TEMPO: A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures

ABSTRACT

Network-on-chip (NoC) has been proposed as a solution for the communication challenges of System-on-chip (SoC) design in the nanoscale regime. NoC design problem involves mapping of cores to router ports, and routing of traffic traces on the interconnection architecture such that the bandwidth and latency constraints are satisfied. We present a novel technique called TEMPO that solves the NoC design problem with an objective of minimizing the communication energy. In contrast to existing research that only take bandwidth constraint as an input, TEMPO solves the NoC design problem in the presence of bandwidth as well as latency constraints. We compare TEMPO with a recent work called NMAP [1] and an optimal MILP based formulation. We prove that the complexity of TEMPO is lower than that of NMAP. For the latency constrained case, while NMAP fails on most test cases, TEMPO is able to generate high quality results. In comparison to the MILP formulation, the results produced by TEMPO were within 14 % of the optimal.

1. INTRODUCTION

The advent of deep sub-micron technology has put forward many interesting challenges in high end System-on-ship (SoC) design. Architectures are expected to be clocked in multi gigahertz range, making synchronous communication infeasible. Noise due to increased RLC effects is expected to play a more prominent role. Network-on-Chip (NoC) has been proposed as a solution for the communication challenges in the nanoscale regime [2]. A NoC is characterized by asynchronous communication between routers, high bandwidth by distribution of signal delay among routers, packet switching based communication, isolation and support for concurrent communication, and scalability. Design of SoC in the gigascale era will have energy minimization as the primary design goal, and communication energy is expected to account for a significant portion of the total energy consumption [2].

The paper addresses the design of a NoC in the context of application specific SoC architecture. A number of researchers have addressed architectures and tools for mesh based NoC [3] [4] [1]. Mesh based interconnection network architectures have the advantage of a regular two-dimensional structure that enables scalability, easier layout, and reduced wiring. Our focus is on core mapping and routing on a mesh based NoC to minimize the total communication energy subject to bandwidth and latency constraints.

An NoC router architecture places a constraint on the peak bandwidth that can be supported by its ports. Periodic applications with deadlines specify an upper bound on the number of clock cycles available for communication over the network. In other words, each communication trace has a deadline or latency constraint that has to be satisfied in order to satisfy the overall performance constraint of the application. The latency constraint can be obtained by profiling the application and estimating the computation and communication time spent by each processing core and the packet route, respectively. Therefore, any NoC design framework must address bandwidth and latency constraints on the communication traces.

The core mapping and routing problem is known to be intractable [1], thus justifying heuristic based solutions to the problem. 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 technique. A pre-determined routing technique may result in low resource utilization. Moreover, a pre-determined routing technique may drive the network into congestion mode, thus requiring even more resources to generate valid solutions. On the other hand, an application specific routing technique can be designed such that resource utilization is maximized, and network congestion is avoided.

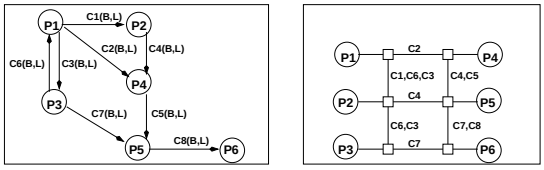

Figure 1: Low energy mapping and routing

Figure 1 shows the core mapping and routing problem. The input to the problem is a computation specification in the form of a communication trace graph. Each node in the graph represents a processing 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. Applications in multimedia and network processing domains demonstrate fairly well defined periodic communication characteristics and hence can be easily modeled in the trace graph.

The output of TEMPO is a mapping of cores onto different routers, a corresponding mesh based NoC, and a static packet route for each traffic trace. 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 100 nm technology with the help of a cycle accurate energy and performance evaluator.[1] In the interest of space, we have omitted the details of the experiments. The variation of energy with injection rate and acceptance rate are shown in Figures 2 and 3, respectively. 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 4. We observe from the plot that average latency remains almost constant in the un-congested mode, and onset of congestion is marked by a sharp increase in latency. TEMPO prevents network congestion by static routing of the communication traces subject to the peak bandwidth constraint on the router ports. Since the network is always operated in the un-congested mode, we can represent the network latency constraint in terms of router hops (such as 1 or 2) instead of an absolute number (such as 100 cycles).

