©2010 International Journal of Computer Applications (0975 - 8887)
Volume 1 – No. 21

A Generic Graph-Oriented Mapping Strategy for a Honeycomb Topology

Gaurav Kumar Singh, Mythri Alle, Keshavan Vardarajan, S K Nandy
Indian Institute of Science, Bangalore

Ranjani Narayan
Morphing Machines, Bangalore, India

ABSTRACT

REDEFINE [3] is a polymorphic ASIC, in which arbitrary computational structures on hardware are defined at runtime. The REDEFINE execution fabric comprises Compute Elements (CEs) interconnected by a Honeycomb network, which also serves as the distributed Network-on-chip. Each computational structure is dynamically assigned to a subset of the CEs on the execution fabric by the REDEFINE support logic. A HLL specification of the application is compiled into Hyper Operations (HyperOps) by the REDEFINE compiler [3], where each HyperOp is a set of interacting operations. The compiler also determines partitions of the HyperOps (pHyperOps) that can be assigned to CEs to suitably meet the structural constraints of the execution fabric. In this paper we propose an algorithm to map HyperOps onto Computational Structures. A pHyperOp communication graph (PCG) captures the communication between the various pHyperOps. Through a sequence of transformations, the PCG is transformed into a Cayley tree. The Cayley tree is then overlayed on the Cayley graph to form a computational structure. The proposed mapping algorithm offers a solution that incurs a penalty 18% on average over that of the optimal.

Categories and Subject Descriptors

G.2.2 [**Mathematics of Computing**]: DISCRETE MATHEMATICS—Graph Theory

General Terms

Algorithm

Keywords

Honeycomb, mapping, REDEFINE, Cayley graph, Cayley tree

1. INTRODUCTION AND MOTIVATION

On the architecture front, ever changing requirements of applications in terms of power, performance and throughput have demanded reconfigurable architectures in contrast to ASIC solutions due to prohibitive cost of ASICs. One such reconfigurable architecture REDEFINE by Alle et al. [3] serves a range of applications from a domain. This architecture comprises homogeneous set of Compute Elements (CEs) connected by a Network-on-chip (NoC). NoC is a novel approach to establish inter-chip communication based on data communication network. One of the influencing factors in energy consumption, latency and other performance parameters of NoC is topology. A special class of networks, called symmetric interconnection networks has been widely used for the purpose. The property of such network is that the network viewed from any vertex of the network looks the same. In such a network, congestion problems are minimized since the load is distributed uniformly through all vertices. Moreover, this symmetry allows for identical routers at every vertex with identical routing algorithms. It is also very useful in designing algorithms that exploit the structure of the network. In designing symmetric interconnection networks, the overall objective has been to construct large vertex symmetric graphs with small degree and diameter, high connectivity, and offering simple routing algorithms. Examples of this network are mesh, honeycomb, etc. Honeycomb has a lower degree per node than a 2-D Mesh [12]. This reduces the complexity and area of the network router. A detailed comparison of the honeycomb and mesh topologies is provided in [13]. The architecture REDEFINE in [3] uses Honeycomb topology as its underlying NoC.

REDEFINE [3,12] hardware resources, comprise Compute Elements (CEs) with local storage and routers that communicate over network on chip (NoC) as in Fig. 1. Resources are controlled by the Support Logic. The Support Logic comprises HyperOp Launcher (HL), Load Store Unit (LSU), Inter HyperOp Data Forwarder (IHDF), Hardware Resource Manager (HRM) and Resource Binder (RB). In [4], functional description of these modules is briefly provided. In REDEFINE, diverse data paths are composed in terms of computational structures at runtime. A computational structure is a physical aggregation of hardware resources that can perform a coarse grained operation, referred as a Hyper Operation (HyperOp). Compiler for REDEFINE, compile applications to an intermediate form and convert it into dataflow graphs [3]. These dataflow graphs are directed graphs of nodes where each node represents a HyperOp. Each HyperOp comprises multiple fine grained operations. In order to exploit parallelism that exists within a HyperOp (also due to storage limitation in a CE), each HyperOp is divided into several partitions (pHyperOp) and each pHyperOp is assigned a CE. Compiler captures the computation to be performed by each pHyperOp in terms of compute metadata and the inter/intra HyperOp communication in terms of transport metadata. The exact CEs to which HyperOps need to be loaded, is determined by the Resource Binder (RB). RB does this by, first maintaining a list of status (busy/idle) of all CEs and then by finding the appropriate place for each HyperOp. RB loads HyperOps in accordance to configuration matrix provided by compiler. Configuration matrix holds the information of relative positioning of nodes. Generation of configuration matrix requires the

embedding of nodes of PCG as per the architectural constraint of NoC. This embedding is needed primarily due to mismatch between the communication patterns among different modules of an application and the topology of the NoC. The problem hence translates to embedding an application represented by a source graph, G (that captures the communication patterns among modules of the application) onto a destination graph, H that depicts the topology of the NoC. In set theoretic definition, this is equivalent to assigning to each vertex u of G, a single vertex f(u) in H and each edge (u,v) of G to a path between f(u) and f(v) in H. This path may have intermediate vertices between f(u) and f(v). The general mapping problem is (unfortunately) an NP-hard combinatorial optimization problem, thus inherently intractable as shown by Johnson [2]. This necessitates the elaboration of efficient heuristics. Some of the earlier heuristics in Sarkar [9] and Gerasoulis [10] have used clustering of nodes in the source graph before physical mapping onto the target graph (in the case of source graph having more nodes than target graph).

