# A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures*

Krishnan Srinivasan and Karam S. Chatha
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ, 85287
{ksrini,kchatha}@asu.edu

## ABSTRACT

Network-on-chip (NoC) has been proposed as a solution for the global communication challenges of System-on-chip (SoC) design in the nanoscale technologies. NoC design with mesh based topologies requires mapping of cores to router ports, and routing of traffic traces such that the bandwidth and latency constraints are satisfied. We present a novel automated design technique that solves the mesh based NoC design problem with an objective of minimizing the communication energy. In contrast to existing research that only take bandwidth constraints as inputs, our technique solves the NoC design problem in the presence of bandwidth as well as latency constraints. We compare our technique with a recent work called NMAP and an optimal MILP based formulation. We prove that the complexity of our technique is lower than that of NMAP. For the latency constrained case, while NMAP fails on most test cases, our technique is able to generate high quality results. In comparison to the MILP formulation, the results produced by our technique are within 14 % of the optimal.

## Categories and Subject Descriptors

B.4 [**Input/Output Data Communications**]: Interconnections

## General Terms

Algorithm, Performance, Design

## Keywords

Network-on-Chip, Automated design, Mesh topology, Core mapping, Routing

## 1. INTRODUCTION

The advent of deep sub-micron technology will pose many challenges for next generation high end System-on-Chip (SoC) design.

*The research presented in this paper was supported in part by a grant from the National Science Foundation (IIS-0308268) and Consortium for Embedded Systems.


*ISLPED'05,* August 8–10, 2005, San Diego, California, USA


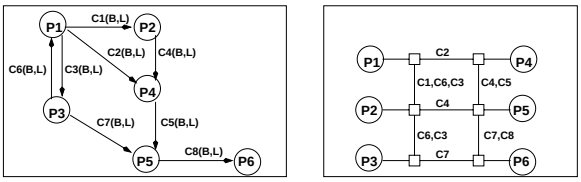


**Figure 1: Low energy mapping and routing**

In the future a SoC architecture is expected to consist of tens of computing cores operating in the multi-gigahertz range. The various cores would require a communication medium that can support simultaneous high bandwidth data transfers with low latencies. Current day bus based shared medium architectures will not be suitable as they would have to be implemented as hierarchical structures extending to multiple levels. The high signal propagation delays in deep sub-micron technologies will also make synchronous bus based global communication difficult. Noise due to increased RLC effects in deep sub-micron technologies will lead to signal integrity issues which also cannot be easily addressed by bus based architectures.

On-chip packet switched interconnection architectures or Network-on-Chip (NoC) have been proposed as a solution for the communication challenges in the nanoscale regime [1]. NoC is characterized by asynchronous communication between routers. NoC is inherently scalable and can be easily applied towards the design of larger sized SoC architectures. NoC supports easier application of error control schemes for increased signal integrity. In particular mesh based NoC architectures are especially attractive due to their regular two dimensional structure that results in IP re-use, easier layout, and predictable electrical properties. Consequently, in recent years a number of researchers have proposed architectures and tools for mesh based NoC [2, 3, 4].

NoC design for an application specific SoC architecture offers an opportunity for optimizing the mapping of cores to different routers, and incorporation of custom routing of the packets that do not necessarily conform to a pre-determined routing scheme. Periodic high performance applications with deadlines enforce bandwidth and latency requirements for data transfer on the communication medium. Further, each router also places an upper bound on the bandwidth of traffic that can supported at every input/output port. In the nanoscale technologies energy minimization has emerged as a first order design goal. The global communication energy is expected to account for a significant portion of the total energy consumption [1]. Therefore, the objective of the interconnection design is to obtain an implementation that satisfies the performance requirements and minimizes the communication energy. The paper addresses automated NoC design for mesh based interconnection architectures.

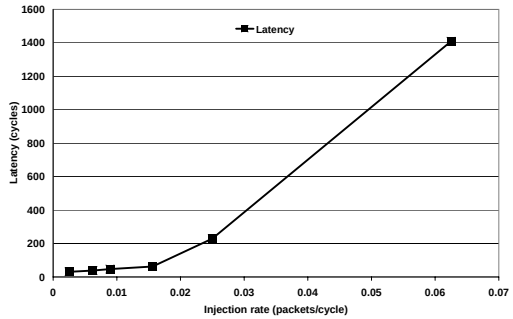


**Figure 2: Latency versus injection rate**

The NoC design problem on mesh based interconnection networks is depicted in Figure 1. The input to the problem is a directed graph called the communication trace graph. Each node in the graph represents a computation or storage core, and the directed edges represent communication between the cores. Each communicating trace is annotated as "C*m*(B,L)" where '*m*" represents the trace number, "B" represents the bandwidth requirement, and "L" is the latency constraint. The bandwidth and latency requirement on the communication traces can be easily obtained from the desired performance of the overall application and the individual task latencies of each node. The output of the NoC design problem is a mapping of cores onto different routers, a corresponding mesh based NoC, and a static packet route for each traffic trace such that the total communication energy is minimized. On the right hand side of Figure 1, the static routing of a communication trace is shown by the corresponding annotation of physical links.