In the following section we define the NoC design problem.

1.1 Problem Definition

Given:

- A directed communication trace graph $G(V, E)$, where each $v_i \in V$ denotes either a processing element or a memory

[1] reference omitted for anonymous review

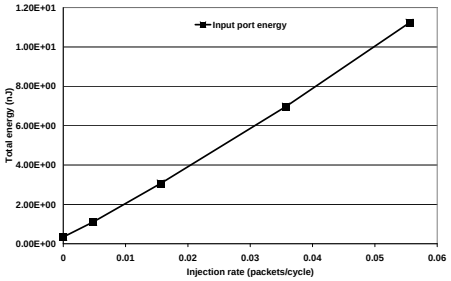

Figure 2: Input traffic energy

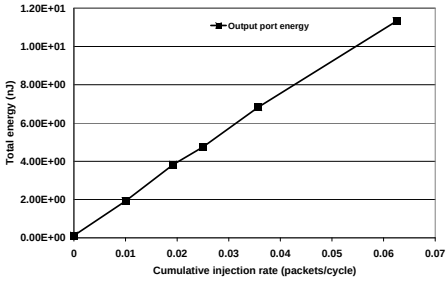

Figure 3: Output traffic energy

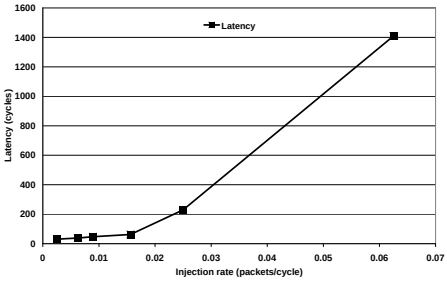

Figure 4: Latency versus injection rate

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 cycle, 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 neighbouring 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:
  - Ω which denotes the peak input and output bandwidth that the router can support on any one port,
  - Ψ_i that denotes the energy consumed per $Mbps$ of traffic bandwidth flowing in the input direction for any port of the router, and
  - Ψ_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 $r_i\langle n_i, n_j, \ldots, n_k\rangle \in \mathcal{R}$ 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 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. Traffic flowing in a network can be reduced by placing communicating cores close to each other.

The latency constraints imposed by the traffic traces specify the maximum number of hops allowed for the trace. 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 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 [5]. A mapping and routing framework 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 called TEMPO 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 communcation 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 [1], and against an optimal MILP formulation.

The paper is organized as follows: In section 2 we discuss previous work, in section 3 we present our mapping and routing technique called TEMPO, in section 4 we discuss our experimental results, and finally in section 5 we conclude the paper.

2. PREVIOUS WORK

In recent past, researchers have begun to address the problem of automated design of NoC architectures. Existing research can be broadly classified into two categories: i) optimization of mesh based NoC by intelligent mapping and routing, and ii) synthesis of application specific NoC topologies. Our work in this paper falls in the first category.

Hu et al. [4] presented a branch and bound technique to map cores onto a regular mesh based NoC architecture. In [6], Hu et al. extended their previous work to incorporate a deadlock free routing function. Murali et al. [1] presented a technique called NMAP, for mapping cores and routing traffic traces on mesh based NoC architectures. Ascia et al. [7] presented a genetic algorithm based technique for mapping cores to mesh based architectures. All existing research only accept bandwidth constraints on communication traces. The novelty of our technique is that we address the problem of bandwidth and latency constrained core mapping and routing. 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. In order to demonstrate the superiority of our technique, we compared TEMPO with NMAP. We prove that the computational complexity of our technique is lower than NMAP. We also show that for the latency constrained designs, TEMPO is able to generate high quality solutions while NMAP fails in most cases.