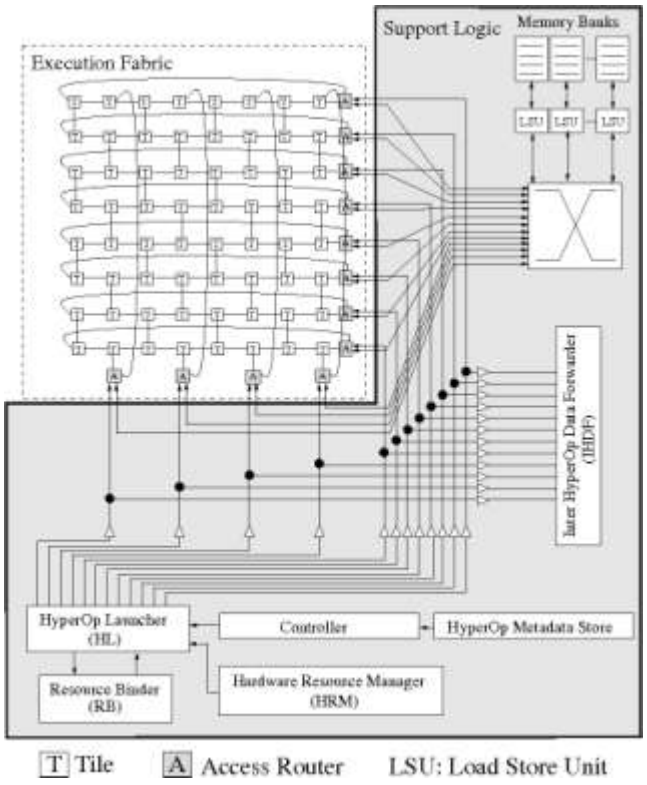

Figure 1: Architecture of REDEFINE

In this paper we present an embedding of a general graph into a honeycomb, with a constraint of one-to-one mapping of source graph to the target graph. Our embedding technique, called GOMAP (Graph Oriented Mapping) takes PCG generated by the compiler of REDEFINE [3] as source graph (limited to 16 nodes) and maps it onto the honeycomb network topology of the architecture. We also discuss the format of configuration matrix and processing of this matrix by RB. The remainder of this paper is as follows: Section 2 reviews the graph models for source graph, target (destination) graph as well as cost function for evaluating physical mapping. We have defined the case of optimality and cost overhead for mapping with respect to this cost function. We have also introduced the terminology of Cayley graph and Cayley tree in the section. Section 3 outlines the approach and steps of proposed algorithm. Section 4 gives an example on proposed method. In section 5 the experimental results of proposed algorithmic approach are presented in comparison with optimal in terms of communication overhead for honeycomb. In Section 6 we conclude our approach and give perspectives into future work.

2. GRAPH MODEL

Application of graph theory to finding mapping of a source graph to a destination graph is well known [8]. We will represent the source graph as $G_s(V_s, E_s)$ and the target graph as $G_d(V_d, E_d)$. We use the term "Hop" to indicate the unit distance between any two directly connected nodes in the target graph. The "distance" between two compute nodes u and v is the number of hops in the shortest path connecting u and v.

Definition 1: The source graph is an undirected graph, $G_s(V_s, E_s)$ where each vertex $v_i \in V_s$ represents a node and the edge (v_i, v_j), denoted as $e_{i,j} \in E_s$, represents the communication between v_i and v_j. The weight of the edge $e_{i,j}$ denoted by $comm_{i,j}$, represents the total number of communication between the two nodes.

An example of source graph is Fig. 2.

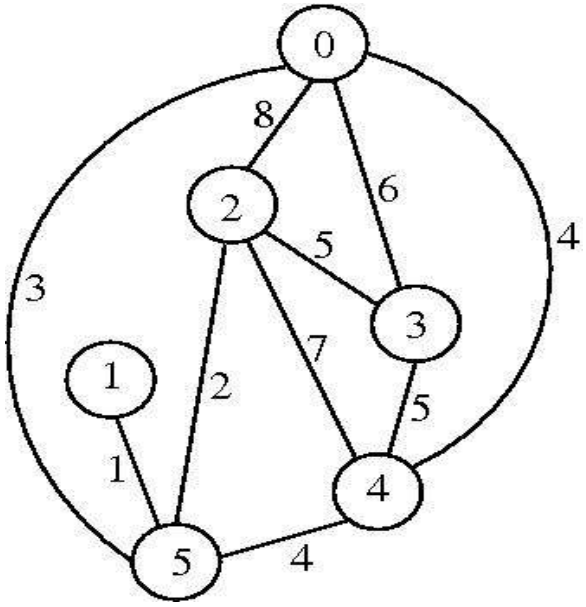

Figure 2: Communication graph of 6 Nodes

Definition 2: The target graph is an undirected graph $G_d(V_d, E_d)$ where each vertex $v_i \in V_d$ represents a node in topology and the direct edge (v_i, v_j) denoted as $e_{i,j}$ represents the hop distance of one, between the two nodes.

Source graph is a weighted graph, so we use adjacency matrix representation for weights. For example, adjacency matrix representation for weights of Fig. 2 is in Table 1.