We characterized the energy consumption of the unit router in $100nm$ technology with the help of a cycle accurate energy and performance evaluator [3]. In the interest of space, we have omitted the details of the experiments. We observed that over time, the energy consumption of the input and output ports varied linearly with the injection and acceptance rates, respectively. Quantitatively, we estimated the energy consumption of $2.07pJ/Mbps$ for the input port, and $2.29pJ/Mbps$ for the output port. The variation of average latency on a port with respect to injection rate is shown in Figure 2. We observe from the plot that average latency remains almost constant in the un-congested mode, and the onset of congestion is marked by a sharp increase in latency. Our technique prevents network congestion by static routing of the communication traces subject to the peak bandwidth constraint on the router ports. As the network is always operated in the un-congested mode, we can represent the network latency constraint as router hops (such as 1 or 2) instead of an absolute number (such as 60 cycles).

**Problem Definition :** We define the mesh based NoC design problem as follows. Given:

- A directed communication trace graph $G(V, E)$, where each $v_i \in V$ denotes either a processing element or a memory unit (henceforth called a node), and the directed edge $e_k = (v_i, v_j) \in E$ denotes a communication trace from $v_i$ to $v_j$.
- For every $e_k = \{v_i, v_j\} \in E$, $\omega(e_k)$ denotes the bandwidth requirement in bits per second, and $\sigma(e_k)$ denotes the latency constraint in hops.
- A mesh based topology of NoC $I(N, L)$, where each $n_i \in N$ denotes a router, and each $l_i \in L$ denotes a physical link. All routers are identical 5-port routers with 4 ports connected to neighboring routers via links and one open port for node mapping.
- $I$ is placed on a grid in the XY plane with unit distances between adjacent routers. $x(n_i)$ and $y(n_i)$ denote the x and y coordinates of a router $n_i \in N$.
- Each router architecture is characterized by:
  - $\Omega$ which denotes the peak input and output bandwidth that the router can support on any one port,
  - $\Psi_i$ that denotes the energy consumed per $Mbps$ of traffic bandwidth flowing in the input direction for any port of the router, and
  - $\Psi_o$ which denotes the energy consumed per $Mbps$ of traffic bandwidth flowing in the output direction for any port of the router.

The objective of the NoC mapping and routing problem is to obtain:

- a one to one mapping function $\mathcal{M} : V \rightarrow I$ that denotes the mapping of a node to a router,
- a set $\mathcal{R}$ of ordered tuples of routers, where each $r_i\langle n_i, n_j, \ldots, n_k\rangle \in \mathcal{R}, n_i, \ldots, n_k \in I$ denotes a route for a trace $e(v_i, v_k) \in E$ ($\mathcal{M}(v_i) = n_i$, $\mathcal{M}(v_k) = n_k$),

such that

- the bandwidth constraints on router ports are satisfied,
- the bandwidth and latency constraints on the traces are satisfied, and
- the total communication energy is minimized.

As mentioned earlier, the energy consumption of the NoC in the un-congested mode varies linearly with the traffic flowing through the network. Therefore, the energy consumption of the NoC can be minimized by minimizing the cumulative traffic flowing through the ports of all routers.

Bandwidth requirements and latency constraints of communication traces can be viewed as mutually independent. A trace such as a signalling event or a cache miss is not expected to have a high bandwidth requirement, but is bound by tight latency constraints. On the other hand, many non-critical multimedia streams have high bandwidth requirement, and their latency is bound only by the period constraint of the application. A mapping and routing technique has to perform a trade-off between placing high bandwidth traffic traces close to each other to minimize energy, and placing tight latency traces close to satisfy the performance constraints.

In this paper, we present a novel two phase technique that effectively performs energy versus latency trade-off in stage 1 to obtain a mapping of cores on the mesh based NoC, and then generates a custom route for each communication trace in stage 2 such that communication energy is minimized and performance constraints are satisfied. We evaluate the performance of our technique by comparing it with a recent work called NMAP [4], and against an optimal MILP formulation.

The paper is organized as follows: Section 2 discusses previous work, Section 3 presents our automated design technique, Section 4 discusses our experimental results, and finally Section 5 concludes the paper.

## 2. PREVIOUS WORK

Hu et al. [2] and Ascia et al. [5] presented branch and bound and genetic algorithm based techniques, respectively, to map cores onto a regular mesh based NoC architecture. Murali et al. [4] presented a heuristic technique called NMAP for mapping cores and routing traffic traces on mesh based NoC architectures. All existing research only accepts bandwidth constraints on communication traces. The novelty of our technique is that we address the problem of bandwidth and latency constrained NoC design. Unlike previous work, we trade off energy minimization (obtained by routing

high bandwidth traces in minimum hops), with the objective of obtaining legal solutions (by routing tight latency traces in minimum hops) to obtain a pareto optimal point. The results of NMAP were shown to be better than other existing research. We prove that the computational complexity of our technique is lower than NMAP. We also show that for the latency constrained designs, our technique is able to generate high quality solutions while NMAP fails in most cases.

