Physical Synthesis of Bus Matrix for High Bandwidth Low Power On-chip Communications[*]

Renshen Wang[†], Evangeline Young[‡], Ronald Graham[†] and Chung-Kuan Cheng[†]
[†]University of California, San Diego, La Jolla, CA 92093-0404
[‡]The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong
{rewang, ckcheng, graham}@cs.ucsd.edu, fyyoung@cse.cuhk.edu.hk

ABSTRACT

As the thermal wall becomes the dominant factor limiting VLSI circuit performance, and the interconnect wires become the primary power consumer, power efficiency of on-chip data throughput is nowadays a critical target for SoC designers. Under this trend, bus matrices are mostly used in current system-on-chips (SoCs) because of their simplicity and good performance. We introduce a bus matrix synthesis flow to optimize on-chip communications, to keep the low delay of buses, reduce power by bus gating, and reduce wires by wire sharing. The proposed algorithms are able to help designers create high capability yet compact and efficient bus matrices for future low power SoCs.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-aided design

General Terms

Algorithms, design, performance

Keywords

Bandwidth, power efficiency, wire efficiency

1. INTRODUCTION

Advances in process technology have enabled more components integrated into a single silicon wafer as a system-on-chip (SoC), thanks to the scaling of feature size under Moore's law. Meanwhile, clock frequencies of the components as well as the system have stopped increasing due to the "thermal wall" and power issues. As a result, higher performance can only be extracted from parallelism, where communication between components will be a possible bottleneck for many applications. It is illustrated in [4] that power consumption of local computation can be significantly less than the power of moving data from component to component on the chip. Therefore, efficient on-chip communication architectures will become another critical part in future SoC designs.

[*]This work is partially supported by NSF CCF-0811794 and California Discovery program.

1.1 Bus vs NoC

Bus and network-on-chip (NoC) are the two types of popular on-chip communication architectures. Bus has been widely used for its speed and simplicity [16] [18], but lacks the communication bandwidth to support parallelism. Bus matrix [17] extends its bandwidth, but not in an efficient way on power or wires [11] compared to NoC [5], which is therefore regarded as a better choice for many applications because of its bandwidth capacity, regularity and scalability.

However, NoC has relatively large delay, which is a critical disadvantage to system performance, because communications in NoC must take a series of hops on routers in the network. Even with sophisticated routers [9] taking only one clock cycle each hop, the total delay over a long path is still significant. The accumulation of delay on hops is inevitable due to the independency of routers, and it scales up with the number of routers and system complexity. Therefore, we believe bus based communication provides better performance in delay-sensitive systems, because bus delay can be minimized through centralized control and arbitration. On the other hand, the weaknesses such as low bandwidth and wire efficiency, are not intrinsic in bus. In this paper, we address these issues on bus matrix to make it a capable and efficient on-chip communication architecture.

1.2 Previous works

There is a large body of work on system-level communications, such as [2] [7] [8] [10] [11] [12] [14], *etc.* Most of the works take approach by optimizing the bus topologies for good bandwidth and/or power consumption. On the analysis of performance, usually a "communication constraint graph" [8] is extracted from a specific application, and different topologies can be evaluated by estimative calculations [8] or detailed simulations [10] [11].

Over the time, optimization on bus architectures have been changing due to technology scaling and its effect on signal transmissions. Long interconnections are relatively scaling up [4] [6] on power and wires, therefore considered as a more important minimization target. Physical synthesis has not been emphasized in literature, but because of these changes, on-chip physical locations of components are becoming more and more relevant to both power and performance of SoCs. On the performance side, delay is dictated by propagation distance, bandwidth is limited by routing congestion, and ultimately limited by power/thermal constraints. On the power side, wire capacitance is consuming a higher portion of dynamic power.

Floorplans are considered in [10] to estimate delays, but not power. Bus gating is introduced in [14] to minimize delay and power within a given floorplan, but without the consideration on performance with high bandwidth requirement. Moreover, most previous works assume that the bus or bus matrix are using tree structures. New objectives necessitate explorations on more possibilities on topologies, which are strongly correlated with geometrical properties, affecting both wire routing and data routing.

