NoCOUT : NoC Topology Generation with Mixed Packet-switched and Point-to-Point Networks

Jeremy Chan
School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia
jeremyc@cse.unsw.edu.au

Sri Parameswaran
School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia
sridevan@cse.unsw.edu.au

ABSTRACT

Networks-on-Chip (NoC) have been widely proposed as the future communication paradigm for use in next-generation System-on-Chip. In this paper, we present NoCOUT, a methodology for generating an energy optimized application specific NoC topology which supports both point-to-point and packet-switched networks. The algorithm uses a prohibitive greedy iterative improvement strategy to explore the design space efficiently. A system-level floorplanner is used to evaluate the iterative design improvements and provide feedback on the effects of the topology on wire length.

The algorithm is integrated within a NoC synthesis framework with characterized NoC power and area models to allow accurate exploration for a NoC router library. We apply the topology generation algorithm to several test cases including real-world and synthetic communication graphs with both regular and irregular traffic patterns, and varying core sizes. Since the method is iterative, it is possible to start with a known design to search for improvements. Experimental results show that many different applications benefit from a mix of "on chip networks" and "point-to-point networks". With such a hybrid network, we achieve approximately 25% lower energy consumption (with a maximum of 37%) than a state of the art min-cut partition based topology generator for a variety of benchmarks. In addition, the average hop count is reduced by 0.75 hops, which would significantly reduce the network latency.

1. INTRODUCTION

In the last decade, the available number of transistors per chip has increased by several orders of magnitude. To maximize the use of these transistors, meet crucial time to market deadlines, simplify verification, overcome clock skew problems and reduce reliance upon extremely large design teams, there has been an increased need for microprocessors in the design of emerging embedded systems. Such systems often contain many microprocessors, with a few designs even containing up to 300 microprocessors. This trend of having multiple processors is only expected to increase. Multi-processing in an SoC promises increased performance with reduced power consumption.

Reduced power consumption is enabled in Multi-Processor SoC (MPSoC), by reducing the context switching, having task specific processors (thus small) and switching off idling processors. In embedded systems, which typically execute a single application or a class of applications, it is possible to optimize the system to reduce power. Reduced power consumption benefits the system by performing longer on limited battery supplies, reducing weight by eliminating the need for chip cooling technologies, and increasing reliability by dissipating less energy and running cooler.

Limitations of traditional bus-based architectures start to become apparent when numerous processing elements compete for communication resources. A typical bus-based architecture will require the processing elements to wait, while other data transfers are being carried out. Waiting, in turn, increases energy consumption due to leakage currents and decreases performance. Thus, a new communication paradigm is a prerequisite in MPSoCs to enable processing systems to reduce both latency and energy of data transfers between processors. Networks-on-chip (NoCs) has been proposed as a future replacement of on-chip buses in SoC applications where large bandwidth requirements and plentiful wiring resources are available. However, these networks add considerable logic to the design and require optimization to make them energy efficient.

Motivation for this work

Customizing NoC topologies leads to faster, smaller and more energy efficient networks. They can be made faster, by adding links between congested routers and direct routes between frequently communicating processing cores; smaller, by eliminating links, ports and routers which are under-utilized; and more energy efficient, simply by the virtue of a faster, smaller network.

In this paper, we present an iterative refinement strategy to generate an optimized NoC topology that supports both packet-switched networks and point to point connections. We use cost functions to rank various local improvements to guide the optimization process. The first phase creates a good initial starting point while the second phase optimizes the topology. For each topology change, a system-level floorplanning tool estimates the wire lengths. We use characterized energy macro-models to account for routers with differing number of ports. Experiments are performed to illustrate the potential for the algorithm to generate good custom NoC topologies for specific applications for both asymmetric communication patterns as well as symmetric.

The remainder of the paper is organized as follows: Section 2 surveys the related work and states our contribution. Section 3 overviews the framework and describes the topology generation problem and application and architecture model. Section 4 describes the topology generation algorithm and its implementation. Section 5 presents the experimental results. Section 6 concludes the paper.