# 3. LOW ENERGY MAPPING AND ROUTING ON NOC ARCHITECTURES

This section presents our technique for design of low energy mesh based on-chip interconnection architectures, henceforth called MOCA. MOCA operates in two phases. In the first phase, it invokes a bi-partitioning based slicing tree generation technique to map cores on to the different routers of the mesh. In the second phase, MOCA invokes a hierarchical router that generates routes for all the communication traces.

## 3.1 MOCA Phase I: Core to router mapping

The MOCA core mapper (CM) takes a CTG and a mesh topology as inputs, and maps the cores to different routers of the given mesh. The mesh is assumed to be placed in the first quadrant of the X-Y plane with the routers placed at unit distance apart. Therefore, the mesh can be denoted by a finite sized plane $P$ defined by bottom left hand side $(x_1, y_1)$ and top right hand side $(x_2, y_2)$ co-ordinates, respectively. Each integral location $(x, y)$ $(x_1 \leq x \leq x_2, y_1 \leq y \leq y_2)$ in the plane denotes the co-ordinate of the respective router. The mapping of the cores to routers assigns a unique $(x, y)$ co-ordinate to each core that corresponds to the router at that particular location.

We determine the coordinates of the cores by recursively invoking the technique proposed by Fiduccia and Mattheyses [6] (FM) to solve the graph equicut problem. We restrict the FM technique to generate partitions with equal sizes. The input to the equicut problem is a graph with weights on its edges. The solution is a partition of the graph such that the two partitions have the same number of nodes, and the cumulative weight of the edges crossing the partition is minimized. Each partitioning step divides the mesh and the CTG into two halves to generate a slicing tree. On algorithm completion, the intermediate nodes of the tree are the directions of each cut (horizontal or vertical) and the leaf nodes are the cores. The CTG is modified through two pre-processing steps before it is mapped onto the mesh.

### 3.1.1 Core mapper pre-processing

The core to router mapping algorithm performs two preprocessing steps.

In the first pre-processing step, CM adds $m$ additional nodes to the $n$ nodes in the CTG such that the total number of nodes in the graph is a power of 4. Mathematically, $2^{2p} < n < 2^{2p+2}$ *and* $n + m = 2^{2p+2}$ for some $p$. This step is performed so that every recursive call to the FM partitioner divides the nodes into two equal halves. Note that $m < 3n$.

The second pre-processing step performed by CM pertains to determining a new weight for each edge in the CTG for min-cut purposes. Bandwidth constraints can be satisfied by finding alternative (sometimes longer) route for the trace. Latency constraints, on the other hand, cannot be adhered to by finding alternative paths. Therefore, CM gives higher priority to latency compared to bandwidth. Let $e_i$ be a trace with the highest bandwidth requirement among all traces in the graph. Let $e_j$ be the trace with tightest

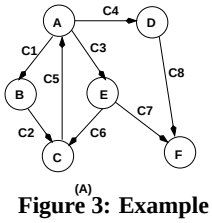


**Figure 3: Example CTG**

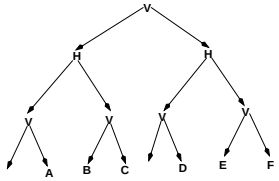


**Figure 4: Partitioning based slicing tree**

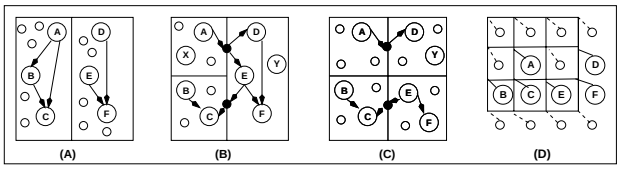


**Figure 5: Output of MOCA Phase I: Core to router mapping**

(lowest) latency constraint among all traces in the graph. CM determines an integer $k$ such that it is the minimum value required to ensure that $\frac{\omega(e_i)}{\sigma(e_i)^k} \leq \frac{\omega(e_j)}{\sigma(e_j)^k}$. Once $k$ is determined, CM assigns an edge weight to each edge given by: $\forall e \in E,\ \rho(e) = \frac{\omega(e)}{\sigma(e)^k}$. For two edges with the same edge weight, the one with tighter latency has higher priority. This heuristic ensures that traces with low bandwidth requirements but with tight latency constraints are given priority over those with high bandwidth requirement and relaxed latency constraints.