Pinto et al. [8] presented a technique for constraint driven communication architecture synthesis of point to point links by utilizing deterministic heuristic based k-way merging. Their technique results in network topologies that have only two routers between each source and sink. Hence, their problem formulation does not address routing. In contrast, we address the problem of mapping and routing in mesh based NoC architectures. In [5], Murali et al. presented a technique that integrates physical planning with quality of service. They propose a computationally complex solution for the problem, where they iteratively invoke an MILP based placement technique in a tabu search framework. Tabu search itself is a computationally expensive technique [9]. Further, the complexity

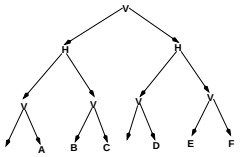

Figure 5: Slicing Tree

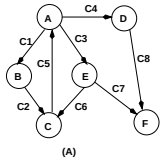

Figure 6: CTG

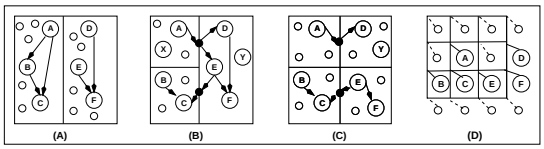

Figure 7: TEMPO Placement

of MILP is exponential in the number of inputs unless $P = NP$. Invoking MILP iteratively inside a tabu search loop makes the algorithm prohibitively expensive. On the other hand, we present a low complexity heuristic technique that generates close to optimal results for mesh based NoC architectures as shown in Section 4.

3. TEMPO

In this section, we present our energy aware core mapping and routing techinque called TEMPO. TEMPO is a heuristic technique that places cores with high edge weights close to each other, thus routing the corresponding traffic trace in minimum number of hops. The technique operates in two phases. In the first phase, TEMPO invokes a slicing tree based technique to map processing cores on different routers of a mesh network. In the second phase, TEMPO invokes a hierarchical router that routes all traffic traces. In Section 3.1, we describe the slicing based core mapping technique, and in section 3.2, we describe our routing and topology generation technique.

3.1 TEMPO Core Mapping Technique

Figure 8 shows the algorithm for mapping cores onto different routers of the mesh. Figures 5, 6 and 7 give examples of the slicing tree, input CTG and various stages of the algorithm execution, respectively. The TEMPO core mapper (TCM) takes a communication trace graph as input, and maps the cores to different routers of the given mesh $I(N, L)$. The coordinates of the nodes are determined by recursively invoking the partitioning technique proposed by Fiduccia and Mattheyses [10] to solve the graph equicut problem, and each node is mapped to the router available at its coordinate.

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. In this paper, we invoke the technique presented by Fiduccia and Mattheyses (FM) [10] to generate a partition at a given level of hierarchy. We restrict the FM technique to generate partitions with equal sizes. In Section 3.1.1, we present the pre-processing steps of TCM, and in Section 3.1.2 we describe the mapping technique.

3.1.1 TEMPO Core Mapper Preprocessing

TEMPO core mapper (TCM) performs two preprocessing steps before invoking slicing tree based mapping technique.

In the first preprocessing step, TCM 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 $m < 3n$.

TCM *(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 $y_1 = y_2$)*
$\mathcal{M}(v) = n$, *s.t.* $x(n) = x_1, y(n) = y_1$ */* node $\rightarrow$ rtr */*
return()
end if
(G_1, G_2) = FM(G)
if *(dir_cut = v_cut)*
$P_1 = [(x_1, y_1), (\lceil \frac{x_1+x_2}{2} \rceil - 1, y_2)]$, $P_2 = [(\lceil \frac{x_1+x_2}{2} \rceil, y_1), (x_2, y_2)]$
else */* dir_cut = h_cut */*
$P_1 = [(x_1, y_1), (x_2, \lceil \frac{y_1+y_2}{2} \rceil - 1)]$, $P_2 = [(x_1, \lceil \frac{y_1+y_2}{2} \rceil), (x_2, y_2)]$
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| $\neq$ 0) slp(Q)* **end if**
return()
end

Figure 8: TEMPO Core Mapper

The second preprocessing step performed by TEMPO core mapper pertains to determining the edge weight of each edge in the input graph. Bandwidth constraints can be satisfied by finding alternative (sometimes longer) route for the trace. Latency constraint on the other hand cannot be adhered to by finding alternative paths. Therefore, TCM 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 (lowest) latency constraint among all traces in the graph. TCM 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, TCM 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 Mapping

Given an X-Y plane P, the slicing tree recursively divides the plane into two equal halves by partitioning it either vertically, or horizontally. The direction of cut of the two children in the tree is complementary to that of the parent. In the figure, the first cut is a vertical cut, the two children cuts are horizontal, and so on. The leaves of the tree are occupied by nodes. The position of the node in the tree indicates its coordinates after successive partitioning steps.

