral monotonic routing method under their length bounds. If the net is a non-bus net, the original maze routing method of NTHU-Route 2.0 is applied on this net to seek for an overflow-free result, while the original monotonic routing method of NTHU-Route 2.0 is disabled. We keep continuing the whole rip-up and reroute process until the total overflow is smaller than a user-defined threshold or the number of iterations reaches a pre-defined number.

We now explain why we use different routing methods for each type of nets. A simple example is shown in Fig. 7. Fig. 7(a) shows a bus net \( n_1 \), a non-bus net \( n_2 \) and congested regions (in gray color). (a) A net to be routed, where congested regions are in gray color. (b) A path with long length and low cost. (c) A path with short length and high cost. (d) A path with short length and low cost.
the routing resources of congested regions. Therefore, the bus net $n_1$ is first rerouted by length-bounded hybrid unilateral monotonic routing. In order to meet the length bound constraint, the routing path passes through a congested region, but the routing path has the least cost because the passed overflowed edges have the least amount of overflow (see Fig. 7(b)). Then, our router reroutes the non-bus net $n_2$ by the adaptive multi-source multi-sink maze routing method, so $n_2$ will avoid passing through the congested region and detour to find an overflow-free path. Finally, we have two overflow-free nets that are shown in Fig. 7(c).

Therefore, at each rip-up and reroute iteration, a bus net can hold the routing resources of congested regions because nearby non-bus nets will make a detour to avoid the congested regions that are caused by bus nets, and bus nets can try not to detour and reduce the resultant bus deviations.

At each iteration of the main stage of our router, a length bound is imposed on each bus net such that if the net gets rerouted, its new wirelength should not exceed the length bound. Our empirical observation indicates that the length bound determination method is crucial to the solution quality. If the length bound is set too large, it loses the effect on controlling the wirelength. On the other hand, if the length bound is set too small, it is difficult for our router to find a legal result. In order to determine a length bound for each two-pin bus net, we calculate the Manhattan distance $L$ between the two pins. Then the initial length bound for the net is set to $L$. However, if we keep the length bound of each net of a bus fixed in each iteration, those congested regions that are passed by bus nets will always be crowded. Therefore, we will increase the length bound little by little to relieve the congested regions that are caused by bus nets. If a net of a bus is overflowed, the length bound of each net of the bus is increased by one at the next iteration. Increasing length bound will also increase the bus wirelength deviation, but the overflow of each bus net will be reduced effectively.

**IV. Experimental Results**

In this section, we present the experimental results of our bus-aware global router. Our router is implemented in C/C++ language on a Linux machine with Intel Xeon 2.0GHz and 96G memory. To test our global router, we modified the benchmarks from the ISPD08 global routing contest [2] and added bus nets. Table 1 shows the detailed information of each test case, where the columns “Nets”, “Bus nets”, and “Bus” respectively give the amounts of all nets, bus nets, and buses. Note that each bus has a number of bus nets (8, 16, 32 or 64). We compare our router with NTHU-Route 2.0, where NTHU-Route 2.0 runs on the same machine as ours.

Table 2 shows the experimental results. We compare the results in terms of total overflow (TOF), total bus wirelength deviation that does not include via wirelength (BD), total wirelength (TWL), and runtime measured in minute (CPU). The experimental results show that our router successfully solve 9 of 11 test cases without overflow. Although our router causes more overflow in bigblue4 and newblue4 than NTHU-Route 2.0, we can find one more overflow-free solution (i.e., for newblue5). In addition, compared with NTHU-Route 2.0, our router can get much smaller total bus wirelength deviation for each test case. Overall, our router can reduce total bus wirelength deviation by 55.9% on average. Since our router needs to detour to avoid the congested regions caused by buses when rerouting non-bus nets, the average total wirelength result of our router is slightly worse than that of NTHU-Route 2.0 by 1%. Due to the fact that our router takes buses into consideration, our router may reduce a less amount of overflow than NTHU-Route 2.0 in each iteration of the main stage. Therefore, our router may need more iterations to converge in the main stage. In other words, our router may need more time to complete the global routing, although the length-bounded hybrid unilateral monotonic routing method is faster than the adaptive multi-source multi-sink maze routing method. According to our experimental results, the average runtime of our router is 5.5% more than NTHU-Route 2.0.

We also compare the wirelength deviation of each bus between the results of NTHU-Route 2.0 and our router. In Table 3, column “#BSBD” indicates the number of buses that have smaller wirelength deviation produced by our router than by NTHU-Route 2.0, and column “SBD” indicates the total reduction of bus wirelength deviation among those buses. Column “#BLBD” indicates the number of buses that have larger wirelength deviation produced by our router than by NTHU-Route 2.0, and column “LBD” indicates the total addition of bus wirelength deviation among those buses. “#BTBD” indicates the number of buses that have same bus wirelength deviation produced by our router and by NTHU-Route 2.0. Table 3 shows that our router can produce the routing results in which most buses have no greater bus wirelength deviation than NTHU-Route 2.0. Although some buses have larger wirelength deviation in our results, the SBD results are always much greater than LBD ones. Due to the fact that buses affect each other, a small addition of bus wirelength deviation in one bus may help reduce more bus wirelength deviation in other buses by our router.