### 3.1.2 Slicing tree based core mapping

Given a mesh based interconnection network placed in the first quadrant of the X-Y plane, the CM generates a slicing tree by recursively dividing the mesh into two equal halves by partitioning it either vertically, or horizontally. Figures 3, 4, and 5 give examples of the input CTG, slicing tree, and various stages of the algorithm execution, respectively. In the figure, the first cut is a vertical cut, the two child cuts are horizontal, and so on. The leaves of the tree are occupied by the nodes of CTG. The position of the node in the tree indicates its coordinates after successive partitioning steps.

Figure 6 shows the algorithm for mapping cores onto the routers of the mesh. The slicing tree based core mapping technique maintains a queue data structure ($Q$). Each element of the queue consists of a subgraph $G_i$, a sub-plane $P_i$, and a direction of cut $dir\_cut$. The queue is utilized to perform a breadth first traversal of the slicing tree. Initially, the given X-Y plane $P$, the graph $G$, and a cut direction $dir\_cut$ are enqueued. Without loss of generality, we assume that the first cut is vertical. The $slp$ function is invoked by passing the queue as a parameter to generate a mapping of nodes to routers.

For the purpose of the discussion on $slp$, we denote the subgraph belonging to the element at the head of the queue as $G$, the corresponding sub-plane as $P$, and the corresponding cut as $dir\_cut$. The head of the queue is dequeued and stored in a temporary data structure. Initially, $slp$ checks if the sub-plane is a point. If yes, the node in the corresponding subgraph is mapped to the router located at the location of the sub-plane, and the function returns.

If the sub-plane is not a point, the $slp$ function performs the following steps on the temporary data structure. First, $slp$ invokes the FM technique to partition $G$ into subgraphs $G_1$, and $G_2$. The second step partitions $P$ into two sub-planes, $P_1$ and $P_2$ of equal size. If the direction of the cut is vertical ($dir\_cut = v\_cut$ ), $P$ is partitioned into left sub-plane and right sub-planes. On the other hand, if the direction of the cut is horizontal, $P$ is partitioned into top and bottom sub-planes.

```
CM (G)
    add_extra_nodes() / * Pre-processing step 1 */
    assign_edge_weights() /* Pre-processing step 2 */
    enqueue(Q,G,P,v_cut)
    slp(Q)
end

slp(Q)
    (G,P,dir_cut) = dequeue(Q)
    if (x_1 = x_2 AND y_1 = y_2)
        M(v) = n, s.t. x(n) = x_1, y(n) = y_1 /* node → rtr */
        return()
    end if
    (G_1, G_2) = FM(G)
    if (dir_cut = v_cut)
        P_1 = [(x_1, y_1), (⌈(x_1+x_2)/2⌉ - 1, y_2)], P_2 = [(⌈(x_1+x_2)/2⌉, y_1), (x_2, y_2)]
    else /* dir_cut = h_cut */
        P_1 = [(x_1, y_1), (x_2, ⌈(y_1+y_2)/2⌉ - 1)], P_2 = [(x_1, ⌈(y_1+y_2)/2⌉), (x_2, y_2)]
    endif
    add_dummy_nodes(P_1, G_1, G_2)
    if (dir_cut = v_cut) next_cut = h_cut else next_cut = v_cut end if
    enqueue(Q,G_1,P_1,next_cut)
    enqueue(Q,G_2,P_2,next_cut)
    if (|Q| ≠ 0) slp(Q) end if
    return()
end
```

**Figure 6: MOCA Phase I: Core to router mapper**

Assuming that the left partition is processed before the right partition, the partition of each left sub-plane is followed by placing dummy nodes at the intersection with the right sub-plane. Similarly, for a horizontal cut, the partition of each top sub-plane is followed by placing dummy nodes at the intersection with the bottom sub-plane. All crossing traffic traces are captured by these dummy nodes. This dummy propagation step attracts connected nodes towards each other such that, in the final mapping, nodes connected with large edge weights are placed close to each other. In order to place the dummy nodes effectively, the slicing tree should be traversed in a breadth first manner.

The next step determines the direction of $next\_cut$ for $P_1$ and $P_2$ which is complement of $dir\_cut$. Finally, $(G_1, P_1, next\_cut)$, and $(G_2, P_2, next\_cut)$ are enqueued, followed by a recursive call to the $slp$ function. The recursive call initiates a breadth first traversal of the search space such that, when CM terminates, the mapping function $\mathcal{M}$ contains a node to router mapping for all nodes in the original graph $G$.

Figure 5 presents the different stages of slicing tree based mapping of the nodes constituting the CTG. The empty circles in the figure denote the $m$ additional nodes. The black circles denote the dummy nodes. In Figure 5(B), traces A-D, and A-E are captured by the dummy node on the top half plane, and trace C-E is captured by the dummy node on the bottom half plane.

## 3.2 MOCA Phase II: Route generation

The MOCA route generator (RG) is a novel technique that operates on the slicing tree to formulate a unique route for each communication trace (see Figure 7 for pseudo code). RG operates in two stages, namely, RG hierarchical router ($RG_{hier}$) that generates a route for every trace by traversing the slicing tree, and RG shortest path router ($RG_{sp}$) that searches for a minimal distance route for a communication trace that was not successfully routed by $RG_{hier}$.