Table 1: Adjacency-Matrix Representation for Weighted Graph

Node	0	1	2	3	4	5
0	**0**	**0**	**8**	**6**	**4**	**3**
1	**0**	**0**	**0**	**0**	**0**	**1**
2	**8**	**0**	**0**	**5**	**7**	**2**
3	**6**	**0**	**5**	**0**	**5**	**0**
4	**4**	**0**	**7**	**5**	**0**	**4**
5	**3**	**1**	**2**	**0**	**4**	**0**

2.1 Mapping Formulation

The mapping of the source graph $G_s(V_s, E_s)$ onto the target graph $G_d(V_d, E_d)$ is defined by an injection function

$$\text{map } \mathbf{f}_{map} : V_s \rightarrow V_d, \text{ s.t map } (v_s) = v_d, \forall\, v_s \in V_s, \exists! v_d \in V_d, \text{ where } |V_s| \leq |V_d|$$

2.2 Cost Function Formulation

To evaluate our technique, we define a Cost Function (CF), similar to the kind of cost function used in Koziris [5] and Zhonghai [6].

$$CF(\mathbf{f_{map}}) = \sum \mathbf{dist}(\mathbf{f_{map}}(u_s), \mathbf{f_{map}}(v_s)) \times \mathbf{comm}(u_s, v_s)$$

where:

- **dist**$(\mathbf{f_{map}}(u_s), \mathbf{f_{map}}(v_s))$ gives the shortest distance between the two nodes (u_d, v_d) in target graph onto which nodes (u_s, v_s) in the source graph are mapped.
- **comm**(u_s, v_s) is the weight of the edge $e(u_s, v_s)$ in the source graph.

Our objective is to find the mapping $\mathbf{f_{map}}$ which minimizes CF(f_{map}). We also define case of optimality for f_{map}(optimal) as :

$$CF(\mathbf{f}_{map}(\text{optimal})) = \min \{ CF(\mathbf{f}_{map}) \; \forall \; \mathbf{f}_{map} \in MAP_{ALL} \}$$

where MAP_{ALL} is the set of all possible mapping exist for this particular target graph

- **Cost Overhead** of mapping is defined in comparison to optimal as follows:

$$\text{Cost Overhead} = \frac{CF(f_{map}) - CF(f_{map}(\text{optimal})}{CF(f_{map}(\text{optimal})} \text{ X } 100\,\%$$

2.3 Cayley Graph and Cayley Tree

In section 1 we mentioned that our target graph is a honeycomb network, which is a symmetric interconnection network. From group theory perspective, honeycomb networks come under a special type of algebraic model, called Cayley graph (see Fig. 3). Krishnamurthy [1] discusses Cayley Graph and describe its properties in terms of degree, diameter, connectivity and routing algorithms.

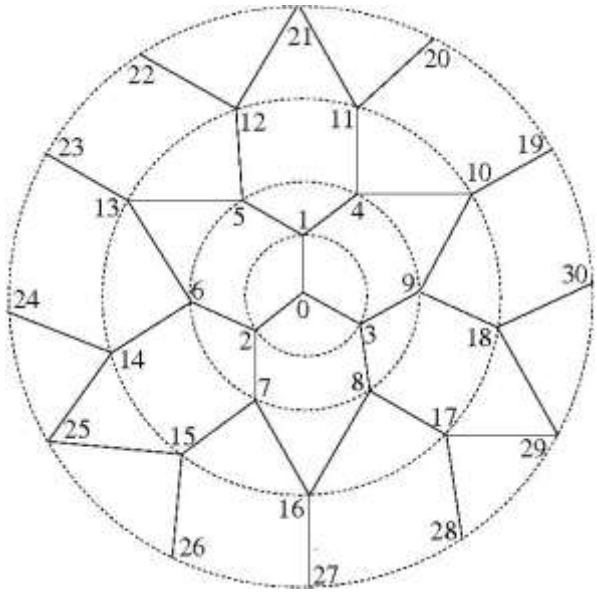

Figure 3: Honeycomb as Representation of Cayley Graph

Cayley tree is a connected cycle-free graph where each non-leaf node is connected to Z neighbors, where Z is called the coordination number. This can be viewed as a tree-like structure emanating from a central node, with all the nodes arranged in concentric shells around the central node. The central node is termed as root or origin. Ancestor and descendant relationships in the Cayley tree are defined relative to the root.

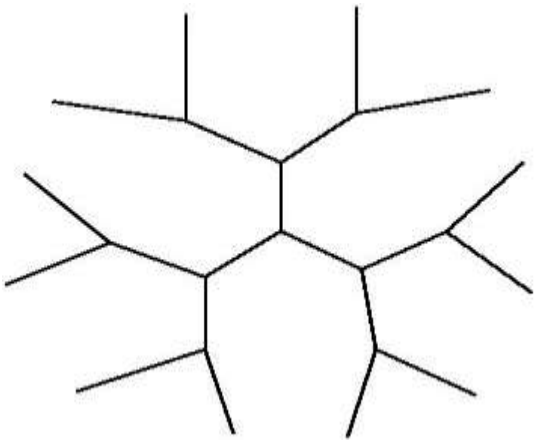

Figure 4: Cayley Tree (Bethe Lattice) of degree 3

The number of nodes in the k^{th} shell is given by,

$$N_k = Z(Z\text{-}1)^{k-1} \text{ for } k > 0$$

A star type Cayley tree of degree 3, also known as Bethe lattice [7] of coordination number 3 is shown in Fig. 4. Levels in Cayley graph and Cayley tree are defined with respect to root as concentric circles as shown by dotted circles in Fig 3 and Fig 5. For example, in Fig 3 node 0 is at level 0, nodes 1, 2, 3 are at level 1. The use of Cayley tree and Cayley graph is in section 3.