1.3 Paper overview and contributions

In this work, we optimize bus matrix for bandwidth, power and wires. We use ABMA AHB [16] or AMBA AXI [17] bus architectures as representatives, since they are the most popular on-chip buses in industry. To reduce power, gated bus proposed in [14] are used, and the topology is therefore based on Steiner graphs instead of traditional tree structures.

The concept of gated bus on shortest-path Steiner graph dramatically changes some previous conclusions on bus power saving. In [11], the full AMBA AHB bus matrix makes a crossbar like connection, ensures high bandwidth for master-slave data transactions, but also consumes a lot of power. With bus gating techniques, the full bus matrix can minimize power while maintaining the high bandwidth, because every connection can take a single shortest path which involves minimal wires. This optimal crossbar connection, however, may not be achievable because of the limited routing resource. The total wire length increases on the scale of O(n^2) if we naively route every master-to-slave connection. So we introduce optimization techniques based on the Steiner graph to solve this problem, and provide a framework to explore tradeoffs among power, bandwidth and wires.

In the rest of this paper, section 2 proposes problem formulations for ideal bus matrices and tractable, place-and-route friendly bus matrices. Section 3 proposes a heuristic algorithm to further reduce routing congestion. Section 4 shows the experiments on the test cases in [14], and section 5 finalizes our conclusions.

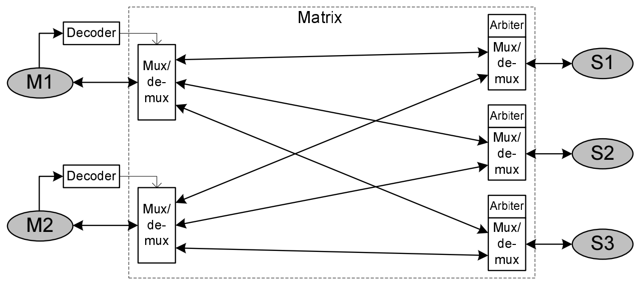

Figure 1: Sketch of a power efficient full bus matrix

2. PROBLEM FORMULATIONS

Current bus architectures like AMBA AHB and AXI can be flexibly reconfigured for on-chip communication patterns. To represent the pattern, we use the “communication constraint graph” in [12]. Here with AMBA AHB protocols, it is a bipartite graph $G = (U, W, A)$ where U is the set of masters, W is the set of slaves, and A is the set of arcs from U to W. $(u, w) \in A$ means master u will access slave w.

Figure 1 is the sketch of a full 2×3 bus matrix. This type of full matrix guarantees maximum parallelism in SoCs, because the only limitation on the bandwidth is the throughput of component units: Each master can only access one slave at a time, and vice versa. And if we use bus gating in [14] and add the multiplexers and demultiplexers, each data transaction between a master and a slave only takes one path, which can take the shortest Manhattan distance. Thus, the bus matrix also guarantees high power efficiency.

The only problem with this full matrix is that all these paths may take too much on-chip routing resource, possibly impractical with a large SoC and intensive communications. So our formulation below emphasize on wire length reduction, while maintaining the high bandwidth capabilities and low power consumption.

2.1 Ideal bus matrix formulation

Given a bipartite communication graph $G = (U, W, A)$ of an SoC, and given that for every component $v \in (U \bigcup W)$, its physical location on $P(v) \in R^2$ is determined by the floorplan, our goal is to find a set of wires and control units that can meet the communication demand, while each data transaction takes the shortest Manhattan distance between the master and slave. We denote the Manhattan distance between a and b as $\|P(a) - P(b)\|_1$, and the set of paths (same defition as in [12]) as Π.

Definition 1. For communication graph $G = (U, W, A)$ and location function P, an ideal bus matrix graph is a weighted graph $\Theta = (V, E, \omega)$ that

- $U \subseteq V$
- $W \subseteq V$
- For any $A' \subseteq A$ such that $\forall\ (u_i, w_i), (u_j, w_j) \in A'$, $i \neq j \Rightarrow u_i \neq u_j$ and $w_i \neq w_j$, there is a set of paths $\Pi' \subseteq \Pi$, such that
  - ∘ $|\Pi'| = |A'|$
  - ∘ $\forall\ r \in \Pi'$, $r \subseteq V \bigcup E$
  - ∘ $\forall\ (u, v) \in A'$, $\exists\ r \in \Pi'$ such that $u \in r$, $v \in r$, and $\sum_{(i,j)\in r} \|P(i) - P(j)\|_1 = \|P(u) - P(v)\|_1$
  - ∘ $\forall\ e \in E$, $|\{r \in \Pi' : e \in r\}| \leq \omega(e)$

The objective is to find the bus matrix graph with minimal total wire length $L(\Theta) = \sum_{(u,v)\in E} \omega((u, v))\|P(u) - P(v)\|_1$.

