A Hierarchical Three-Way Interconnect Architecture for Hexagonal Processors

Feng Zhou, Esther Y. Cheng, Bo Yao, Chung-Kuan Cheng, Ronald Graham

Department of Computer Science and Engineering
University of California, San Diego, La Jolla, CA 92093-0114

ABSTRACT

The problem of interconnect architecture arises when an array of processors needs to be integrated on one chip. With the deep sub-micron technology, devices become cheap while wires are expensive. On the other hand, high performance systems require the shortest communication routes among the processors. Non-blocking hierarchical interconnect architectures have been found to be a feasible solution. First, they can be expanded recursively and so can be applied in large-scale arrays. Second, if well designed, they have the best trade-off between the cost of wire resources and the communication performance. In this paper, a new type of non-blocking hierarchical three-way interconnect architecture, Y tree architecture, is put forward. We find that the arrays of hexagonal cells also have the property of hierarchical expansion, and we put an algorithm to build up a Y tree. We compare the Y architecture with an X hierarchical non-blocking architecture.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Interconnection architectures

General Terms

Design, Algorithms

Keywords

Y tree, Y architecture, interconnect architecture

1. INTRODUCTION

With the technology going to deep sub-micron, it becomes possible to integrate an array of processors on a single chip, and consequently the interconnect among the processor array turns out to be a key problem [1, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14].

SLIP'03, April 5–6, 2003, Monterey, California, USA.

In [2], Chen et al. proposed a Y architecture for on-chip interconnections, which has three symmetrical routing directions (0-degree, 60-degree, and 120-degree). They show that the Y architecture can gain a communication throughput improvement of 24 percent over the traditional Manhattan architecture on a squared mesh. This improvement is almost the same (2.6% less) with that of the X architecture [9], which needs one more routing direction. Furthermore, they show that if the hexagonal chip shape is adopted, the throughput improvement can be increased to 31 percent.

In [14], Shin proposed a hexagonal mesh for the interconnection of multiple processors in the system. The hexagonal mesh provides better communication performance and robustness than other topologies, including the square meshes.

In [3], Leiserson introduced a class of universal routing networks called fat-trees for the communication between the processors. The fat-tree network is constructed according to both the number of processors and the ammount of simultaneous communication requirements. A three-dimensional model which incorporates wiring and hardware size as costs is used to evaluate the performance.

In [11], Mai et al. utilized crossbars for local communication. Buses are used to link the crossbars for a second-level hierarchy and meshes are used for global communication. The optimizing target is the signal delay. In [12], Moritz presented a packet communication strategy, in which the area of the network and the memory for the packets are the optimized targets. In [13], Dally used a mesh to link an array of processors. The area and latency are used to evaluate the network.

In deep sub-micron technology, devices are shrunk to very small sizes and are not costly, while wires and buses are lengthened, resulting in the increase of wire resistance and capacitance. The performance of such parameters as power consumption and signal delay can be seriously degraded. So the length of signal path is much more important than the number of switches in the path. On the other hand, a large-scale system on the chip is hungry for the wire resources, so it is not feasible to set up the shortest path for every pair of processors in the array.

In [1], Cheng et al. proposed an optimizing function to balance the cost of wire resources and the power consumption for the interconnect topology on a processor array. The total wire length is used to measure the cost of the wire resources and the length of signal path is used to evaluate the power consumption, since the power consumption is proportional to the wire capacitance, which is further proportional to the distance traveled by the signal.

With this optimizing function, the X type non-blocking hierarchical interconnect architecture, which will be explained in Section 2, has been found to have the best trade-off for a two-dimensional processor array. One of the most important properties of X architecture is that it can be hierarchically expanded.

In this paper, we introduce a three-way non-blocking hierarchical interconnect architecture, with which to organize hexagon cells. We found that the hexagon cell array also has the property of hierarchical expansion. We summarized the principle for the expansion of Y architecture. We put forward an algorithm to set up the hierarchical tree of interconnect. We compare the Y architecture and the X architecture as the optimizing function is concerned. We also propose a representation of the polygons on the hexagonal backbone, which is useful in the study of large scale Y trees.

The remaining of paper is organized as follows. In Section 2 we introduce the objective function and the X architecture; In Section 3 we explore the properties of hexagonal arrays and the principle of setting up a Y tree; In Section 4 we prove that the Y architecture has the same values of objective function as the X architecture, and in Section 5 we present our algorithms for setting up a Y tree ; In Section 6 we give the representation for the boundary of the polygons on the hexagonal backbone. Finally, we draw our conclusions in Section 7.