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. Note that if the sub-plane is a point, the corresponding subgraph contains exactly one node. This property is ensured by adding extra nodes during preprocessing.

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 the 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-plane, respectively. On the other hand, if the direction of the cut is horizontal, P is partitioned into top and bottom sub-planes, respectively.

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, highly communicating nodes 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 is to determine the direction of cut for P_1 and P_2. For P_1 and P_2, the direction of cut assigned is complementary to that of P. Let $next\_cut$ denote the direction of cut for P_1 and P_2. 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 function such that, when TCM terminates, the mapping function $\mathcal{M}$ contains a node to router mapping for all nodes in the original graph G.

Figure 7 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 7(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 TEMPO Routing Technique

The TEMPO routing technique (TRT) is shown in Figure 9 and Figure 10 gives an example of the algorithm execution on the mapping shown in Figure 7(D). In the figure the black squares represent the routers, the solid lines represent the links, and the labeled circles represent the cores from Figure 6. TRT is a novel routing technique that operates on the slicing tree to generate a route for all traffic traces. TRT operates in two phases namely, TRT minimal router (TRT_{min}), and TRT shortest path router (TRT_{sp}).

3.2.1 TRT Minimal Router

The TRT minimal router attempts to find a minimal path from the source to the destination for each traffic trace. The input to the TRT_{min} function is a sub-plane P, and a direction of cut dir_cut. The function is called by passing the original X-Y plane that constitutes the mesh $I(N, L)$, and a vertical cut to it. Note that similar to the *slp* function, there is no loss of generality in passing the vertical cut. TRT_{min} returns a set of ordered tuples $\mathcal{R}$, where each $r_i \in \mathcal{R}$ denotes a route for a unique traffic trace. TRT_{min} also returns a list of traffic traces that could not be routed without violating the bandwidth or latency constraints.