Due to the limited bandwidth on each single component, there is no need for all the arcs in A to have disjoint paths. Only for the subsets of A in which the arcs do not share any vertices (A'), we require that the superposition of all the paths are covered by the bus matrix, so any data transactions not naturally conflicting can be simultaneously active.

This formulation defines an ideal high bandwidth and low power on-chip communication solution. However, it is computationally expensive to obtain the minimal cost of Θ, because the combinations of possible subsets A' is exponential. Even if we have a solution of graph Θ, it is still impractical to store the paths of every Π' in bus control units, or compute the path set Π' in real time for each subset A'. To make it achievable with small silicon resource and high efficiency, we change the formulation as follows.

2.2 Practical bus matrix formulation

Given a communication graph $G = (U, W, A)$ and location function P same as previous subsection, we assign a fixed path $\rho(a) \in \Pi$ for each arc $a \in A$, i.e. each pair of master-slave always takes the same path when they are connected.

Definition 2. For communication graph $G = (U, W, A)$ and location function P, a bus matrix graph is a weighted graph $H = (V, E, \omega)$ with a set of paths $\rho : A \mapsto \Pi$ that

- $U \subseteq V$
- $W \subseteq V$
- $\forall\ a \in A,\ \rho(a) \subseteq V \bigcup E$
- $\forall\ (u, v) \in A$, $\sum_{(i,j)\in\rho(a)} \|P(i) - P(j)\|_1 = \|P(u) - P(v)\|_1$
- For any $A' \subseteq A$ such that $\forall\ (u_i, w_i), (u_j, w_j) \in A',\ i \neq j \Rightarrow u_i \neq u_j$ and $w_i \neq w_j$, we have $\forall\ e \in E,\ |\{a \in A' : e \in \rho(a)\}| \leq \omega(e)$

By this formulation, the fixed path for each arc in A can be easily stored in control units, so the arbitration and control can be implemented with small silicon footprint.

3. ALGORITHMS FOR BUS MATRIX SYNTHESIS

The formulation in definition 2 is more suitable for optimization of on-chip bus architectures. To find the minimal total wire length, it is still a hard problem because the bus matrix graph H should be a Steiner graph of G. Steiner graph is defined in [1] and used in [14] to define the topology of a gated bus, and is shown to be very effective for power reduction on the basic AMBA AHB bus.

The shortest paths in [14] have the same power saving effect in bus matrix, which is extended from the bus adding multiple data transactions. Since the shortest-path Steiner graph is also minimizing wire length, we adopt the same topological structure of the Steiner graph and add weight function ω to meet the communication requirement by G.

3.1 Graph generation

Finding the minimal shortest-path Steiner graph is NP-hard, since the single-source case, minimal rectilinear Steiner arborescence (MRSA) problem is NP-hard [13]. But we have efficient heuristic algorithms to find solutions close to minimal [3] [14]. Once we have the Steiner graph $S_G = (V, E)$ generated from communication graph G, each arc (u, v) in G has a path of length exactly the Manhattan distance between $P(u)$ and $P(v)$. Also we have the path set Π which contains the paths for all the arcs.

To allow multiple data transactions, some edges need larger weight, i.e. multiple bus lines. Although on topological level (as figure 1), it seems that we can only add all the paths into the Steiner graph H, the actual number of wires needed on each edge can be largely reduced.

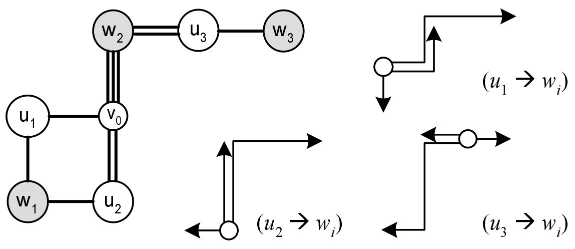

Figure 2: A 3×3 bus matrix and its 12 paths

Figure 2 is an example of a bus matrix for communication graph $G = (U, W, A)$, where $|U| = 3$, $|W| = 3$, and $|A| = 9$, so $(u_i, w_j) \in A$ for all i, j. Graph H with $|V| = 7$ is shown with edge weight drawn as number of parallel lines, and the paths for all the arcs are denoted in the three subgraphs.