2. BACKGROUND

In [1], an objective function is put forward to balance the cost of wire resources and the power consumption for the non-blocking interconnect architecture in an array of processors:

$$M = L * D$$

where,

$$L = \sum Length\ of\ each\ wire$$
$$D = \sum_{1 \le i < j < \le P} d_{i,j}$$

$d_{i,j}$ is the shortest route length between processor(i) and processor(j). P is the total number of processors. L represents the cost of wire resources and D reflects the power consumption resulting from the wire capacitance because $Power \propto D$. M should be as small as possible.

For rectangular cells, the X-type hierarchical architecture has been found to have the best performance [1]. As illustrated in Fig.1 (a), the small circle has the internal structure shown in Fig.1 (b), in which the 6 filled circles represent the switches that may connect the intersected buses. Every two of the four cells covered by the X interconnect architecture can set up a communicating route through the switch box. Meanwhile, the buses from the four cells are bundled together to form a new bus going to the higher level (Fig.1 (c)). The bigger circles lying at higher levels have the similar structure of the switch box except that the bus width grows 4 times for every expansion to a higher level. This architecture guarantees that whenever two processors in the array need to communicate, there always exists some route.

If the distance between the centers of the neighbor cells is 1, we have the following conclusions about the objective functions of X architectures. Here, n denotes the highest level of the X tree (Fig.1 (c)).

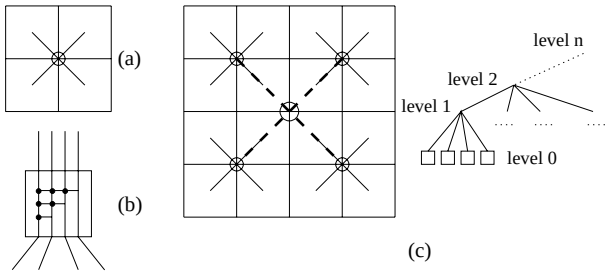

Figure 1: X-type hierarchical non-blocking architecture

Table 1: Recursive form solutions for L_x and D_x

L_X	D_X
$L_1 = 2\sqrt{2}$	$D_1 = 6\sqrt{2}$
$L_n = 4L_{n-1} + 2^{3n-2}\sqrt{2}$	$D_n = 4D_{n-1} + 6 \cdot 2^{4n-4}\sqrt{2}(2^n - 1)$

From the study of the X architecture, we can draw following desired properties for a hierarchical tree structure in an array of processors:

1. The tree grows recursively, and the topology of each level remains identical, with certain rotations allowed.

2. The total number of cells covered by the tree architecture is k^n, where k is the number of the sub-trees clustered on each level of the interconnect architecture.

3. The length of the trunk at level n, T_n is $\sqrt{k}T_{n-1}$.

4. There are no "holes", or "empty cells" within the array. In other words, the cells in the array form a "continuous" region bounded by a closed curve.

3. THE HIERARCHICAL ARCHITECTURE OF HEXAGONS

The hexagon processor array is shown in Fig.2, in which the processors have the physical layout of hexagons. We found that, like the X-type architectures, the hexagonal processor array can also be expanded hierarchically. Referring to Fig.2, every 3 cells can be clustered with a Y-type interconnect architecture, consisting the first level of the hierarchical tree. Then 3 first level sub-trees can also be clustered with a Y-type interconnect architecture, but the "Y" has a rotation of 90 degrees. By recursively doing this, a hierarchical Y architecture can be set up. Here the number of the sub-trees clustered on each level is 3, and the length of the trunk $T_n = \sqrt{3}T_{n-1}$. At the second level, we have 3^2 hexagon cells bounded by the dashed line Fig.2. There are no empty cells in the region.