### *3.2.1 RG hierarchical router*

The $RG_{hier}$ attempts to find a minimal path from the source to the destination for each traffic trace. $RG_{hier}$ traverses the slicing

```
RG ()
    RG_hier(P,v_cut)
    RG_sp(tbd_trace_list)
end

RG_hier(P, dir_cut)
    if (x_1 = x_2 AND y_1 = y_2) return() end if
    if (dir_cut = v_cut)
        P_1 = [(x_1, y_1), (⌈(x_1+x_2)/2⌉ - 1, y_2)], P_2 = [(⌈(x_1+x_2)/2⌉, y_1), (x_2, y_2)]
        C_1 = ((x_1+x_2)/2, y_1), C_2 = ((x_1+x_2)/2, y_2)
    else /* dir_cut = h_cut */
        P_1 = [(x_1, y_1), (x_2, ⌈(y_1+y_2)/2⌉ - 1)], P_2 = [(x_1, ⌈(y_1+y_2)/2⌉), (x_2, y_2)]
        C_1 = (x_1, (y_1+y_2)/2), C_2 = (x_2, (y_1+y_2)/2)
    endif
    trace_list = get_traces(C_1, C_2) /*list of traces*/
    for t ∈ trace_list
        (n_1, n_2) = get_routers(t,P_1, P_2)
        update_routers(t, n_1, n_2, R)
        if (mapping_fail(t))
            remove(t, R) /* Remove trace from route */
            add(t, tbd_trace_list) /* Add trace to list for next phase */
        end if
    end for
    if (dir_cut = v_cut) next_cut = h_cut else next_cut = v_cut end if
    RG_hier(P_1, next_cut), RG_hier(P_2, next_cut)
    return()
end
RG_sp(tbd_trace_list)
    for t ∈ tbd_trace_list
        for e ∈ L /* For all physical links in I */
            if (ω(e) + ω(t) > Ω) /* BW violation */
                edge_weight(e) = ∞ else edge_weight(e) = 1
            end if
        end for
        shortest_path(t, I, R)
    end for
end
```

**Figure 7: MOCA Phase II: Route generation**

tree generated by CM and routes traces that cross the cut at each level of the tree. The inputs to $RG_{hier}$ are a plane $P$ that constitutes the mesh, and a direction of cut $dir\_cut$ (vertical in the pseudo code). $RG_{hier}$ returns a set of ordered tuples $\mathcal{R}$, where each $r_i \in \mathcal{R} = \{(x_1, y_1), (x_2, y_2), \ldots\}$ denotes a unique route for every traffic trace $e_i \in$ CTG. Each $(x_j, y_j) \in r_i$ denotes the router through which the trace is routed. $RG_{hier}$ also returns a list of traffic traces that could not be routed due to violation in either the bandwidth or latency constraints.

Initially, $RG_{hier}$ checks if the plane passed to it is a point. If yes, the function returns as no routing is necessary. If the plane is not a point, $RG_{hier}$ generates a partial route for the traces. It considers two partitions of the given sub-planes, $P_1$ and $P_2$ as defined by CM (discussed in Section 3.1.2). It generates the cut that defines the two partitions as $(C_1, C_2)$, where $C_1$ and $C_2$ are the end points of the cut. Any traffic trace that crosses the cut $(C_1, C_2)$ to complete its route from source to sink, is added to the list of traces to be routed ($trace\_list$). A partial route for the traces in $trace\_list$ is generated by assigning the traces on the routers adjacent to the cut. This step is equivalent to assigning the traces to physical links that are across the cut $(C_1, C_2)$ subject to the bandwidth constraints. Clearly, this is a knapsack problem and is known to be NP-Complete.

We route traces on the respective routers by considering the traces in a decreasing order of their bandwidth requirement. The pair of routers that are connected to the physical links affected by the cut are considered for routing the trace. The trace is routed through the pair of routers that is closest to the source, and can support the traffic without bandwidth violation. The selection of the pair of routers is performed by *get_router()* in Figure 7. Once the trace is routed, the partial route of the trace in the set $\mathcal{R}$ is updated.