2. RELATED WORK

Various NoC router circuits supporting services such as fault tolerance and quality of service have been previously proposed using several different regular topologies. Regular networks benefit by allowing for a simplified placement and early exploration of topologies. Several researchers have developed algorithms to map processing cores onto tiles of regular topologies such as the mesh and torus [8, 12]. In [13], Ogras et al. enhanced the mesh topology by adding customized long-range links to minimize hop distance. Other works have addressed the selection of the best topology from a large library of alternatives [9].

Several methods have been proposed for synthesizing application specific topologies for point-to-point networks and ring architectures [2, 15]. Krishnan et al. [18] presented several mixed integer linear programming (MILP) formulations that minimize total network bandwidth. However, these works lack floorplanning information.

More recently, Krishnan et al. [17] presented a MILP formulation that addresses both wire and router energy by splitting the topology generation problem into two distinct sub-problems: system-level floorplanning and topology and route generation. In [14], Pinto et al. considered the synthesis of topologies including both networks and point-to-point buses. They mapped the topology generation into two sub-problems: *k-median* and *multi-commodity min-cost flow* and solved these problems using approximation algorithms. The main drawback of these two approaches is that it assumes that the routing resources have negligible impact on the floorplan. In [11], Murali et al. proposed a two step topology generation procedure using a general purpose min-cut partitioner to cluster highly communicating cores on the same router and a path allocation algorithm to connect the clusters together.

Our work is similar to that presented in [11], in that we produce an energy optimized topology from an application graph using calibrated energy models with the aid of a system-level floorplanner. How-

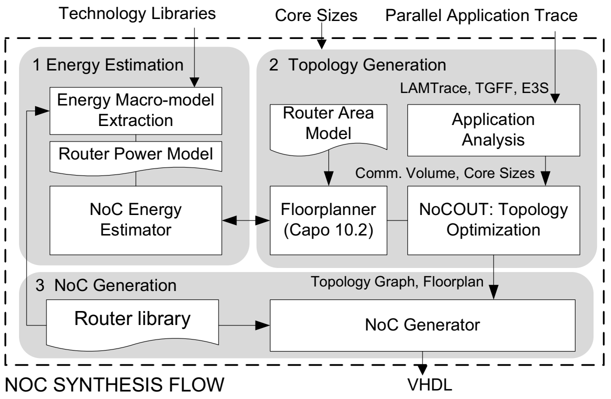


Figure 1: Complete NoC synthesis flow

ever, our approach differs from [11] in the following ways: we support both point-to-point connections and packet-switched networks allowing further energy reductions in certain applications. Instead of exhaustively exploring all balanced cuts of the application graph, we start from a point in the design space and iteratively evolve the solution through a series of guided local moves. The benefit of this approach is that it enables us to control the parts of the design space to be explored, thus allowing additional architecture constraints to be added.

The novel contributions of our work are:

- an iterative optimization technique and selection heuristic for guided design space exploration of NoC topologies using floorplan feedback
- a design flow that allows architectural constraints to be added without major algorithmic changes
- a methodology that supports hybrid networks with both packet-switched routers and point-to-point connections
- an augmented system-level energy model to explore active, idle and leakage energy in custom NoC topology generation

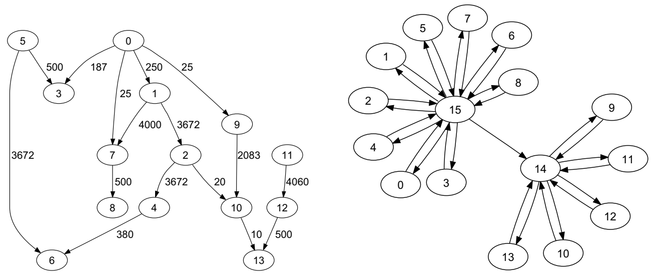
3. SYNTHESIS OF A CUSTOM TOPOLOGY

Figure 1 presents the proposed NoC synthesis flow. There are three major parts to this framework: (i) energy estimation, (ii) NoC optimization and (iii) NoC generation. The focus of this paper is the NoC topology optimization component. The other parts of the flow allow energy characterization and generation of a realizable NoC. We refer interested readers to [4] and [5]. We have implemented all three parts as an extensible web-application. This NoC generation framework allows the creation of new router models through the definition of router templates, automated characterization of their energy characteristics and the creation of an optimized NoC using the algorithms described in this paper.