Although there are 9 different paths of data transactions, only one edge has weight 3, (v_0, w_2) shared by path $(u_1 \rightarrow w_2)$, $(u_2 \rightarrow w_3)$ and $(u_3 \rightarrow w_1)$. Any other edge does not need 3 parallel bus lines even when it is covered by 3 or more paths, because those paths can not be active at the same time for sharing either a common u_i or a common w_i. The weight required by each edge is the maximum subset A' in definition 2. Regardless of the possible combinations on A', the maximum subset can be computed by the maximum bipartite matching algorithm [15]. The bipartite graphs for the 7 edges in figure 2 are shown in figure 3 below.

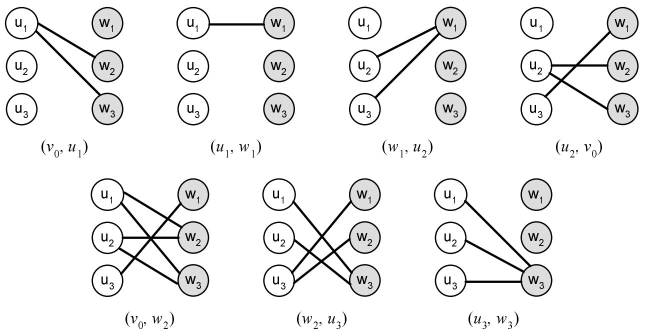

Figure 3: Bipartite graph of each edge in figure 2

Hence, the weight function ω is created by the subgraphs of the communication graph G, so that each edge line is shared by multiple paths whenever possible. And since the topology of the shortest-path Steiner graph S_G it based on is also generated for sharing edges as much as possible, the resulting bus matrix graph $H = (V, E, \omega)$ has a total cost much lower than the sum of path lengths (11 vs 20 in the example of figure 2).

3.2 Parallel segment merging for wire length reduction

With more components in SoC, high bandwidth bus matrix will need more wires to support parallelism in communications. Especially when components are placed in an irregular floorplan instead of cell arrays, the shortest-path Steiner graph generated by the algorithm in [14] may contain a lot of loops, which bring additional wires. As in figure 4(a), there are long and narrow rectangles formed by graph edges, where the long edges in parallel are necessarily constructed in S_G to support different shortest paths.

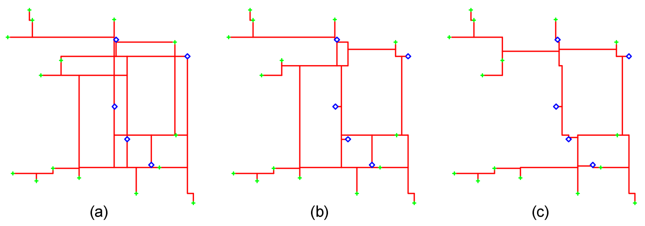

Figure 4: Segment merging

If we loosen the requirement on the path length of data transactions, e.g. from the exact shortest distance to within $(1+\epsilon)$ Manhattan distance, then the long edges of a narrow rectangle can be merged into a single edge. Assume the rectangle has dimensions $h \times w$, and the new edge is placed in the middle of the parallel edges, the total wire length is reduced by about h, while the lengths of transaction paths will increase by at most $\frac{w}{2} + \frac{w}{2} = w$. For long and narrow rectangles with large h/w ratios, this operation can greatly help reducing edges in the Steiner graph and reducing the total wire length in the bus matrix, at a relatively low cost of power increase.

The algorithm should iteratively find parallel double segments in the Steiner graph, calculate the potential of edge length reduction Δl and the possible path length increment Δp in a merging. The parallel segments with largest $\Delta l/\Delta p$ are merged in each iteration. Eventually, there will be no positive Δl found in the graph, and we have a series of Steiner graphs with decreasing edge length and increasing path lengths, where the best tradeoff can be selected.