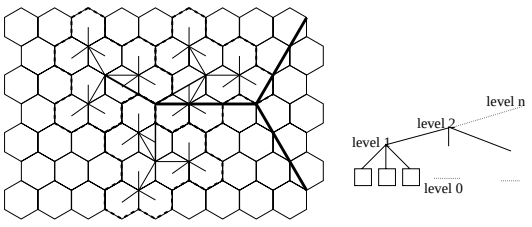

Figure 2: Y-type hierarchical architecture

Assuming that the distance between the centers of adja-

Table 2: Recursive form solutions for L_y and D_y

L_Y
$L_1 = \sqrt{3}a$
$L_n = 3L_{n-1} + 3^{\frac{3n}{2}-1}a$
D_Y
$D_1 = 2\sqrt{3}a$
$D_n = 3D_{n-1} + (3+\sqrt{3})(3^{\frac{n}{2}}-1)3^{2n-2}$a

cent cells is a, we summarize the key results of the Y architecture in Table 2. The derivation of the closed form solutions will be described in section 4.

4. Y ARCHITECTURE VERSUS X ARCHITECTURE

To make a fair comparison between X and Y architectures, we first assume that the cells in the two architectures have unit area. Thus the distance between the centers of adjacent cells in the X architecture is 1, and the distance between the centers of adjacent cells in the Y architecture, a in table 2, is $\sqrt{2}/\sqrt[4]{3}$.

From Table 1, we derive:

$$L_X(n) = \sqrt{2}(2^{3n-1} - 2^{2n-1})$$
$$D_X(n) = \frac{\sqrt{2}}{14}4^n(6 \cdot 2^{3n} - 7 \cdot 2^{2n} + 1)$$

From Table 2, we derive:

$$L_Y(n) = \frac{3^n(\sqrt{3}^n - 1)}{3-\sqrt{3}}a$$
$$D_Y(n) = \frac{3+\sqrt{3}}{78}3^n[(9+\sqrt{3})((3\sqrt{3})^n - 1) - 13(3^n - 1)]a$$

To make the comparison for larger n, we neglect the lower order terms of M_X and M_Y:

$$M_X(n) \approx \frac{3}{7}4^{4n}$$
$$M_Y(n) \approx \frac{11+7\sqrt{3}}{39}3^{4n}$$

n denotes the number of levels of an X or Y tree. Note that the number of cells, A, in the X and Y trees are:

$A_X = 4^n \quad A_Y = 3^n$

We normalize M_X and M_Y with A^4:

$$M_{Xnorm}(n) \approx \frac{3}{7} \approx 0.43$$
$$M_{Ynorm}(n) \approx \frac{11+7\sqrt{3}}{39} \approx 0.59$$

Similarly we normalize L and D with $A^{\frac{3}{2}}$ and $A^{\frac{5}{2}}$ respectively:

$$L_{Xnorm}(n) \approx \frac{\sqrt{2}}{2} \approx 0.71$$
$$L_{Ynorm}(n) \approx \frac{1}{3-\sqrt{3}}a \approx 0.85$$
$$D_{Xnorm}(n) \approx \frac{3\sqrt{2}}{7} \approx 0.61$$
$$D_{Ynorm}(n) \approx \frac{5+2\sqrt{3}}{13}a \approx 0.70$$

The derived results for M_X and M_Y show that the Y hierarchical architecture has a comparable performance with that of the X architecture.

5. THE GROWTH OF THE Y TREE

A hexagonal array (Fig.3) has the following properties:

Property 1: There exists a 1/2 grid shift between rows.

Property 2: Each cell is physically adjacent to two cells in the same row.

Property 3: Each cell is physically adjacent to two cells in the neighboring row above and two cells in the row below.

Thus we can cluster 3 neighboring cells to set up a Y interconnect, as illustrated with bold lines. Note that the "Y" can have two directions (Fig.3 (a) and (b)).

If we regard the clustered 3 cells as a unit, we can see that the array of such units also has the property of a 1/2 grid shift, but this time in the vertical direction. So we can expand the Y architecture to the second level; however, the directions of "Y"s at the second level must have a rotation of 90 degrees compared to the first level. It can be shown that the property of a 1/2 grid shift always holds when the Y architecture is recursively expanded to higher levels. Note that the direction of "Y" must have a rotation of ± 90 degrees when going to a higher level (Fig.2).

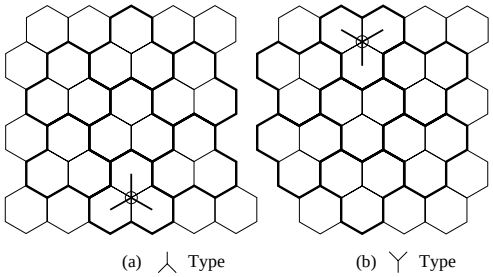

Figure 3: hexagonal processor array

Theorem 1: For a Y tree of n levels, there are 2^n combinations of the orientations of the n "Y"s on different tree levels.