**V. Conclusion**

In this paper, we present a global router considering buses. Our router can control the lengths of bus nets by a length-
Table 2: Comparison between NTHU-Route 2.0 and our router.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>TOF</th>
<th>BD</th>
<th>TWL (e5)</th>
<th>CPU (min)</th>
<th>TOF</th>
<th>BD</th>
<th>TWL (e5)</th>
<th>CPU (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptec1</td>
<td>0</td>
<td>219704</td>
<td>61.0</td>
<td>8.2</td>
<td>0</td>
<td>199882</td>
<td>62.7</td>
<td>11.2</td>
</tr>
<tr>
<td>adaptec2</td>
<td>0</td>
<td>129494</td>
<td>62.8</td>
<td>2.1</td>
<td>0</td>
<td>95914</td>
<td>63.4</td>
<td>2.6</td>
</tr>
<tr>
<td>adaptec3</td>
<td>0</td>
<td>109754</td>
<td>146.5</td>
<td>5.1</td>
<td>0</td>
<td>42954</td>
<td>146.9</td>
<td>4.5</td>
</tr>
<tr>
<td>adaptec4</td>
<td>0</td>
<td>88760</td>
<td>135.4</td>
<td>2.2</td>
<td>0</td>
<td>41776</td>
<td>135.7</td>
<td>2.1</td>
</tr>
<tr>
<td>adaptec5</td>
<td>0</td>
<td>234564</td>
<td>165.4</td>
<td>13.3</td>
<td>0</td>
<td>83172</td>
<td>167.0</td>
<td>15.8</td>
</tr>
<tr>
<td>bigblue3</td>
<td>0</td>
<td>47912</td>
<td>142.2</td>
<td>4.0</td>
<td>0</td>
<td>18218</td>
<td>142.3</td>
<td>3.5</td>
</tr>
<tr>
<td>bigblue4</td>
<td>102</td>
<td>206684</td>
<td>240.5</td>
<td>13.8</td>
<td>138</td>
<td>49182</td>
<td>241.6</td>
<td>13.6</td>
</tr>
<tr>
<td>newblue2</td>
<td>0</td>
<td>31544</td>
<td>84.9</td>
<td>0.6</td>
<td>0</td>
<td>19148</td>
<td>85.2</td>
<td>0.7</td>
</tr>
<tr>
<td>newblue4</td>
<td>158</td>
<td>328758</td>
<td>139.6</td>
<td>65.0</td>
<td>180</td>
<td>61978</td>
<td>140.5</td>
<td>27.9</td>
</tr>
<tr>
<td>newblue5</td>
<td>2</td>
<td>200620</td>
<td>245.0</td>
<td>15.0</td>
<td>0</td>
<td>78216</td>
<td>246.0</td>
<td>14.5</td>
</tr>
<tr>
<td>newblue6</td>
<td>0</td>
<td>317738</td>
<td>189.4</td>
<td>19.0</td>
<td>0</td>
<td>185900</td>
<td>197.5</td>
<td>30.8</td>
</tr>
</tbody>
</table>

Table 3: Comparison of wirelength deviation of each bus between NTHU-Route 2.0 and our router.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#BSBD</th>
<th>SBD</th>
<th>#BLBD</th>
<th>LBD</th>
<th>#BTBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptec1</td>
<td>40</td>
<td>116056</td>
<td>6</td>
<td>6244</td>
<td>4</td>
</tr>
<tr>
<td>adaptec2</td>
<td>37</td>
<td>36708</td>
<td>7</td>
<td>3128</td>
<td>6</td>
</tr>
<tr>
<td>adaptec3</td>
<td>44</td>
<td>60800</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>adaptec4</td>
<td>33</td>
<td>47004</td>
<td>1</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>adaptec5</td>
<td>36</td>
<td>152796</td>
<td>9</td>
<td>1404</td>
<td>5</td>
</tr>
<tr>
<td>bigblue3</td>
<td>24</td>
<td>30636</td>
<td>3</td>
<td>942</td>
<td>23</td>
</tr>
<tr>
<td>bigblue4</td>
<td>39</td>
<td>158004</td>
<td>3</td>
<td>502</td>
<td>8</td>
</tr>
<tr>
<td>newblue2</td>
<td>23</td>
<td>12646</td>
<td>2</td>
<td>250</td>
<td>25</td>
</tr>
<tr>
<td>newblue4</td>
<td>44</td>
<td>268084</td>
<td>3</td>
<td>1304</td>
<td>3</td>
</tr>
<tr>
<td>newblue5</td>
<td>41</td>
<td>123080</td>
<td>5</td>
<td>1676</td>
<td>4</td>
</tr>
<tr>
<td>newblue6</td>
<td>38</td>
<td>138512</td>
<td>8</td>
<td>6674</td>
<td>4</td>
</tr>
</tbody>
</table>

The experimental results show that our router can greatly reduce the total bus wirelength deviation while achieving comparable total overflow, total wirelength and runtime.

A possible future work is to take buses into account in the layer assignment stage, which considers the length-matching issue for vias and makes our bus-aware global router more complete.

REFERENCES