To find these parallel segments, we illustrate the process for scanning and combining vertical edges in figure 5. The vertical line segments in the Steiner graph are first sorted by their x coordinates as $s_1, s_2, \cdots, s_k$. Then for each pair of segments s_i, s_j $(i < j)$ with a common y interval $[y_1, y_2]$, if between i and j there is no other vertical segment on $[y_1, y_2]$, do the following calculations.

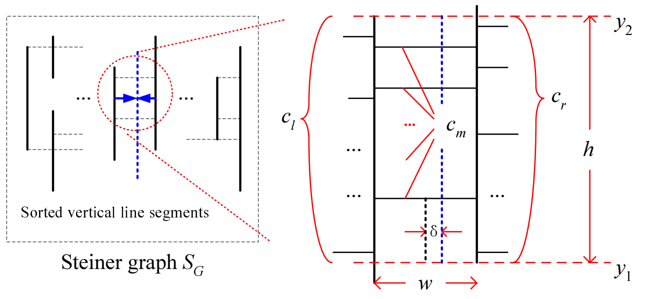

Figure 5: Searching for mergeable parallel segments (vertical only)

Let c_l denote the count of horizontal lines connecting to the left, c_r the count connecting to the right, and c_m the count connecting s_i and s_j in the middle. Assume $c_l < c_r$, so the combined vertical segment may not be at the middle but have an offset δ to the right of the midpoint.

The reduction in edge length Δl is by removing a vertical segment of length h, and making some changes in horizontal segments. For the possible increment on path lengths Δp, since the left vertical segment is pushed right by $\frac{w}{2} + \delta$, a path may need to detour and add 2 times that distance. So

$$\frac{\Delta l}{\Delta p} = \frac{h + c_m w - c_l(\frac{w}{2} + \delta) - c_r(\frac{w}{2} - \delta)}{w + 2\delta}$$
$$= \frac{c_r - c_l}{2} + \frac{h - (c_r - c_m)w}{w + 2\delta}$$

Offset δ can be decided by the right part $\frac{h-(c_r-c_m)w}{w+2\delta}$ when we try to maximize $\frac{\Delta l}{\Delta p}$. If $h-(c_r-c_m)w \geq 0$, or $\frac{h}{w} \geq c_r-c_m$, then let $\delta = 0$, so the merged vertical segment is placed at the middle. Otherwise, $\frac{h}{w} < c_r - c_m$, we let $\delta = \frac{w}{2}$ which is the maximum value it can take, and the merged segment is at the right most position (assuming $c_l < c_r$).

Figure 4 shows the effect of a Steiner graph being processes by the merging algorithm. All the long and narrow rectangles are removed by the operations without significantly increasing average length of data transactions. This result is not surprising since we always choose the merging with maximum edge length reduction over path length increment. More test cases are shown in section 4.

3.3 Overall optimization flow

Merging parallel segments can significantly reduce the total edge length in a Steiner graph. However, its effect on the weighted bus matrix graph H will be smaller, since the number of paths is not reduced, and less edges will generally result in larger average edge weight. Nevertheless, the mergings are still helpful for reducing the total cost of H. Also with less edges, less switches are needed in the bus matrix and the control overhead is reduced.

The overall optimization flow of bus matrix is shown below. It generates a series of bus matrix graphs with maximum bandwidth capacity.

Table 1: Bus matrix physical synthesis

```
Given a communication graph $G = (U, W, A)$,
and a location function $P : U \bigcup W \mapsto R^2$
1. Generate shortest-path Steiner graph $S_G = (V, E)$
   by the algorithm in [14];
2. Repeat
       For each arc $a = (u, v) \in A$,
           find a shortest path $\rho(a)$ from $u$ to $v$;
       For each edge $e \in E$,
           $A' \leftarrow \{a \in A : e \in \rho(a)\}$;
           $\omega(e) \leftarrow$ Max_match$(A')$;
       Bus matrix graph $H_i = (V, E, \omega)$;
       $i \leftarrow i + 1$;
       Find the parallel segments with maximum $\frac{\Delta l}{\Delta p}$;
       Merge the two segments into one in $S_G$;
   Until ($S_G$ has no parallel segments with $\Delta l > 0$)
3. Evaluate all the bus matrix graphs $\{H_i\}$
```

To implement a bus matrix configuration from graph $H = (V, E, \omega)$, we route $\omega(e)$ sets of bus wires on each edge, and also put switches on each Steinter node. The switches will be more complex than the single switch in [14], because there are multiple bus lines per edge and multiple paths a time. However, under current and future technology, crossbar connection at a point can be implemented with small footprint and low power. So the power on switches is generally low and not counted in the average power of data transactions in our experiments.