Initially, TRT_{min} checks if the sub-plane passed to it is a point. If yes, the function returns as no routing is necessary. If the sub-plane is not a point, TRT_{min} performs the following steps. First, it partitions the given sub-plane into sub-planes P_1 and P_2 as described in Section 3.1.2. The cut is denoted by (C_1, C_2), where C_1 and C_2 are the end points of the cut. The next step is the determination of traffic traces to be routed. Any traffic trace that is required to cross the cut (C_1, C_2) to complete its route in $\mathcal{R}$, is added to the list of traces to be routed ($trace\_list$). The next step is the routing of the traces in $trace\_list$ on the routers adjacent to the cut. Clearly, this is a knapsack problem and is known to be NP-Complete.

```
TRT ()
    TRT_hier(P,v_cut)
    TRT_sp(tbd_trace_list)
end

TRT_min (P, dir_cut)
    if (x1 = x2 AND y1 = y2) return() end if
    if (dir_cut = v_cut)
        P1 = [(x1, y1), (⌈(x1+x2)/2⌉ − 1, y2)], P2 = [(⌈(x1+x2)/2⌉, y1), (x2, y2)]
        C1 = ((x1+x2)/2, y1), C2 = ((x1+x2)/2, y2)
    else /* dir_cut = h_cut */
        P1 = [(x1, y1), (x2, ⌈(y1+y2)/2⌉ − 1)], P2 = [(x1, ⌈(y1+y2)/2⌉), (x2, y2)]
        C1 = (x1, (y1+y2)/2), C2 = (x2, (y1+y2)/2)
    endif
    trace_list = get_traces(C1, C2) /*list of traces*/
    for t ∈ trace_list
        (n1, n2) = get_routers(t,P1, P2)
        update_routers(t, n1, n2, 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
    TRT_min (P1, next_cut)
    TRT_min (P2, next_cut)
    return()
end
TRT_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 9: TEMPO router

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 effected 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 9. Once the trace is routed, the partial route of the trace in the set $\mathcal{R}$ is updated. The dotted lines in Figure 10 refer to the successive cut lines that are generated during the algorithm execution. We state the following theorem regarding the optimality of TRT_{min}.

Theorem: TRT_{min} 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 bandwidth constraints are violated, the technique attempts to find non-minimal paths. If a trace cannot be mapped to any router in the cut due to bandwidth violation, the partial route for the trace

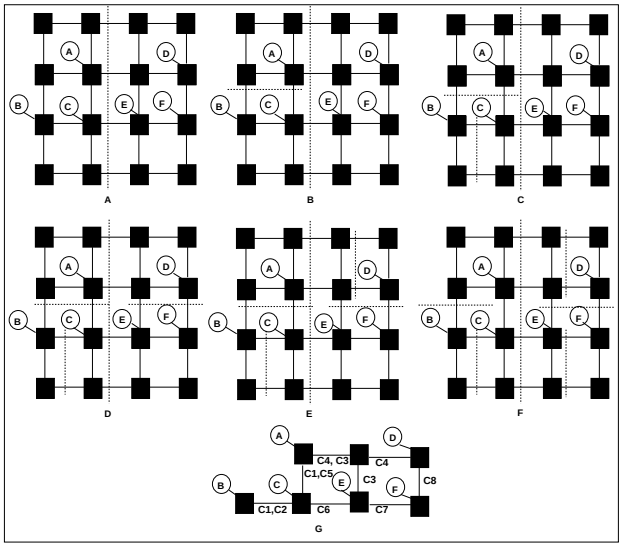

Figure 10: TEMPO Routing

in the set $\mathcal{R}$ is removed, and the trace is added to a list of unmapped traces. The TRT_{min} 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 original graph G 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 TRT Shortest Path Router

The TRT shortest path router is called for each traffic trace that is left unmapped at the end of TRT_{min} phase. TRT_{sp} attempts to find alternate routes for these traces. For each trace in tbd_trace_list, TRT_{sp} sweeps the links L of the mesh $I(N, L)$, and assigns an edge weight of ∞ to all links that would see a bandwidth violation on the ports constituting the links, if the trace was routed through that link. These links are not utilized to generate the route for the trace. This step is followed by calling Dijkstra's shortest path algorithm to find a route for the trace on the mesh.

The NoC at the end of routing and mapping is shown in Figure 10(G). As an example to the traces that are chosen to be routed across a cut, the slice introduced in Figure 10(B) maps traces (A,B), (A,C), and (A,E).

The solutions generated by TEMPO and NMAP could have deadlocks between the various communication traces. However, deadlocks can be easily removed by a post-processing step that introduces additional virtual channels at selected routers [11].

3.3 Complexity Analysis

In this section, we present a complexity analysis of TEMPO and show that its complexity is lower than that of NMAP. We first evaluate the complexity of TCM, then we evaluate the complexity of TRT, and finally, we prove that the complexity of TEMPO is lower than NMAP.

Complexity of TCM Let n be the number of nodes in the graph. Let u be the number of nodes in the mesh. Let e be the number of edges in the graph, and f be the number of links in the mesh. TEMPO adds m nodes such that $n + m = u$. As explained in Section 3.1, $n+m = u < 4n$. The initial processing in TCM takes linear time. TCM performs at most u partitions. The FM technique is a linear time heuristic. Therefore, the overall complexity of TCM is $O(u \cdot u) = O(u^2)$.