3. THE GRAPH ORIENTED MAPPING ALGORITHM (GOMAP)

Primary objective of our heuristic is to minimize the cost function. In order to do this, our approach is to place the nodes of the source graph as close as possible in the target graph. In other words, we start with the node which has maximum communication with its neighbors and map it to a vertex in the target graph. We then place nodes having direct communication with this node close to it (on the target graph), in the descending order of communication. We iteratively repeat this procedure for all the nodes that are already mapped onto the target graph and stop the procedure when all the nodes of the source graph are mapped onto the target graph. There could be conflicts arising during this procedure, which will be resolved in accordance with the steps described in this section.

As mentioned in section 2, honeycomb topology is a Cayley graph of degree 3 and is a graph having the following property: i) root node (node 0 in Fig. 3) at center with three edges emanating from the it, ii) for all nodes, that are independent (only one immediate ancestor) children (node 1,2), two edges emanate from that node, iii) for all other nodes that are shared (two immediate ancestors) children (node 10,13), only one edge emanate from that node, iv) there exists cycles of length 6 nodes and every node of a cycle is part of two other different cycles of length 6.

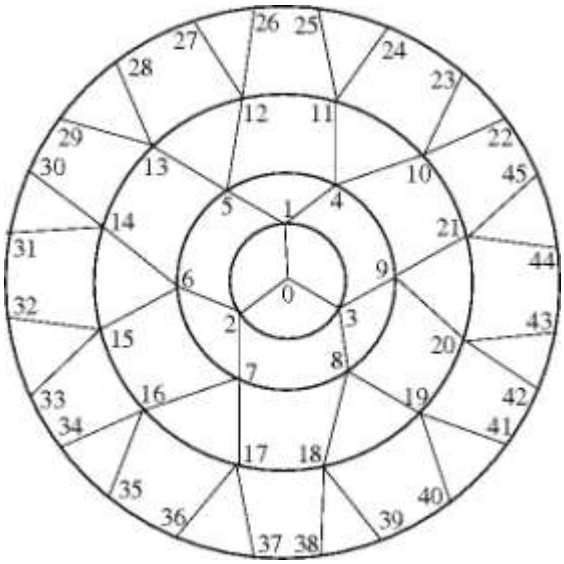

Figure 5: Cayley Tree Representation from a Cayley Graph Perspective

Property (i) and (ii), gives the basis for construction of Cayley tree of degree 3. From this we deduce that, in a Cayley graph, if we relax property (iii) and (iv) among nodes, then Cayley tree data structure can be applied for the purpose of mapping onto Cayley graph. Refer Fig. 5 for an illustration of the same. With this basis we go in detail of our mapping formulation. We will discuss relaxation of (iii) and (iv) later (3.2) in this section. Approach is to start from the source graph and construct an intermediate representation as close a Cayley tree as possible. We term this intermediate representation as MAP-Tree, which is constructed incrementally by best-fitting two highly communicating nodes (in the source graph) as close as possible in the target graph. Our mapping algorithm comprises three phases. In the first phase, MAP-Tree is constructed. In the second phase, conflicts arise in MAP-Tree, due to relaxation of property (iii) and (iv) of Cayley graph, are resolved and in the final phase, mapping information is generated and physical placement of compute nodes is done.

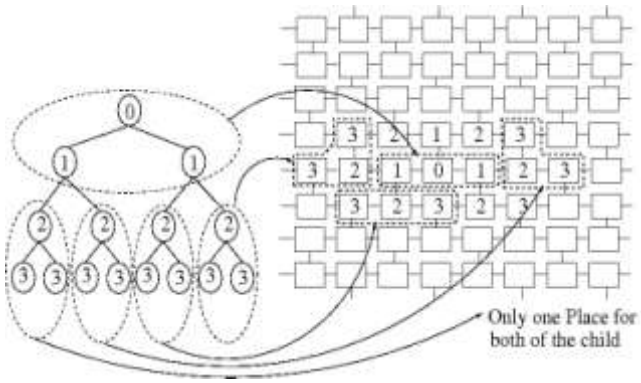

Figure 6: Conflict Scenario

3.1 First Phase: MAP-Tree Construction

Initially there is no mapping on NoC architecture tiles, and with nodes mapping, the possibility of better mapping would be decreased, nodes with higher $comm_{i,j}$ should be mapped earlier. At first the root with the highest priority for MAP-Tree is determined by the function highest-priority(v_j).