4. EXPERIMENTAL RESULTS

We implement the shortest-path Steiner graph generation, the edge weight maximum matching, and the parallel segment merging heuristic. Test cases from [14] are used. On the scale of these data sets, all the programs can finish running in no greater than a few minutes, so the platform information and running times are omitted.

In our test cases, we assume the maximum bandwidth should be guaranteed, so the bus matrix can perform as a crossbar switch. Based on this requirement, the objective can be adjusted for different optimization targets depending on resource availability and designer's choices. We also assume the optimizations are performed on data bus, which always require both master-to-slave and slave-to-master connections. The address bus is slightly different, but can be handled with the same principle.

4.1 Min-power bus matrix

Since most power consumption will be from the data transactions on wires [4], the minimum power configuration is the

H_1 from the algorithm with no parallel segment merging. The total wire length is relatively high, but by extensive wire sharing in section 3.1 and [14], our physical synthesis provides much better results than directly routing the paths from the communication graph G.

In each case (m, n), m is the number of masters and n is the number of slaves. In the random cases T2∼T12, all the components' physical locations are distributed on a 1000×1000 square. The total path length ($\sum L_p$), average path length ($\overline{L_p}$), total edge length ($\sum L_e$, sum of all edge lengths ignoring edge weight ω) and total wire length ($\sum L_{wire}$) are listed in the following table.

Table 2: Results for minimal power

Case(m,n)	$\sum L_p$	$\overline{L_p}$	$\sum L_e$	$\sum L_{wire}$
T0 (3,16)	30500	635	5700	9300
T1 (3,16)	84200	1754	5700	10500
T2 (2,30)	40122	669	6961	10117
T3 (3,16)	33179	691	4240	7168
T4 (5,15)	51660	689	6524	14136
T5 (6,16)	66626	694	9427	23038
T6 (8,8)	44078	689	6631	14606
T7 (12,6)	47282	657	7456	15702
T8 (16,10)	109278	683	10453	32429
T9 (8,16)	79110	618	9529	27274
T10 (8,16)	95828	749	8828	27663
T11 (6,12)	48130	668	5946	14265
T12(12,12)	96276	669	9747	27497

From the table above, the total wire length in our min-power bus matrix is about one third to one fourth of the total path lengths, regardless of the number of masters or slaves. This shows the efficiency of physical synthesis which extensively exploits the physical location information of all the on-chip components. When the on-chip routing resource is able to accommodate all the wires, this min-power bus matrix is always preferable.

4.2 Min-wire bus matrix

On large cases, especially when the components are irregularly placed, we often need to reduce wires by the merging algorithm in section 3.2. In the same test cases as above, we run the algorithm of table 1 and take the H_i with minimum total wire length. The results are in table 3.

Table 3: Results for minimal total wire length

Case(m,n)	$\sum L_p$	$\overline{L_p}$	$\sum L_e$	$\sum L_{wire}$
T0 (3,16)	31500	656	5000	9200
T1 (3,16)	84200	1754	5700	10500
T2 (2,30)	42654	710	4171	8931
T3 (3,16)	33185	691	4240	7168
T4 (5,15)	56340	751	4190	10911
T5 (6,16)	69494	724	6777	18232
T6 (8,8)	48234	754	4812	12445
T7 (12,6)	48154	669	6202	12988
T8 (16,10)	116602	729	7584	23732
T9 (8,16)	83108	649	6717	20761
T10 (8,16)	101702	795	5359	18492
T11 (6,12)	48440	672	5735	13271
T12(12,12)	100832	700	7717	21348

Based on the bus matrices of minimal power, there is a relatively large deviation on the effects of wire length reduction. From no reduction at all on the highly regular T1 case, to about 33% of reduction on the T10 case. On most random cases, about 1/4 to 1/6 reduction on the wire length from the min-power bus matrix can be achieved.