Graph	Graph ID	Nodes	Edges
dsp filter	G1	6	5
263 encoder	G2	7	7
mp3 encoder	G3	8	8
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
263 enc mp3 enc	G9	15	17
263 enc 263 dec	G10	16	17

Table 1: Graph Characteristics

Complexity of TRT TRT_{min} performs routing over u slices. The number of links explored is given by

$$f^{\frac{1}{2}} + 2 \cdot \frac{f^{\frac{1}{2}}}{2} + 4 \cdot \frac{f^{\frac{1}{2}}}{4} \cdots = f^{\frac{1}{2}} log(u)$$

Traces can be sorted during preprocessing. At each slice, at most e edges are explored. Therefore, the complexity of TRT_{min} is given by $O(ef^{\frac{1}{2}}\ log(u))$. TRT_{sp} calls the shortest path algorithm for each trace. The shortest path algorithm has a complexity of $O(f + u)$. Hence, the complexity of TRT_{sp} is $O(e(f + u))$.

From the analysis presented above, the overall complexity of TEMPO is given by $O(max(u^2, ef^{\frac{1}{2}}\ log(u), ef))$. In a graph, $e \leq u^2$, and in a mesh, $f = O(u)$. Hence, the complexity expressed in terms of u is given by $O(max(u^{\frac{5}{2}} log(u), u^3)$.

Comparison with NMAP We will prove that the complexity of TEMPO is lower than that of NMAP by contradiction. The authors of NMAP computed the complexity of NMAP to be $O(u^3 elog(f))$. The complexity of NMAP can be lower than TEMPO only when

$$u^3 e\ log(f) \leq u^{\frac{5}{2}}\ log(u)\ or\ u^3 e\ log(f) \leq u^3$$

Substituting the values of e and f in the above equation, we get

$$u^5 log(u) \leq u^{\frac{5}{2}} log(u)\ or\ u^5 log(u) \leq u^3$$

which is a contradiction for all $u \geq 4$ which is the smallest mesh.

The TRT_{min} technique routes multiple traces together and therefore, is able to generate high quality solution under lower complexity compared to finding a shortest path for each trace. Although the overall complexity of TRT is driven by the shortest path algorithm used by TRT_{sp}, in practice, since it is a post processing step, the number of edges that are mapped by TRT_{sp} is very few.

4. RESULTS

We present the results obtained by running our technique on many multimedia benchmark applications. We first discuss the benchmark applications, then we discuss the experimental setup, and finally, we present and discuss the results.

Benchmark applications: We generated custom NoC architectures for six combinations of four multimedia benchmarks namely, i) mp3 audio encoder ii) mp3 audio decoder iii) H.263 video encoder, and iv) H.263 video decoder. The applications were obtained from the work presented by Hu et al. [4]. In addition, we obtained results for four other benchmarks namely, mpeg4 decoder, video object plane decoder (vopd), multi-window display (mwd), and DSP filter application (dsp). The mpeg4 decoder, vopd, and mwd applications were obtained from the work presented by Jalabert et al. [12], and the dsp application was obtained from NMAP [1]. The benchmarks are shown in Table 1.

Experimental setup: We estimated the energy consumption for the input and output traffic of a port in 100 nm technology to be $2.29pJ/Mbps$ and $2.07pJ/Mbps$, respectively. All results were obtained on a 950 MHz dual sparc processor.

Results and discussion We compared the topologies generated by TEMPO against those generated by NMAP, and an optimal MILP

No.	Graph	Energy ($n\ J$)			Ratio	
		MILP	NMAP	TEMPO	TEMPO vs MILP	TEMPO 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 and TEMPO

No.	Graph	Energy ($n\ J$)			Ratio	
		MILP	NMAP	TEMPO	TEMPO vs MILP	TEMPO 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 and TEMPO

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 desc

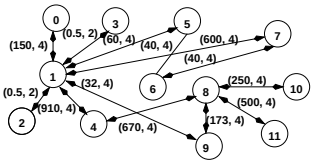

Figure 11: MPEG 4 decoder

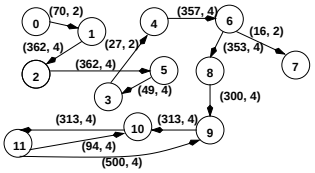

Figure 12: VOPD

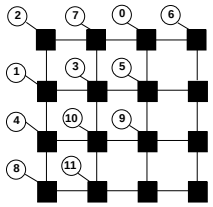

Figure 13: MPEG4: Latency

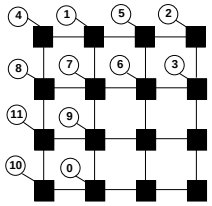

Figure 14: MPEG4: No Lat.

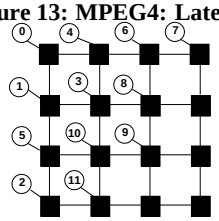

Figure 15: VOPD: Latency

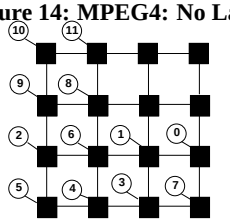

Figure 16: VOPD: No Lat.

formulation. [2]