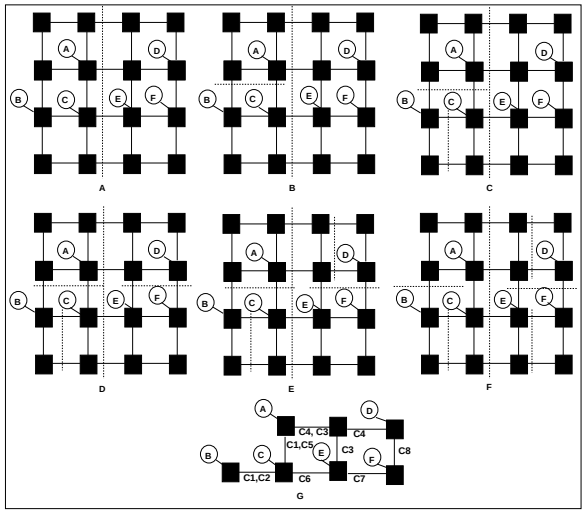


**Figure 8: MOCA Phase II: Route generation example**

Figure 8 gives an example of the algorithm execution on the mapping shown in Figure 5(D). In the figure the black squares represent the routers, the solid lines represent the links, and the labelled circles represent the cores from Figure 3. The dotted lines in Figure 8 refer to the successive cut lines that are generated during the algorithm execution. The NoC architecture at the end of the routing stages is shown in Figure 8(G). As an example of the traces that are selected to be routed across a cut, Figure 8(B) maps traces (A,B), (A,C), and (A,E) across the horizontal cut in the left hand side.

We state the following theorem regarding the optimality of $RG_{hier}$.
*Theorem*: $RG_{hier}$ finds a minimal path if bandwidth constraints are not violated.
*Proof*: Routing of traces through the routers adjacent to a cut take the trace closer to the sink in the X-direction for vertical cut, and Y-direction for horizontal cut. Since the trace is routed through the router closest to the source, at any point, the distance between the router and the source is minimum. This is Dijkstra's shortest path algorithm, and therefore, the path is minimal. Q.E.D

If a trace cannot be mapped to any router in the cut due to bandwidth violation, the partial route for the trace in the set $\mathcal{R}$ is removed, and the trace is added to a list of unmapped traces. The $RG_{hier}$ function finally makes a recursive call on $P_1$ and $P_2$ by setting the $dir\_cut$ parameter to be complementary to the corresponding value for its call with $P$. The recursive call initiates a depth first traversal of the tree such that, when all instances of the function return, each trace in the edge set $E$ of the CTG either has a route in $\mathcal{R}$, or the edge is present in the list of unmapped traces called $tbd\_trace\_list$.

### *3.2.2 RG shortest path router*

The RG shortest path router is called for each traffic trace that is left unmapped at the end of $RG_{hier}$ stage. For each trace in $tbd\_trace\_list$, $RG_{sp}$ sweeps the links $L$ of the mesh, and assigns an edge weight of $\infty$ to all links that would see a bandwidth violation on the corresponding ports constituting the links, if the trace was routed through that link. This step is followed by invoking the Dijkstra's shortest path algorithm to find a route for the trace on the mesh.

The solutions generated by MOCA could have deadlocks that can be removed by a post-processing step that introduces additional virtual channels at selected routers [7].

| Graph | Graph ID | Nodes | Edges |
|---|---|---|---|
| DSP filter | G1 | 6 | 5 |
| H.263 encoder | G2 | 7 | 7 |
| MP3 encoder | G3 | 8 | 8 |
| H.263 enc MP3 dec | G4 | 12 | 11 |
| MPEG4 decoder | G5 | 12 | 13 |
| MWD | G6 | 12 | 13 |
| VOPD | G7 | 12 | 13 |
| MP3 enc MP3 dec | G8 | 13 | 12 |
| H.263 enc MP3 enc | G9 | 15 | 17 |
| H.263 enc H.263 dec | G10 | 16 | 17 |

**Table 1: Graph Characteristics**

## 3.3 Complexity Analysis

**Complexity of CM :** Let $n$ be the number of nodes in the CTG, $u$ be the number of nodes in the mesh, $e$ be the number of edges in the CTG, and $f$ be the number of links in the mesh. MOCA adds $m$ nodes such that $n + m = u$. As explained in Section 3.1, $n + m = u < 4n$. In a square mesh based network with $u = 2^{2p}$ for $p \geq 1$, $f = 2(u - u^{\frac{1}{2}})$. The initial processing in CM takes linear time. CM performs at most $(u - 1)$ partitions as denoted by the number of internal nodes of a balanced binary tree. The FM technique has a linear time complexity in total number of pins when the input is a hypergraph [6]. In the case of a directed graph such as the CTG, the total number of pins is $O(e)$. Therefore, the overall complexity of CM is $O(ue)$.
**Complexity of RG :** $RG_{hier}$ performs routing by traversing a slicing binary tree of height $log_2(u)$ that denotes $(u - 1)$ partitions. During the processing of each slice the algorithm explores at most $e$ traces. Traces can be sorted during pre-processing. The complexity of the algorithm for processing each slice is given by the product of the maximum number of traces and the number of links. The number of links explored at a particular internal node of the tree are half that of the parent. However, the total number of links explored by $RG_{hier}$ at each level of the binary tree are equal, and are given by $u^{\frac{1}{2}}$. Therefore the complexity of $RG_{hier}$ is given by:

$$eu^{\frac{1}{2}} + 2 \cdot \frac{eu^{\frac{1}{2}}}{2} + 4 \cdot \frac{eu^{\frac{1}{2}}}{4} \cdots log_2(u) \ \ terms = eu^{\frac{1}{2}} log_2(u)$$