highest-priority(v_j) = $max_j(comm(v_i,v_j))$, $\forall\, v_i \in V_s$

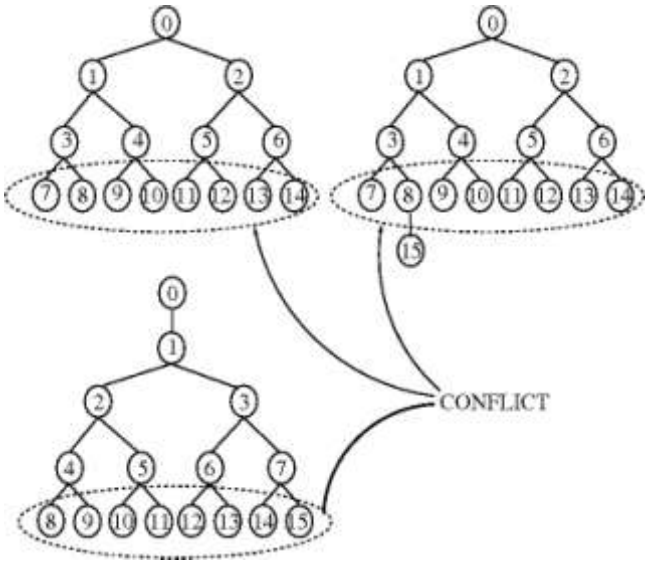

Figure 7: Different Conflict Cases of MAP-Tree

Then the function finds the highest $comm_{i,j}$, between the source (root node in MAP-Tree) and its edge destination. The node with higher ranking(v_j) will be selected as the first choice.

$$Ranking(v_j) = \max_j (comm(v_i,v_j)), \forall v_i \in \text{MAP-Tree and } \forall v_j \notin \text{MAP-Tree}$$

We iteratively calculate the Ranking of each v_j ($\in V_s$) and not in MAP-Tree with respect to the nodes $v_i \in$ MAP-Tree and incrementally add the selected v_j in MAP-Tree with the edge(v_i,v_j). If degree of v_i is 3, we select the descendants(k,m) of v_i and compute sum of remaining communications, which have not been considered for MAP-Tree, individually for both k and m. Select node with lower sum under constraint degree[selected node] < 3. If constraint not fulfilled, repeat for other node. If none applies iterate this until we find one descendant node (say p). Then we add v_j with edge(p,v_j). We repeat this for each node v_j in source graph $G_s(V_s,E_s)$ that are not in MAP-Tree. Pseudo code for MAP-Tree construction is given in Algorithm 1.

```
function MAPTreeConstruction(Gs(Vs, Es),T(V,E) )
/* Root Selection */
root ← highest-prioirty(vs∈ Vs);
/* Include Root in MAP-Tree */
V ← V ∪ {root};
Vs ← Vs - V;
/* Find highest communicative between root and all it
   descendants */
for Each descendant vs of root in Gs(Vs, Es) do
   Ranking(vs) = comm(root,vs) ∀ vs ∈ Vs
endfor
/* Select the highest ranked node. Add it to MAP-Tree
*/
vhrank ← highest{ Ranking(vs) };
Vs ← Vs - vhrank;
V ← V U{ vhrank};
E ← edge(root, vhrank);
while Vs != φ do
   foreach descendant(vi) of VMAP–Tree ∈ V in Gs(Vs,Es) do
      Ranking(vi) = comm(VMAP–Tree,vi) ∀ vi ∈ Vs
   endfch
   /* Select the highest ranked node. Add it to
      MAP-Tree */
   vhrank ← highest{ Ranking(vi ) };
   Vs ← Vs - vhrank;
   V ← V U {vhrank};
   vsource ← (node in MAP-Tree that has highest communication
   with vhrank);
   if degree[vsource]<3 then
   E ← edge(vsource, vhrank);
   endif
   Select vsource as the descendant of vMAP–Tree with
   min{remaining highest communication};
   if degree[vsource] = 3 then
   vsource ← other descendant of vMAP–Tree;
   endif
   if degree[vsource] = 3 then
   /* do selection recursively */
   Endif
   E ← edge(vsource, vhrank);
Endw
```

Algorithm 1: MAP-Tree Construction

3.2 Second Phase : MAP-Tree Conflict Check and Resolution

After constructing MAP-Tree as aforesaid methodology, we get MAP-Tree similar to Fig. 5. This MAP-Tree doesn't ensure an embedding for the target graph. The "conflict" could occur in this MAP-Tree as shown in Fig. 6. Conflict is the case, where two or more nodes of MAP-Tree contend for the same node of NoC. The sole basis of conflict is: consideration of independent child node in MAP-Tree construction with contradiction to shared child node in target graph due to relaxation of constraint (iii) and (iv) as in section (3.1). Difference in these two gets to appear only after, if MAP-Tree has atleast 15 nodes (see Fig 6). With lesser than 15, in practice we can always map it to target graph without any change in MAP-Tree. Fig. 7 shows three different conflict cases. All the conflicts only occur either at level 3 or greater (see Fig. 3), considering root as level 0 and always occurs when any 4 of the nodes at level 2 has two descendants each. This is because before level 3 there are no shared nodes in target graph (see Fig. 3), and at level 3, two of these eight children (nodes 10-17 in Fig. 5) get conflicted for allocation on target graph (nodes 10-16 in Fig. 3), as we have only 7 nodes in target graph to get mapped as shown in Fig. 6. Pseudo code for conflict detection is given in Algorithm 2.

Resolution : Conflict occurs because of mapping 8 nodes in 7 positions available, so conflict resolution has been made by moving "one" among conflicted node as a descendant to it's parent descendants. Least communicative node has been chosen and associated as a descendant, of its parent (immediate ancestor) immediate descendant. The new MAP-Tree will be conflict free as shown in Fig. 8 and ready to place. Pseudo code for conflict resolution is given in Algorithm 3.