We call such a combination of "Y"s a "configuration", which indicates the way the Y tree grows. Every configuration results in an particular boundary for the cells clustered by the Y tree. Fig.4 gives two examples of 6-level Y trees, together with their configurations.

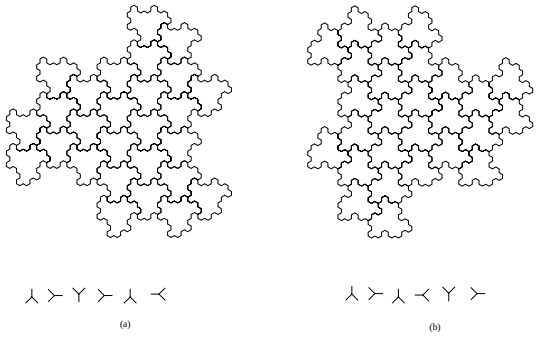

Figure 4: Two examples of 6 levels

Given a configuration C, a tree as shown in Fig.2 can be set up with the following expanding algorithm. The tree is

an ordered one. The first, second and third sub-trees correspond to 3 orientations as illustrated in Fig.5 (b). Every node in the tree, except the leaves, stores the orientation of the "Y" with which to organize its 3 sub-trees. $C[1]$ denotes the orientation of "Y" on the lowest level and $C[n]$ the highest level. $C[m] \in \{up, down, left, right\}$. Without the loss of generality, we always start the configuration with a inverted "Y". In other words, we assume $C[1] =$ "*down*". The coordinate shown in Fig.5(a) is used to distinguish the cells in the array. The filled circle in Fig.5 is always the center of the highest level "Y".

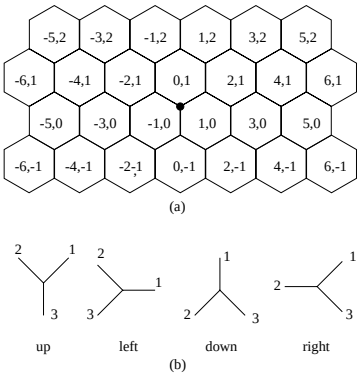

Figure 5: Coordinate and Sub-trees

```
Setup_Y_tree(C)
{
 sub-tree1 = Create_leaf(0,1);
 sub tree2 = Create_leaf(-1,0);
 sub-tree3 = Create_leaf(1,0);
 Tree=Compose_tree(sub-tree1,sub-tree2,
   sub-tree3,C(1));
 z = 2/3; x = 1, y = 1/3;
 for i=2:n
  {
   if (i is even)
    { z = z * 3; y = y * 3; }
   else
    x = x * 3;

   case C(i)
    "up":
     { x1= x; x2= -x; x3=  0;
       y1= y; y2=  y; y3= -z; }
    "left":
     { x1= z; x2= -x; x3= -x;
       y1= 0; y2=  y; y3= -y; }
    "down":
     { x1= 0; x2= -x; x3=  x;
       y1= z; y2= -y; y3= -y; }
    "right":
     { x1= x; x2= -z; x3=  x;
       y1= y; y2=  0; y3= -y; }

   Copy(sub-tree1,Tree);
   Copy(sub-tree2,Tree);
   Copy(sub-tree3,Tree);
  /* Copy Tree to sub-trees */
   Shift(sub-tree1, x1, y1);
   Shift(sub-tree2, x2, y2);
   Shift(sub-tree3, x3, y3);
  /* Shift the coordinates of every leaf in
     Tree by x(k) and y(k) respectively */
   Tree=Compose_tree(sub-tree1,sub-tree2,
     sub-tree3,C(i));
  }
}

Create_leaf(x,y)
{
 Create_tree_node(leaf);
 leaf->x = x;
 leaf->y = y;
 return(leaf);
}

Compose_tree(sub-tree1,sub-tree2,sub-tree3,
     orientation)
{
 Create_tree_node(new_root);
 new_root -> child1 = sub-tree1;
 new_root -> child2 = sub-tree2;
 new_root -> child3 = sub-tree3;
 new_root -> orientation = orientation;
 return(new_root);
}
```