$RG_{sp}$ calls the shortest path algorithm for each trace. The shortest path algorithm has a complexity of $O(f + u)$. Hence, the complexity of $RG_{sp}$ is $O(e(f + u)) = O(e(3u - 2u^{\frac{1}{2}}))$.
**Complexity of MOCA :** From the analysis presented above, the overall complexity of MOCA is given by $O(max(eu, eu^{\frac{1}{2}} log_2(u), e(3u - 2u^{\frac{1}{2}})) = O(eu)$.
**Comparison with NMAP :** The authors of NMAP computed the complexity of NMAP to be $O(eu^3\ log(f))$. Substituting for $f$, the complexity of NMAP is given by $O(eu^3\ log(2u - 2u^{\frac{1}{2}}))$ which is greater than the complexity of MOCA.

# 4. RESULTS

In this section we present the results obtained by the execution of our technique on various multimedia benchmark applications. We generated custom NoC architectures for six combinations of four multimedia benchmarks: MP3 audio encoder, MP3 audio decoder, H.263 video encoder, and H.263 video decoder [2]. In addition, we obtained results for four other benchmarks: MPEG4 decoder, video object plane decoder (VOPD), multi-window display (MWD), and DSP filter application (DSP) [8][4]. Table 1 lists the graph IDs and sizes of the CTG of the various benchmarks.

We compared the topologies generated by MOCA against those generated by NMAP, and an optimal MILP formulation [9]. Table 2 presents the results of the comparison in energy consumption of the topologies generated by MILP, NMAP, and MOCA for traces with latency constraints. In the table, the sixth column presents

| No. | Graph | Energy ($nJ$) | | | Ratio | |
|---|---|---|---|---|---|---|
| | | MILP | NMAP | MOCA | MOCA vs MILP | MOCA vs NMAP |
| 1 | G1 | 20.935 | 20.935 | 27.109 | 1.3 | 1.3 |
| 2 | G2 | 2143.2 | FAIL | 2400.2 | 1.11 | NA |
| 3 | G3 | 91.6 | FAIL | 103.13 | 1.12 | NA |
| 4 | G4 | 2203.6 | FAIL | 2830.0 | 1.28 | NA |
| 5 | G5 | 70.93 | FAIL | 103.13 | 1.45 | NA |
| 6 | G6 | 10.16 | FAIL | 11.191 | 1.10 | NA |
| 7 | G7 | 35.29 | FAIL | 39.227 | 1.11 | NA |
| 8 | G8 | 154.23 | FAIL | 218.61 | 1.41 | NA |
| 9 | G9 | 2128.9 | FAIL | 2724.3 | 1.27 | NA |
| 10 | G10 | 2166.4 | FAIL | 2378.9 | 1.09 | NA |

**Table 2: Comparison of MILP, NMAP and MOCA: With latency constraints**

| No. | Graph | Energy ($nJ$) | | | Ratio | |
|---|---|---|---|---|---|---|
| | | MILP | NMAP | TEMPO | MOCA vs MILP | MOCA vs NMAP |
| 1 | G1 | 20.935 | 20.935 | 27.109 | 1.29 | 1.29 |
| 2 | G2 | 1960.5 | 1960.5 | 1960.5 | 1 | 1 |
| 3 | G3 | 91.362 | 91.362 | 91.738 | 1.00 | 1.00 |
| 4 | G4 | 2018.6 | 2018.6 | 2018.6 | 1 | 1 |
| 5 | G5 | 64.059 | 72.453 | 68.401 | 1.06 | 0.94 |
| 6 | G6 | 9.7440 | 10.708 | 10.612 | 1.08 | 0.99 |
| 7 | G7 | 33.862 | 34.345 | 36.178 | 1.06 | 1.05 |
| 8 | G8 | 152.52 | 152.52 | 165.45 | 1.08 | 1.08 |
| 9 | G9 | 2047.8 | 2366.4 | 2047.8 | 1 | 0.86 |
| 10 | G10 | 2075.7 | 2075.6 | 2080.5 | 1.00 | 1.00 |

**Table 3: Comparison of MILP, NMAP and MOCA: Without latency constraints**

| Node | MPEG4 | VOPD |
|---|---|---|
| 0 | VU | VLD |
| 1 | SDRAM | RUN LEN DEC |
| 2 | ADSP | INV SCAN |
| 3 | AU | STRIPE MEM |
| 4 | UPSP | IQUANT |
| 5 | MCPU | ACDC PRED |
| 6 | SRAM1 | IDCT |
| 7 | RAST | ARM |
| 8 | SRAM2 | UP SAMP |
| 9 | BAB | VOP REC |
| 10 | IDCT | PAD |
| 11 | RISC | VOP MEM |