The input to the NoC optimization step is the application model, NoC router libraries for implementation and an estimate of the silicon area of the processing cores. The NoC optimization step produces an annotated topology graph that describes the interconnection network between the processing cores as well as their placements. Our HDL generator takes this topology graph and produces a synthesizable hardware description of the routers and their interconnections. A floorplanner is used to estimate wire lengths and estimate the energy contribution of the topology. The NoC energy is estimated using characterized macro-models.

3.1 Application Model

In this work, an application volume graph is used to capture the traffic flow characteristics to enable energy estimation. The application graph is a directed graph $A(V, E)$, where each v_i represents a core and the directed edge (v_i, v_j) represents communication between the cores. The communication volume between cores vol_{ij} is specified for each edge. An application run-time t is specified and it is used to calculate the total leakage energy and idle energy dissipated. An example application volume graph for a H.263 decoder combined with an MP3 decoder (263decmp3dec) is shown in Figure 2 (a) from [19]. We use this example throughout the paper to illustrate the operation of our algorithms.



(a) Application graph (b) Topology Graph

Figure 2: Input graphs

In addition to volume, each core in the application graph is annotated with a fixed core size with width $W(v_i)$ and height $H(v_i)$. An application volume capacity constraint vol_{MAX} is specified for each edge to ensure that the generated topology can support the bandwidth requirements. The communication volume vol_{ij} must be less than vol_{MAX} for each edge.

3.2 Architecture Model

A topology graph is used to describe the interconnections between the routers and cores. The *topology graph* is a directed graph $T(N, L)$, where each vertex $n_i \in N$ represents a node in the topology. Each node can be either a core pe_i or router r_i. Each core in the application graph v_i must be mapped to one node pe_i in the topology graph. Each directed edge in the graph (n_i, n_j) represents a physical bus connecting nodes n_i and n_j with traffic flowing in a single direction. For every edge $(v_i, v_j) \in A$, there must exist a path $(n_i, r_i), (r_i, r_k), ...(r_k, n_j)$ in the topology graph T that connects communicating cores n_i and n_j. Figure 2(b) shows the topology output for the 263decmp3dec benchmark.

We use the custom packet-switched wormhole routers from the NoC-GEN framework [4]. We assume a fixed buffering amount on each router and support a maximum of sixteen ports. We limit the number of ports because we lack calibration data for a greater number of ports.

3.3 Problem Description

The topology synthesis problem can be defined as follows: **Given** an application abstraction, a NoC router area model and a characterized router power/energy model, **find** an NoC topology T that minimizes the communication energy E_{noc}.

The NoC energy can be modeled by the combination of the router logic $E(r)$, network interfaces $E(n)$ and interconnect energy $E(e)$. The NoC energy E_{noc} can be modeled as follows:

$$E_{noc} = \sum_{r \in R} E(r) + \sum_{e \in L} E(e) + \sum_{n \in N} E(n)$$

where R is the set of routers and L is the set of links in the interconnection network and N is the set of nodes.

The energy dissipation of each NoC router can be separated into four main categories: active energy E_{dyn}; idle E_{static}; static leakage E_{leak} and wire energy E_{wire}. Active energy is consumed by packet related activities and increases linearly with increasing volume.

The router energy of a single router can be defined as:

$$E_{router}(r_i) = E_{dyn}(r_i) + E_{wire}(r_i) + E_{static}(r_i) + E_{leak}(r_i)$$

To enable efficient computation of the total NoC energy, we separate the router's active energy into input and output components. The active energy component of the edge weights are computed as the output energy of the source node E_{dyn_s} summed with the input energy of the target node E_{dyn_t}. We assume the wire capacitance scales linearly per unit length C_w for a particular technology library. The wire energy also becomes part of the edge weight to the topology graph. The idle energy and leakage is modeled as a fixed cycle energy that is consumed at a given frequency f. Combining these components together,