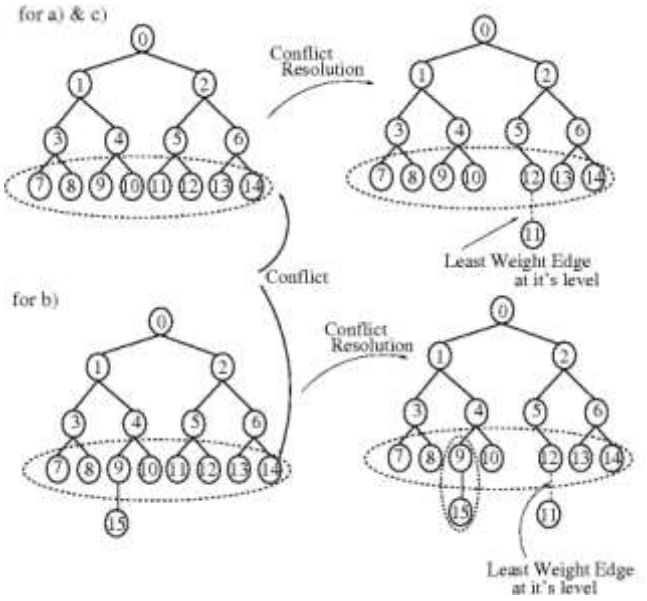

Figure 8: Conflict Resolution of MAP-Tree

3.3 Final Phase : Resource Matrix Generation and Physical Placement

The MAP-Tree is placed on target graph, considering the order of children from root to leaf nodes. After placing all nodes we need to send the mapping information as configuration matrix to the module

RB of Support Logic for physical placement. This step is needed, because as per the configuration matrix, RB finds the suitable position (CE) for each node to place it on NoC. Mapping

```
function Conflict(T(V,E))
/* Returns TRUE if there is conflict in MAP-Tree otherwise FALSE */
|V| = no. of nodes in MAP-Tree
if |V| < 15 then
    return FALSE;
else
    switch degree[root] do
        case 1
            Run BreadthFirstSearch on T(V,E);
            Node3 ← all nodes at level 3 with degree 3 ;
            if |Node3| < 4 then
                return FALSE;
            else
                return TRUE;
            endif
            break;
        case 2
        case 3
            Node2 ← all nodes at level 2 with degree 3 ;
                if |Node2| < 4 then
                    return FALSE;
                else
                    return TRUE;
                endif
                 break;
        endsw
        otherwise
            ;
    endsw
endif
```

Algorithm 2: Conflict in MAP-Tree

```
function ConflictResolution(T(V,E))
/* Move the "one" among conflicted one level Down */
level ← Level at which Conflict(T(V,E)) is TRUE;
foreach edge(node[level],descendant[node[level]]) in E do
    Find edge w(node,descendant) with minimum communication;
    y ← descendant;
    E ← E - w;
    E ← edge(descendant[node],y);
endfch
```

Algorithm 3: Conflict Resolution of MAP-Tree

information can be passed to RB in different ways. One way of representing this information is binary matrix structure (0/1-Matrix). Binary matrices consist of 1 and 0 as element values. In architecture [3], 1's are the relative position of nodes which need to get placed on NoC in exactly same fashion with respect to each other and 0 as relative positions, which are don't care (irrespective of CEs status) place in NoC. Don't care position assumes that RB need not pay attention to the availability status of relative CEs on NoC while making decision for physical placements of nodes. Resource Binder gives the start position (0,0) of configuration matrix in the NoC, after comparing free/busy CEs topology of NoC to binary matrix structure. Design criteria for RB includes variable sized matrix versus fix sized matrix. Variable sized matrix requires lesser memory to be stored, because it uses minimal number of elements to capture all 1's information, whereas fixed size matrix (dimension k) always use the same number of elements ($k \times k$) to do this, thus giving uniformity and ease of implementation of Resource Binder. It can be shown, that configuration matrices fit in a matrix of dimension 6×4 to map a pHyperOp communication graph upto 16 nodes onto honeycomb NoC.

4. EXAMPLE

Let take $G_e(V_e, E_e)$ with communication cost as shown in Fig. 9.

.

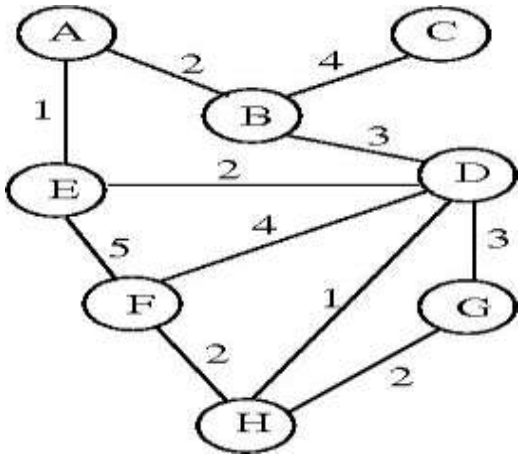

Figure 9: Source Graph

Table 2: Sorted table of Total Communication For Each Node

Node Number	Total communication	Ranking
A	**3**	**8**
B	**9**	**3**
C	**4**	**7**
D	**10**	**2**
E	**8**	**4**
F	**11**	**1**
G	**5**	**6**
H	**5**	**5**

4.1 MAP-Tree Construction

initialize: Sets INCLUDED = φ and NOTINCLUDED = {A, B, C, D, E, F, G, H}

step1 (Root Selection): Root is decided as per the Table 2, node with highest ranking. F is selected as root.