**Table 4: Node descriptions for MPEG4 and VOPD**

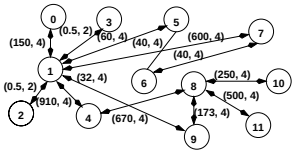


**Figure 9: CTG for MPEG 4 decoder**

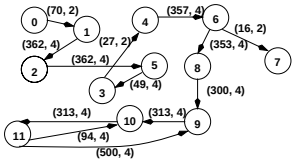


**Figure 10: CTG for VOPD**

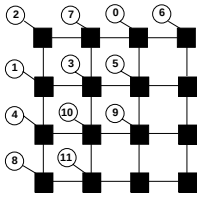

**Figure 11: MPEG4 with latency constraints**

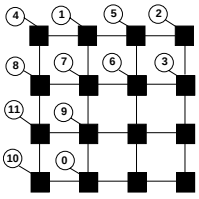

**Figure 12: MPEG4 without latency constraints**

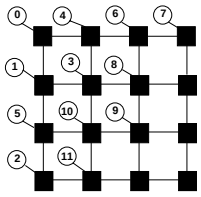

**Figure 13: VOPD with latency constraints**

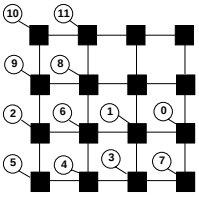

**Figure 14: VOPD without latency constraints**

the ratio of the energy consumption of MOCA over MILP, and the seventh column presents ratio of energy consumption of MOCA over NMAP. Since NMAP does not consider latency constraint, the topologies produced by the technique violated latency constraints on most test cases. On the other hand, MOCA was able to generate topologies satisfying latency constraints for all test cases. On an average, the energy consumption of MOCA was within 22 % of the MILP. The solution time of MILP grows exponentially with input size unless $P = NP$. While MILP timed out for most input cases, MOCA was able to generate results within $0.01secs$.

Table 3 presents the comparison of MOCA with MILP and NMAP for the input traces without latency constraints. As is evident from the table, both NMAP and MOCA performed as well as MILP in many cases. On average, MOCA performed within 6 % of the solution generated by MILP. The overall variation of MOCA against NMAP was negligible. However, as we proved earlier the algorithmic complexity of MOCA is lower than that of NMAP.

Figures 9, and 10 present trace graphs for MPEG4 decoder, and VOPD, respectively. The description of the various nodes in the two figures is shown in Table 4. The labels of the edges denote bandwidth requirement in Mbps, and latency constraint in router hops, respectively. Figures 11, and 12 present the mesh based NoC architectures for MPEG4 generated by MOCA for the latency constrained and latency unconstrained cases, respectively. The corresponding designs for VOPD are shown in Figures 13, and 14. Since MOCA gives higher priority to latency, traces with tight latency are routed through minimum hops. For example, in the case of MPEG4 decoder, trace (1,2) is routed in only two hops due to its tight latency (2 hops). Note that when latency is not a constraint, the same trace is routed in three hops due to its low bandwidth requirement.

## 5. CONCLUSION

We presented a novel polynomial time heuristic technique called MOCA for automated design of low energy mesh based NoC architectures. We proved that the algorithmic complexity of MOCA is lower than that of NMAP [4]. MOCA takes latency constraint on the traces into consideration, and is able to generate valid topologies under tight latency constraints for all benchmarks while NMAP fails for many designs. The quality of results of MOCA and NMAP are comparable when latency is not a constraint. We also compared MOCA against an optimal MILP formulation, and it could produce solutions that were within 14% of the optimum.

## 6. REFERENCES


[1] L. Benini and G. De-Micheli. "Networks on Chips: A New SoC Paradigm". *IEEE Computer*, pages 70–78, January 2002.

[2] J. Hu and R. Marculescu. "Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints". In *ASP-DAC*, 2003.

[3] N. Banerjee, P. Vellanki, and K. S. Chatha. "A Power and Performance Model for Network-on-Chip Architectures ". In *Proceedings of DATE*, Paris, France, February 2004.

[4] S. Murali and G. De-Micheli. "Bandwidth-Constrained Mapping of Cores onto NoC Architectures". In *DATE*, 2004.

[5] G. Ascia, V. Catania, and M. Palesi. "Multi-objective Mapping for Mesh-based NoC Architectures". In *Proceedings of ISSS-CODES*, 2004.

[6] C.M Fiduccia and R.M Mattheyses. "A Linear-Time Heuristic for Improving Network Partitions ". In *Proceedings of DAC*, 1982.

[7] W. J. Dally and B. Towles. "Route Packet, Not Wires: On-Chip Interconnection Networks". In *Proceedings of DAC*, June 2002.

[8] A. Jalabert, S. Murali, L. Benini, and G. De-Micheli. "xpipesCompiler: A tool for instantiating application specific Networks on Chip". In *DATE*, 2004.

[9] K. Srinivasan, K. S. Chatha, and Goran Konjevod. "Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures ". In *Proceedings of ICCD*, San Jose, USA, October 2004.