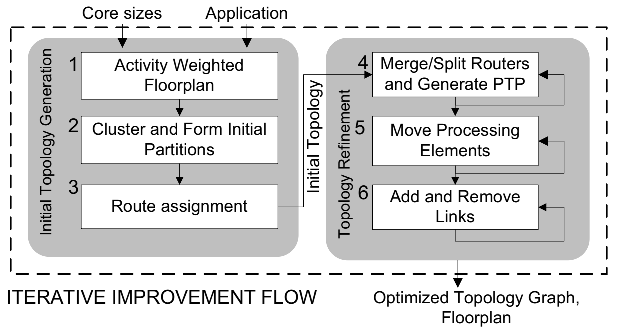

Core sizes
Application
Initial Topology Generation
1 Activity Weighted Floorplan
2 Cluster and Form Initial Partitions
3 Route assignment
Initial Topology
Topology Refinement
4 Merge/Split Routers and Generate PTP
5 Move Processing Elements
6 Add and Remove Links
ITERATIVE IMPROVEMENT FLOW
Optimized Topology Graph, Floorplan

Figure 3: Iterative Improvement Flows

the total NoC energy can be expressed as:

$$E_{noc} = \sum_{(i,j)\in L} (E_{dyn\_s}(n_i) + E_{dyn\_t}(n_j) + C_w d_{ij}) \cdot vol_{ij} + \sum_{n\in N} (E_{static}(n) + E_{leak}(n,f)) \cdot t$$

The linear regression based technique from [5] is used to develop analytical energy models for a library of routers [4] with varying input and output ports. We extracted energy models for a large number of routers, validated them on numerous traces with varying switching activity and timing characteristics with low error rates ($< 5\%$). The energy contributions of the abstract NoC events for each component were collated into lookup tables to allow quick evaluation of energy. Non-characterized points in the energy model are estimated using cubic interpolation. Analytical area models are developed from logic synthesis estimates with input ports, output ports and buffering depth as model parameters. These analytical expressions were found to be very accurate at predicting the gate count ($< 2\%$) as the router is highly modular.

3.4 Floorplanning

A floorplanner is used to evaluate the wire lengths between each processing element and router. Each topology graph is floorplanned and mapped to a two-dimensional layout. Each node in the topology graph n_i is represented by a rectangular region with width, $W(n_i)$ and height, $H(n_i)$ and network interfaces at coordinates $X(n_i)$ and $Y(n_i)$. The distance d_{ij} between two nodes is measured by the manhattan distance between their network interfaces $|X(n_i) - X(n_j)| + |Y(n_i) - Y(n_j)|$. For point-to-point links, the network interfaces are located on the perimeter of cores. As there may be multiple network interfaces on each core, the network interface is assumed to be connected from the center of the cores. The network interfaces for the processing cores are assumed to be located on the corner while routers nodes are in the center. The edges in the floorplanning problem are weighted proportional to the activity (the relative number of packets) that traverses across the links between the nodes. We create dummy edges based on the application graph to allow high communicating pairs of nodes to organize themselves close together to reduce the wire length for the point-to-point networks.

We evaluated three academic standard-cell floorplanners (Parquet [1], Capo 10.2 [16] and mPL6 [6]) for our floorplanning input. Both Parquet and Capo 10.2 produced adequate floorplans but exhibited large run-to-run wire length variability (about 20% in the several tested cases) due to the randomized algorithms used. The analytical placer mPL6 produced stable results but its floorplans had significantly longer wire lengths.

The high variability in wire lengths makes the comparison of two topologies difficult as one may have lower energy because of a better floorplan run. We overcame the inherent instability of the floorplanning algorithm, by picking the best floorplan out of multiple runs (about 100). To reduce the runtime, we parallelize the multiple runs over multiple processors with a near linear speed up in processing time.

4. TOPOLOGY GENERATION ALGORITHM

Our topology generation algorithm consists of two distinct phases: an initial topology creation phase and a refinement phase shown in Figure 3. In the first phase, we create a floorplan based on the application volume graph (Step 1). The pin locations from the floorplan are used to group together frequently communicating cores into partitions (Step 2). We use the term partition or cluster interchangeably to describe processing elements connected to the same router. To connect the routers in the initial partition, we use a modified route shortest path algorithm similar to that presented in [11] (This will be explained in Section 4.2). The first three steps form an initial topology that can be refined quickly towards the optimized solution. It is possible to substitute this stage with a known initial topology such as a mesh or other known custom topology.