Fig.6 gives an example for the algorithm. The configuration is "$down, left, up$". Fig.6 (a), (b) and (c) are the results for $i = 1, 2, 3$, respectively. Fig.6 (d) is the tree representation for Fig.6 (c), in which the coordinates in the leaves, from the most left to the most right, are:

(5,2)(4,1)(6,1)(2,3)(1,2)(3,2)(2,1)(1,0)(3,0)(-1,2)
(-2,1)(0,1)(-4,3)(-5,2)(-3,2)(-4,1)(-5,0)(-3,0)
(2,-1)(1,-2)(3,-2)(-1,0)(-2,-1)(0,-1)(-1,-2)(-2,-3)
(0,-3)

From the hexagonal array's properties and the algorithms for setting up the Y trees, we can draw the following conclusions:

Theorem 2: The proposed algorithm generates Y tree architecture without cell overlapping, and can cover all the cells without empty space.

Theorem 3: The number of cells covered by the generated Y tree of n levels is 3^n.

Theorem 4: the length of the trunk at level n is $(\frac{1}{\sqrt{3}})^n$.

6. THE BOUNDARY REPRESENTATION OF THE MERGED HEXAGONS

In this section, we first give the representation of polygons. Then, based on it, the algorithm for merging the polygons is presented.

6.1 The representation of polygons

Suppose we have a polygon merged by several hexagons, as illustrated in fig.7. We represent the polygon with a sequence of integers $i \in 0, 1$. We get the sequence of integers by traversing the boundary of the polygon counterclockwise. The boundary consists of a series of adjacent edges. We can see that every edge must have a rotation of either 120 degrees or -120 degrees with respect to its preceding edge. Then, we have the following definitions.

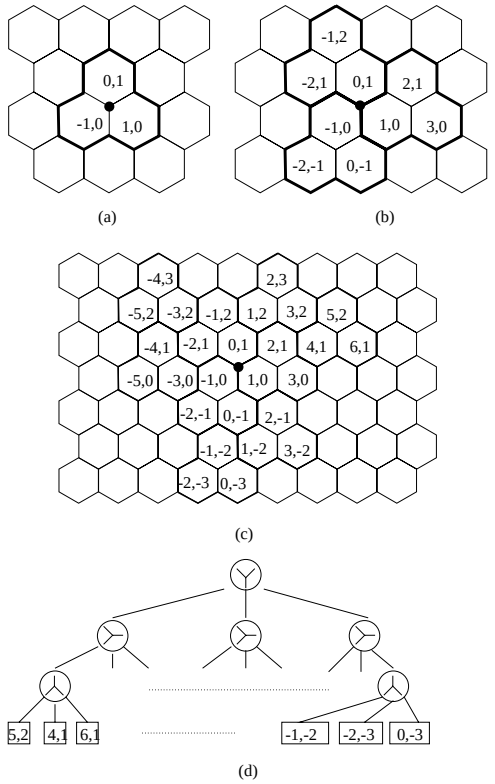

Figure 6: An example for "Setup_Y_tree"

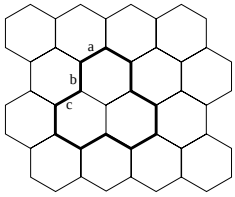

Figure 7: A polygon on the backbone of hexagons

Definition 1: If an edge A has a rotation of 120 degree relative to its preceding edge, we say the edge has a positive rotation; On the other hand, if the edge has a rotation of -120 degrees, we say the edge has a negative rotation.

For example, in Fig.7, edges a and b have positive rotations while edge c has a negative rotation.

Now we give the definition of the sequence to represent the polygon. We call such a sequence a hex-sequence.

Definition 2: We start from an edge on the boundary of a polygon and make a counter-clockwise traverse. If the edge has a positive rotation, we put a 1 into the sequence; If the edge has a negative rotation, we put a 0 into the sequence. Thus a hex-sequence is formed. If A denotes a hex-sequence, then $A(i)$ refers to the ith element in A, $A(i) \in 0, 1$.

For example, the hex-sequence of the polygon in Fig.7 is 110111011101, and the hex-sequence of the polygons in Fig.8 is 11011011101101. Now we do not impose any direction on the polygons and therefore regard the two polygons in Fig.8 the same.