Table 2 presents the results of the comparison in energy consumption of the topologies generated by MILP, NMAP, and TEMPO for traces with latency constraint. In the figure, the first column gives the serial number, the second column identifies the graph type, the third column presents the energy consumption of MILP, the fourth column presents the energy consumption of NMAP, the fifth column presents the energy consumption of TEMPO, the sixth column presents the ratio of the energy consumption of TEMPO and MILP, and the seventh column presents ratio of energy consumption of TEMPO and 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, TEMPO was able to generate topologies satisfying latency constraints for all test cases. On average, the energy consumption of TEMPO was within 22 % of MILP. Solution time of MILP grows exponentially with input size unless $P = NP$. While MILP timed out for most input cases, TEMPO was able to generate results within $0.01secs$. We have already shown that the complexity of TEMPO is lower than that of NMAP.

Table 3 presents the comparison of TEMPO with MILP and NMAP for the input traces without latency constraints. The organization of the table is similar to that of 2. As is evident from the table, both NMAP and TEMPO performed as well as MILP in many cases. On average, TEMPO performed within 6 % of the solution generated by MILP. The overall variation of TEMPO against NMAP was negligible.

Figures 11, and 12 present trace graphs for MPEG4 decoder, and VOPD, respectively. The labels of nodes in the figures are explained in Table 4. The labels of the edges denote bandwidth requirement in Mbps, and latency constraint in router hops, respectively. Figures 13, and 14 present the mesh topology generated by TEMPO for the latency constrained and latency unconstrained case respectively, for MPEG4 decoder. The corresponding topologies for VOPD are shown in Figures 15, and 16. Since TEMPO 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

In this paper, we proposed a novel technique called TEMPO for low energy mapping and routing in a mesh based NoC. We compared TEMPO against an existing technique called NMAP [1]. In this paper, we proved that the complexity of TEMPO is lower than that of NMAP. Our experiments demonstrate that the quality of results of TEMPO and NMAP are comparable when latency is not a constraint. TEMPO takes latency constraint on the edges into consideration, and is able to generate valid topologies under tight latency constraints. On the other hand, NMAP does not consider latency constraint, and therefore, fails for many latency constrained benchmarks. We also compared TEMPO against an MILP formulation that produces optimal solutions. Overall, TEMPO could produce solutions that were within 14 % of the optimal. While the complexity of MILP is known to be exponential in the number of inputs unless $P = NP$, TEMPO is a polynomial time heuristic technique. The technique is aimed at minimizing dynamic energy consumption, and does not address leakage energy consumption. Future work will include developing dynamic heuristics for leakage energy reduction.

[2] reference omitted for anonymous review

6. REFERENCES

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

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

[3] H-S Wang, L-S Peh and S. Malik. "Orion: A Power-Performance Simulator for Interconnection Network". In *International Symposium on Microarchitecture*, Istanbul, Turkey, November 2002.

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

[5] S. Murali, L. Benini, and G. De Micheli. "Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees ". In *Proceedings of ASPDAC*, 2005.

[6] J. Hu, and Radu Marculescu. "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures". In *DATE*, 2003.

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

[8] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. "Efficient Synthesis of Networks On Chip". In *ICCD*, 2003.

[9] E.D. Taillard. "Robust Tabu Search for the Quadratic Assignment Problem". *Parallel Computing*, 17:443–455, 1991.

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

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

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