Compared to the reduction on wire length, the increase on the average path length is much smaller, ranging from 0 to 9% and mostly under 5%. On the T8 case where 27% of wire length is reduced, the average path length is increased by 6.7%. The two corresponding bus matrix graphs H_1 and H_{18} of T8 are shown in figure 6 below, where the thickness of each edge e indicates its weight $\omega(e)$. We can see from the figure that most high weight edges are in the central area of the chip, because most paths need to go through this area in order to be shortest. Comparison between the two graphs shows how edges can be combined and redundancy be reduced for a compact and efficient bus matrix.

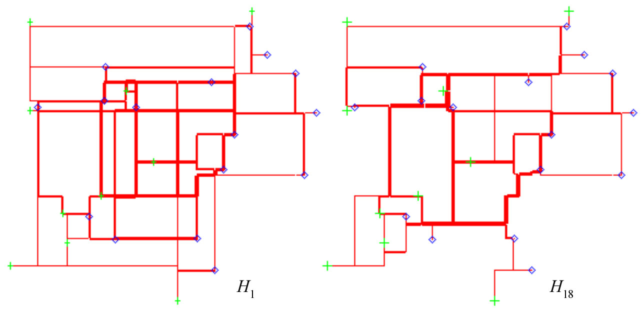

Figure 6: Min-power & min-wire on the T8 case

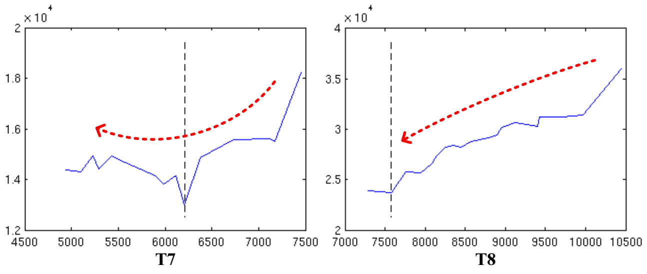

Figure 7: Total wire length vs total edge length

By further observation on these cases, we have interesting curves on the total wire length of $H_1, H_2, H_3, \cdots$. Usually, in the first few mergings, the wire length is decreasing because the wires can be shared by more paths. In later steps, it will go up again because the average length of the paths is increasing and cancel out the wire sharing benefits. The minimal wire length point may vary largely depending on the distribution of masters and slaves. The curves of T7 and T8 are drawn in figure 7, where the total edge lengths are on horizontal axes, and the total wire lengths on vertical axes. The two graphs of T8 in figure 6 correspond to the highest point and the lowest point in the right chart.

4.3 Tradeoffs on power, wire and bandwidth

The wire merging algorithm provides more space efficient bus matrices at the cost of only a small increase on power consumption. Between the min-power and min-wire bus matrices, more choices are available on the tradeoff. Since we generally increase average path length by parallel segment merging, by figure 7 we should make the power-wire tradeoff on the right half of T7's curve, while on T8 it should be the entire curve. The results are in figure 8. The shape of a tradeoff curve like these largely depends on the detailed configuration of the Steiner graph generated from the system floorplan.

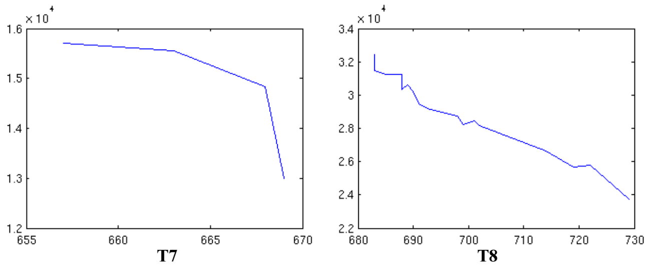

Figure 8: Total wire length vs average path length

If allowed by the system performance requirement, bandwidth capability can also be added into the tradeoff. Without compromise, the bus matrix should have the capacity of handling $\min(m, n)$ data transactions at the same time. Of course we can save wire/power by reducing this capacity. For bandwidth capacity k, each edge e will need $\min(\omega(e), k)$ copies of bus wires. If we need only one transaction a time, the total edge length in table 2 and 3 will be the exact total wire length. For more detailed results, we put in figure 9 the curves of total wire length for different bandwidth capacities on the two bus matrices in figure 6. The two curves are generally similar, from which we can observe the weights on all the edges in H_1 and H_{18}.