step2: Now two sets INCLUDED = {F} and NOTINCLUDED =

{A, B, C, D, E, G, H }

1st iteration :
step3: Of the two sets, highest communicated edge is selected. adjacent{F} = {D,E,H}
Select edge = max{F-D,F-E,F-H} = F − E
Node = {E}
INCLUDE = {F, E} and NOTINCLUDED = {A, B, C, D, G, H }
step4: Adding edge F − E

2nd iteration :
step3: adjacent{F, E} = {D, H, A}
Select edge = max{F − D, F − H, E − D, E − A} = F − D
Node = {D}
INCLUDED = {F, E, D} and NOTINCLUDED = {A, B, C, G, H }
step4: Adding edge F − D

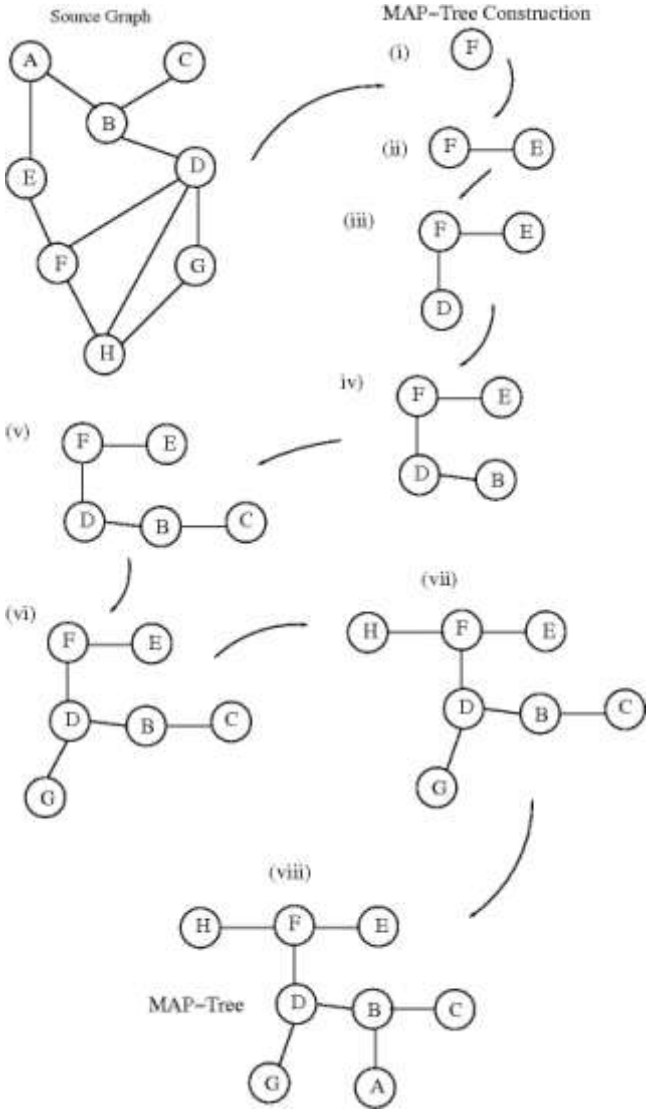

Figure 10: MAP-Tree Construction from Source Graph

3rd iteration :
step3: adjacent{F, E, D} = {H, A, B, G}
Select edge = max{F − H, E − A, D − B, D − G, D − H} = D − B
Node = {B}
INCLUDED = {F, E, D, B} and NOTINCLUDED = {A, C, G, H }
step4: Adding edge D − B

4th iteration :
step3: adjacent{F, E, D, B} = {H, A, G, C }
Select edge = max{F −H, E−A, D−G, D−H, B−C, B−A} = B − C
Node = {C }
INCLUDED = {F, E, D, B, C } and NOTINCLUDED = {A, G, H }
step4: Adding edge B − C

5th iteration :
step3: adjacent{F, E, D, B, C } = {H, A, G}
Select edge = max{F − H, E − A, D − G, D − H, B − A} =

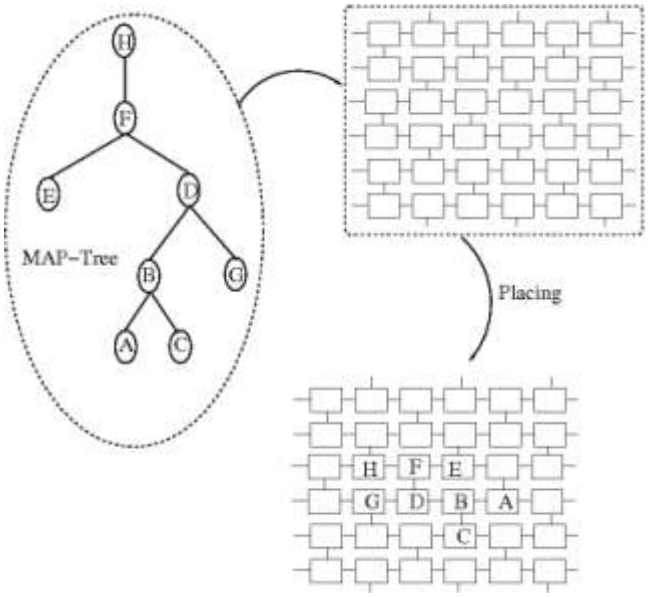

Figure 11: MAP-Tree with Placing on HoneyComb

D − G
Node = {G}
INCLUDED = {F, E, D, B, C, G} and NOTINCLUDED = {A, H }
step4: Adding edge D − G