In the second phase, the topology is refined through three steps: (i) coarse partition refinement, (ii) fine partition refinement and (iii) route refinement. The coarse partition refinement step evolves the topology by changing the number of routers or aggressively swapping groups of processing elements across routers or adding point-to-point links (Step 4). It transforms the partitions by increasing/decreasing the number of routers by merging smaller partitions, splitting large partitions and moving groups of processing elements between partitions. Figure 4 shows the scaling trends of the input and output ports for dynamic and static energy. Static energy increases linearly with input ports at about 0.1 mW per router port for a 4-flit buffer due to an increase in buffering. The merge operation is typically used to combine two routers to reduce their buffer resources. A split operation breaks large partitions into two smaller independent pieces which results in lower dynamic energy costs as there are fewer router ports. In Step 5, the fine partition refinement step moves single routers to improve the partitioning. The route refinement procedure (Step 6) modifies the links between the routers to either decrease dynamic energy by reducing the number of links or reduces hop energy by adding profitable links.

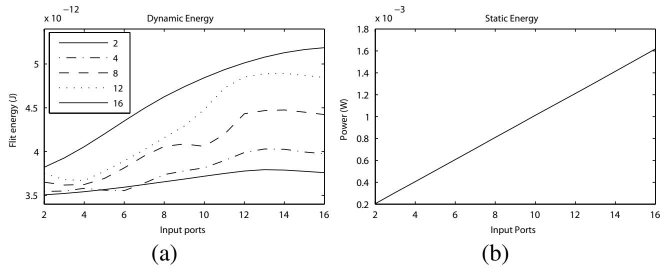


Figure 4: Port scaling of dynamic energy

Each design is evaluated using the cost function in Equation 1. The topology is floorplanned and traffic flows are mapped to the topology. The paths taken by each traffic stream can be determined using a modified shortest path algorithm on every processing element in the topology graph with edges weighted based on their dynamic energy and wire energy. Once shortest paths are determined, the total NoC energy is computed from the edge volumes. To support the volume constraint, each traffic flow is mapped to a commodity in the multi-commodity min-cost flow (MCMCF) problem with capacity constraints. We have used the linear programming based MCMCF solver PPRN [3] to determine the shortest paths. If the volume capacity constraint on an edge is not met, the candidate topology is rejected. Deadlock prevention is implementable using turn prohibition methods[20]. The prohibited turns can be implement using the MCMCF solver. Due to space constraints, we are unable to discuss this in detail.

We use tabu lists to prohibit the same sequence of moves from repeating. Separate tabu lists are maintained for each of the possible moves (split, merge, split-move, move, add and remove). Some tabu lists are cleared when a split and merge occurs as the nodes no longer correspond to the nodes in the tabu list. In the route refinement and fine partition refinement steps, we maintain an unchanged threshold count $U_{threshold}$ to decrease the algorithm run-time when the current set of moves is making no improvement to the topology.

4.1 Forming the initial topology

The initial topology is formed by performing an activity weighted floorplan based on the application graph [7]. The network interfaces of frequently communicating cores will be clustered together by the floorplanner as it attempts to minimize wire length. The pin locations

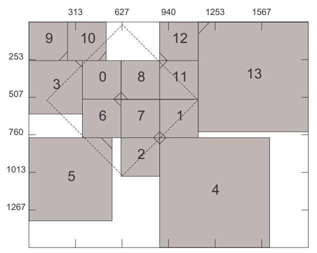


Figure 5: Activity weighted floorplan without routers

are assigned to be the corner of the rectangular cores as shown in Figure 5. For each unique set of pin locations, we group together pins reachable within the distant constraint D. The diamond in the figure shows the pins reachable within 0.5mm (manhattan distance) of the clustered network interfaces for cores 0, 6 and 8. We form partitions greedily by grouping the pins with the most intra-cluster volume. In the diagram, the cluster with the highest intra-cluster traffic was centered around cores 0, 6 and 8 which includes cores 0, 1, 2, 3, 4, 5, 6, 8, 7 and 10. All these cores are initially connected on a single router. The next iteration groups 11, 12, 13 and final iteration groups core 9 onto a separate router.