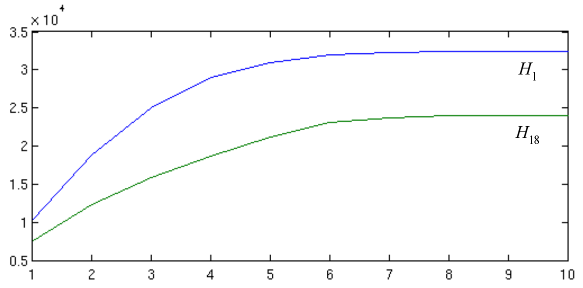

Figure 9: Wirelength vs bandwidth on the T8 case

By figures 7, 8 and 9, the overall design of SoC communications are very flexible. Solutions optimized by our bus matrix synthesis algorithm are applicable to various design choices. With more detailed communication patterns of the SoCs, such as specific sets of simultaneous master activities from simulation or testing results, the maximum matching can be more effective in reducing wire lengths and/or increasing bandwidth capacities.

5. CONCLUSIONS

We optimize the AMBA AHB bus matrix architecture for on-chip communications. Compared to network-on-chip (NoC), it has great advantage on delay, with a similar total bandwidth capacity. On power efficiency, it is also at advantage since the data is always taking the shortest (or close to shortest) path with minimal control overhead.

Under the goal of maximizing data throughput and minimizing data movement on chip [4], we devise algorithms which extensively exploit the physical design space, a necessity for a thorough optimization on power efficiency. The results show promising potentials of bus matrices in future low power SoCs. More possibilities on formulations and algorithms are to be explored on bus matrix architectures.

6. REFERENCES

[1] B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Math.*, pages 1029–1055, 2005.

[2] J. Y. Chen, W. B. Jone, J. S. Wang, H. I. Lu, and T. F. Chen. Segmented bus design for low power systems. *IEEE Trans. VLSI Systems*, 7(1):25–29, 1999.

[3] J. Cong, A. B. Kahng, and K.-S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. *IEEE Trans. Computer-Aided Design*, 17(1):24–39, Jan. 1998.

[4] W. Dally. Keynote: The end of denial architecture and the rise of throughput computing. *ACM/IEEE Design Automation Conf.*, 2009.

[5] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection network. *ACM/IEEE Design Automation Conf.*, 2001.

[6] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings IEEE*, 89:490–504, 2001.

[7] K. Lahiri and A. Raghunathan. Power analysis of system-level on-chip communication architectures. *Int'l Conf. Hardware-Software Codesign and System Synthesis*, pages 236–241, 2004.

[8] K. Lahiri, A. Raghunathan, and S. Dey. Power analysis of system-level on-chip communication architectures. *Int'l Conf. Computer-Aided Design*, pages 424–430, 2000.

[9] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. *IEEE Int'l Symp. Computer Architecture*, 2004.

[10] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane. Floorplan-aware automated synthesis of bus-based communication architectures. *ACM/IEEE Design Automation Conf.*, 2005.

[11] S. Pasricha, Y.-H. Park, F. J. Kurdahi, and N. Dutt. System-level power-performance trade-offs in bus matrix communication architecture synthesis. *Int'l Conf. Hardware-Software Codesign and System Synthesis*, pages 300–305, 2006.

[12] A. Pinto, L. Carloni, and A. Sangiovanni-vincentelli. Constraint-drive communication synthesis. *ACM/IEEE Design Automation Conf.*, 2002.

[13] W. Shi and S. Chen. The rectilinear Steiner arborescence problem is NP-complete. *ACM-SIAM Symp. on Discrete Algorithms*, pages 780–787, 2000.

[14] R. Wang, N.-C. Chou, B. Salefski, and C.-K. Cheng. Low power gated bus synthesis using shortest-path Steiner graph for system-on-chip communications. *ACM/IEEE Design Automation Conf.*, 2009.

[15] D. West. *Introduction to Graph Theory.* Prentice Hall, 1999.

[16] Amba 2.0 specification. *http://www.arm.com/products /solutions/AMBA_Spec.html*, 1999.

[17] Amba 3 specification. *http://www.arm.com/products /solutions/axi_spec.html*, 2003.

[18] Coreconnect bus architecture. *IBM White Paper*, 1999.