6th iteration :
step3: adjacent{F, E, D, B, C, G} = {A, H }
Select edge = max{F − H, E − A, D − H, B − A, G − H } = F − H
Node = {H }
INCLUDED = {F, E, D, B, C, G, H } and NOTINCLUDED = {A}
step4: Adding edge F − H

7th iteration :
step3: adjacent{F, E, D, B, C, G, H } = {A}
Select edge = max{E − A, B − A} = B − A
Node = {A}
INCLUDED = {F, E, D, B, C, G, H, A} and NOTINCLUDED = { }
step4: Adding edge B − A

Set NOTINCLUDED is empty, MAP-Tree is done. MAP-Tree construction is shown in Figure 10. MAP-Tree looks like Cayley tree as we see in Fig 11. Configuration Matrix for MAP-Tree is in Table 3.

4.2 MAP-Tree Conflict Check and Resolution

Numbers of nodes are less than 15, so MAP-Tree is conflict free.

Table 3: Configuration Matrix

N	0	1	2	3	4	5
0	0	1	1	1	0	0
1	0	1	1	1	1	0
2	0	0	0	1	0	0
3	0	0	0	0	0	0

4.3 Resource Matrix generation and Physical Placement

MAP-Tree is Placed as described in section 3 to generate the mapping information. Root is placed at any location. Childs are placed with respect to root. Subsequently, children are placed with left and right consideration. After mapping we get the configuration matrix as in TABLE 3.

The physical placement of configuration matrix on execution fabric is shown in Fig 11. The overall cost for this mapping is 36, whereas optimal cost is 33.

5. RESULTS

We run the GOMAP algorithm for graphs generated by REDEFINE [3] compiler for two applications FFT and IDCT. For FFT, the graph size ranged upto 7 nodes while for IDCT it reached upto 16 nodes. We also implemented Optimal mapping for graph size limit upto 16 for honeycomb network for the purpose of comparison. Graph size higher than this was not feasible as well as not practical for the purpose of comparison.

As per our knowledge, there exists no mapping algorithm specifically meant for honeycomb, which keeps our comparison limited to the case of Optimality. PCG (pHyperOp communication graph) which we used as source graph is random while the target graph is honeycomb topology. For FFT we incurred cost overhead upto 2%, because all the graph (PCG) sizes were of small size (upto 7 nodes only). For IDCT we incurred on average 18% cost overhead.

6. CONCLUSION

In this paper we have reported a polynomial time heuristic to map a source graph onto a target graph in general. The source graph is an intermediate representation of the application transformed by a compiler for a runtime reconfigurable SoC. The target graph represents the Compute Elements and their interconnections provided by the NoC. The nodes of the source graph represent the computations and the edges denote the require communication between them. We show that the proposed scheme incurs on average a cost overhead of 18% in terms of performance when compared to an optimal mapping.

7. REFERENCES

[1] B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks, IEEE Trans. on Computers, vol. 38, pp. 555-566, 1989

[2] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H Freema and Co, 1979

[3] Mythri Alle, Keshavan Varadarajan, Alexander Fell, Ramesh Reddy C, Nimmy Joseph, Saptarsi Das, Prasenjit Biswas, Jugantor Chetia, Adarsh Rao, S K Nandy. REDEFINE: Runtime Reconfigurable Polymorphic Asic, IEEE Transaction On Embedded Computing Systems, Special Issue on configuring Algorithms, Process and Architecture, 2008

[4] Alexander Fell, Mythri Alle, Keshavan Varadarajan, Saptarsi Das, Prasenjit Biswas, Jugantor Chetia, S K Nandy. Streaming FFT on REDEFINE-v2: An Application Architecture Design Space Exploration, to be appeared in CASES '09, 2009

[5] Koziris N., Romesis M., Papakonstantinou G. and Tsanakas P. An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures, Proceedings PDP 2000 Conference, pp. 406-413, Rhodes, 2000

[6] Zhonghai Lu, Lei Xia, Axel Jantsch. Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip, ddecs, pp. 1-6, 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008

[7] M. E. Fisher , J. W. Essam. Rev. Mod. Phys. 42:271, 1970

[8] W.K. Chen and E. Gehringer. A graph-oriented mapping strategy for a hypercube, Proc. Third Conference on Hypercube Concurrent Computers and Applications. pp. 200-209, 1988

[9] Sarkar V. Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors, Cambridge, MA, MIT Press, 1989.

[10] Yang T. and Gerasoulis A. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors, IEEE Transactions on Parallel and Distributed Systems,Vol.5, No.9, pp. 951-967, 1994

[11] Mythri Alle et al. Synthesis of Application Accelerators on Runtime Reconfigurable Hardware. In ASAP '08, Proceedings of the 19th IEEE International Conference on Application specific Systems, Architectures and Processors, July 2008

[12] A. N. Satrawala et al. Redefine: Architecture of a SOC Fabric for Runtime Composition of Computation Structures, In FPL '07: Proceedings of the international Conference on Field Programmable Logic and Applications, August 2007

[13] I. Stojmenovic. Honeycomb Networks: Topological properties and Communication Algorithms. In IEEE Trans, Parallel and Distributed Systems, vol. 8, no. 10, pp. 1036-1042, Oct. 1997.