From experiments, we found that setting the distance constraint to the height of the largest core produces a good initial partitioning for most of the benchmarks tested. A distance constraint that is set too high under-partitions the topology, resulting in large routers. On the other hand, a small distance constraint will create small partitions and many routers. Neither pose a problem because the refinement procedure quickly merges the small partitions or splits the large partitions.

4.2 Route assignment

The purpose of the route assignment procedure is to add links to the topology graph to connect routers and provide more direct paths, thus reducing hop count. A weighted fully connected graph with $|R|$ nodes is used to determine the shortest paths for each flow, where $|R|$ is the number of routers in the topology. Each edge that already has a direct link between two routers is weighted by the cost of transferring one packet of data (dynamic energy cost). Edges that do not currently exist in the topology graph include an additional installation cost, i.e., the sum of the dynamic energy cost of transferring the packet on the larger router; the increase in cost for other traffic on both source and destination router; and the static energy for lifetime of the application. Each communication stream is mapped to communication paths using a shortest path algorithm to determine whether it is better to use existing routes or add additional routes to connect that traffic flow. installation costs are minimized by assigning highest communication traffic streams first.

Algorithm 1 Coarse Partition Refinement

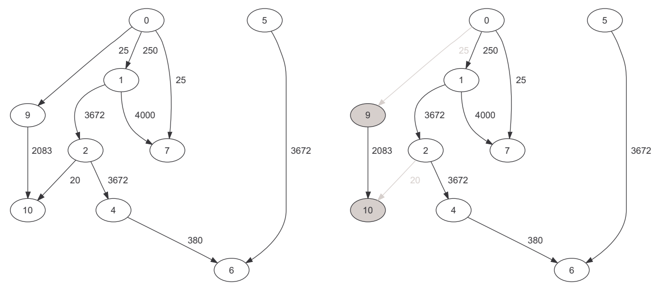
```
1: while merge, split and split-move candidates exist do
2:     Attempt best merge, split, split-move or multi-merge
3:     Perform fine partition refinement (return refined $T$)
4:     if last merge/split or split-move improves on topology then
5:         Accept refined topology $T$, adjust tabu list
6:     else
7:         Add that merge/split/split-move to the tabu list
8: return topology
```

4.3 Coarse partition refinement

The pseudo code for the partition refinement step is shown in Algorithm 1. We use suitability functions to evaluate the benefit of the each potential merge, split and split-move operation. The method used to evaluate merge, split and split-move will be described in the next three subsections. After each potential move is evaluated, it is ranked and the best move is chosen.

4.3.1 Selecting a merge candidate

The merge gain is evaluated by estimating the change in router dynamic, static and leakage energies for each pair of merge candidates. There are three contributing factors that affect the suitability of a merge: (i) static energy reduction due to the removal of common ports (ii) decreased volume due to reduction in hop distance and (iii) increase in



(a) Application subgraph (b) 1st split partition

Figure 6: Split partition example

packet energy due to more router ports. The decrease in volume can be estimated by summing the traffic between the two partitions. The increase in dynamic energy is computed by estimating the cost before and after the merge.

Any pair of router nodes that has a positive merge gain is added to a list to be considered for merging by the coarse refinement algorithm. The coarse partition refinement algorithm will attempt to merge multiple routers at once, if they are beneficial. New links may need to be added to ensure the connection of all streams. In these cases, the route assignment procedure is used to route the unconnected traffic streams. It is also possible to perform two merges simultaneously where they are likely to be beneficial.

Algorithm 2 Get Split Candidates

```
1: for each router partition $p \in P$ do
2:     Construct application subgraph $A'$ from $p$
3:     Add edges in subgraph $A'$ to edge list $EL$
4:     Sort edge list $EL$ by communication volume (ascending order)
5:     while partition $p$ can be split further do
6:         Split N Partition($A'$, $EL$)
7:         if Gain($p$) $>$ split threshold, add to split candidates
8: return split candidates
```