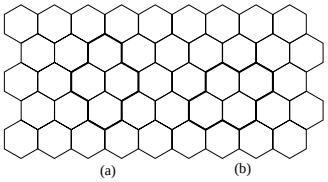

Figure 8: Two polygons with different orientations

Utilizing the geometry of hexagonal cells and the properties of hexagonal array, we deduce the following lemmas:

Lemma l: We can make any n bits barrel-shift (in this paper we always regard a barrel-shift to be leftwards) on a non-oriented hex-sequence without changing the corresponding polygon.

Lemma 2: For a legal hex-sequence, the number of '1's is just 6 more than the number of '0's; while for any subsequence, the difference between the number of '1's and the number of '0's should not exceed 5.

Lemma 3: Two polygons have the same shapes and areas iff they have the same hex-sequences.

Lemma 4: If we flip a polygon, then its hex-sequence should be horizontally flipped.

It is obvious that for a symmetric polygon, the application of Lemma 4 will keep the hex-sequence unchanged.

So far, we do not care about the orientation of the polygons. That means, the polygons can rotate and move freely. This property is very useful for the searching of optimized forests. However, rotations are not permitted for the generation of trees, so we give the definition for an oriented hex-sequence. Every edge on a polygon can have only 3 possible directions.

Definition 3: An oriented hex-sequence is denoted by starting the hex-sequence with a vertical edge that is traversed downwards.

Thus, the oriented hex-sequence for the polygon in Fig.8 (a) is 10110111011011, and the oriented hex-sequence for the polygon in Fig.8 (b) is 10111011011101. The directions of each edge can be easily calculated according to the numbers of '1's and '0's ahead of the edge.

Also we have the lemma for the oriented hex-sequences:

Lemma 5: We can make any i bits' barrel-shift on an oriented hex-sequence A without changing the direction of

the polygon iff the difference between the number of '1's and the number of '0's is either 0 or 5 in the sub-sequence from $A(2)$ to $A(i+1)$.

6.2 The polygon merging algorithm

The merging algorithm can be used to generate the polygons of the sub-trees in the Y tree. The polygon of a Y tree is the merging of the polygons of the three sub-trees in the next lower level.

Given two hex-sequences A and B, we use the following algorithm to get a new hex-sequence C, which is the merging of A and B. First we assume that both polygons A and B can be rotated.

```
Merge(A,B)
{
 /* Inverse every bit in sequence B */
 BI = Inverse(B);
 Bh = Horizontally_flip(BI);
 do
   {
    /* Find the sub-sequences in A and Bh such
       that the two sub-sequences have the same
       pattern */
    if(Find_next_match(A,Bh,*(&sub) == false);
      break;

    /* Judge if the result satisfies the
       requirements */
    if(Accept(A,Bh,sub == true)
     {

      /* Expand sub in Bh by 2 bits leftwards*/
      sub2 = Left_expand(sub);

      /* barrel shift Bh until sub lies at the
         most right end of Bh */
      BhR = adjust(Bh);

      D =  Bh - sub2;
      E =  Horizontally_flip(D);
      F =  Inverse(E);

      /* shift F 1 bit right-wards */
      G =  Right_shift(F);

      /* Inverse the two bits in A just
         adjacent to sub */
      AI = Inverse_sub_neighbor(A,sub);
      /* Replace sub in A with F */
      C = Replace(AI, G);
      Output(C);
     }
     if(Enough == true)
       break;
   }
}
```

Fig.9 gives an example to illustrate the algorithm. We will find the merging with the longest adjacent boundary.

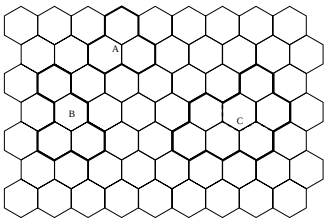

Figure 9: An example of merging

We have:

$$
\begin{aligned}
A &= 110111011101 \\
B &= 0011101110101111 \\
BI &= 1100010001010000 \\
Bh &= 0000101000100011 = 0001010001000110(Lemma1) \\
sub &= 110 \quad sub2 = 00110 \\
BhR &= 0001010001000\underline{110} \\
D &= 00010100010 \\
E &= 01000101000 \\
F &= 10111010111 \\
G &= 11011101011 \\
AI &= 1100\underline{110}01101 \\
C &= 11001101110101101101
\end{aligned}
$$

Now we consider the situation that the polygons are not allowed to rotate. In other words, we are merging two oriented polygons. The design of the function "Accept" should be more complicated. Not only should the two sub-sequences have the same pattern, but also have the same directions for their corresponding edges. In addition, if the adjacent boundary happens to involve the first bit of A, we have to remember the position of the first bit of B before merging, and shift the generated hex-sequence to make it correctly denote the polygon's orientation. It can be proved that the first bits of A and B will not appear in the sub-sequence simultaneously. Fig.10 and following results illustrate the merging of two oriented polygons.

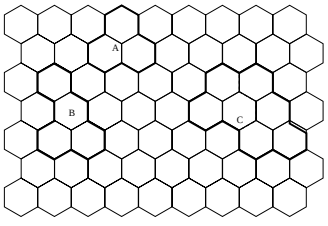

Figure 10: An example of oriented merging

$A = 1100111011101011$

$B = \hat{1}01110111011$//*remember the position of the first bit in B*

$BI = \hat{0}10001000100$

$Bh = 001000100010\hat{0}$

$sub = 100 \quad sub2 = 00100$

$BhR = 010001\hat{0}00\underline{100}$

$D = 0100010\hat{0}$

$E = \hat{0}100010$

$F = \hat{1}011101$

$G = 1\hat{1}01110$

$AI = 0\underline{100}011011101011$//*the boundary involves the first bit of A*

$C = 01\hat{1}01110011011101011$//*C needs an orientation adjustment*

$C = 10111001101110101101$

7. CONCLUSIONS

In this paper a new non-blocking hierarchical interconnect architecture, the Y architecture, is put forward. We derive Theorems 2 to 4 to show that the proposed algorithms can generate Y architecture recursively. We derive the normalized metric to measure the quality of Y architecture and X architecture.

8. ACKNOWLEDGEMENT

This work was supported in part under grants from NSF project number MIP-9987678, the California MICRO program and SRC support.

9. REFERENCES

[1] E.Y. Cheng, F. Zhou, B. Yao, CK Cheng, R. Graham, Balancing the Interconnect Topology for Arrays of Processors between Cost and Power, International Conference on Computer Design, Freiburg, Germany, Sept., 2002.

[2] H. Chen, B. Yao, F. Zhou, CK Cheng, The Y-Architecture: Yet Another On-Chip Interconnect Solution, Proceedings of the asia south pacific design automation conference, pp. 840-846. 2003.

[3] C.E. Leiserson, Fat Tree: Universal Networks for Hardware – Efficient Super-computing, IEEE Trans on Computers, pp. 892-901, Oct 1985.

[4] C.E. Leiserson, VLSI Theory and Parallel Super-computing, pp. 5-16. Advanced Research in VLSI, Proceeding of the Decennial Caltech Conference on VLSI, March 1989, edited by C.L. Seitz.

[5] V.E. Benes, Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, 1965.

[6] D. Knuth, The Art of Computer Programming, vol. 3, Sorting and Searchingi, Addison-Wesley, Reading, MA, 1973.

[7] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Kaufmann, 1992.

[8] C. Clos, A Study of Non-Blocking Switching Networks, Bell System Tech. vol. 32, pp. 406-424, 1953.

[9] M. Igarashi, T. Mitsuhash, et al., A diagonal-interconnect architecture and its applications to RISC core design, ISSCC Digest of Technical Papers, pp.210-211, San Francisco, 2002.

[10] R. Carden and C.K. Cheng, A Global Router Using An Efficient Approximate Multicommodity Multiterminal Flow Algorithm, ACM/IEEE Design Automation Conf., pp.316-321, July 1991.

[11] K. Mai, T. Paaske, N. Jayasena, R. Ho, W.J. Dally, M. Horowitz, Smart Memories, a modular re-configurable architecture, Proceedings of 27th International Symposium on Computer Architecture, pp.161-171, Vancouver, BC Canada, 2000.

[12] C.A. Moritz, D. Yeung, A. Agarwal, Exploring optimal cost-performance designs for Raw microprocessors, Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines, pp.12-27, Los Alamitos, CA, 1998.

[13] W.J. Dally, B. Towles, Route Packets Not Wires: On-chip Interconnection Networks, IEEE Proceedings of 38th Design Automation Conference, pp.684-689, 2001.

[14] K.G. Shin, HARTS: A Distributed Real-Time Architecture, IEEE Computer, May, pp. 25-35, 1991.