4.3.2 Selecting a split candidate

The split is the opposite operation to the merge operation. Similar to the merge case, the split gain is evaluated by estimating the change in router dynamic, static and leakage energies. However, for each partition there may be multiple ways to split the nodes. For example breaking a large router such as a sixteen port router into two eight port routers halves the per packet dynamic energy (See Figure 4(a)). Conversely, splits should be avoided in cases that would result in a large amount of traffic between the newly formed partitions. Given a router and an assignment of nodes into two partitions, the potential gain can be computed in a similar fashion as the merge candidates. The potential gain/loss due to a split operation can be analytically calculated by examining: (i) the increase in total network volume between the newly formed routers; (ii) the decrease in dynamic energy due to smaller routers and (iii) the change in static energy. A split is added to the list of candidates when the potential gain is above the split threshold. The split threshold is set to a slightly negative value so as to include split candidates that increase router energy but decreases wire energy through greater placement freedom.

The router can be split in two partitions in multiple ways. We use a greedy edge removal strategy, which is presented in lines (2 - 8) of Algorithm 2 to generate several splits of the router partition.

For each partition p, construct a subgraph A' of the original application volume graph containing only the nodes and edges in the partition. Next, create edge list EL containing all the traffic flows/edges of subgraph A', sorted in ascending order by volume (Algorithm 2 Line 4) and iteratively remove the minimum volume edge from the application subgraph until two unconnected partitions are formed. A depth first search is used to determine whether two partitions are formed. If two or more partitions are formed, we evaluate the split gain as described above. To find the next split, the last removed edge is added back to the application volume graph and edges are removed until another split partition is found. Each additional partition found by splitting will find partitions with increasing inter-partition communication costs.

An example application subgraph from the *263decmp3dec* application in shown in Figure 6(a). In this example, the processing elements

{0, 1, 2, 4, 5, 6, 7, 9 and 10} are connected to the same router. If we remove the smallest edges {20, 25} as shown in Figure 6(b), two partitions {9, 10} and {0, 1, 2, 4, 5, 6, 7} are formed. The next partition is formed by adding back the 0 to 9 edge and removing the next two smallest edges with volumes 25 and 250. The next partition formed is {0, 9, 10} and {1, 2, 4, 5, 6, 7}. This procedure continues until no more partitions can be formed.

The split-move is a variant of the split with the newly created split router merged with another partition. The procedure for a split-move is similar to both the split and merge methods, except the cost function considers both split and merge simultaneously. Due to space constraints, we do not describe this suitability function.

4.3.3 Point-to-point networks

When there are no more split and merge candidates left, each edge in the application graph is analyzed to determine which traffic streams would benefit from point-to-point links. Point-to-point links are added late in the design space exploration phase, to avoid prematurely adding direct links which would artificially improve a poorly designed initial topology. Two optimization passes are performed to discover potential point-to-point links. The first pass evaluates the effects of adding point-to-point links to replace streams in the original application graph. The second pass evaluates the gain or losses from disconnecting an entire node from the network by converting all of its communication into point-to-point links.

Evaluation is performed by comparing the energy cost of routing the stream through the network and by a point-to-point link. Adding a point-to-point link, will add a static installation cost which includes the static and leakage energy of network interfaces. By adding one or more point-to-point links, the network interfaces or router links may become redundant and hence can be removed. These are evaluated as part of the cost function. The wire length in point-to-point networks is evaluated by considering the distance from the two closest sides of communicating cores.

Algorithm 3 Fine Partition Refinement(T)

```
1: while move candidates != $\emptyset$ and $u < U_{threshold}$ do
2:     Attempt best move candidate
3:     Perform route refinement (return refined $T$)
4:     if Last move improves topology then
5:         Accept refined topology $T$
6:     else
7:         Add Last move to the tabu list, increment $u$
8: return topology
```

4.4 Fine partition refinement

The fine partition refinement step (Algorithm 3) moves processing elements between partitions to reduce the size of large routers and/or reduces the hop count by migrating processing elements nearer to their communicating partners. Similar to the coarse partitioning refinement step, processing element - router pairs are evaluated based on a suitability function.

The suitability function for moving a processing element determines the gain/loss due to router sizing and hop gain. A function is used to determine the best existing router to move a processing element to. The move is attempted and the routes refinement step is executed. The route-refined topology is evaluated for the energy cost. If the move improves on the best topology so far, it is accepted and new move processing element candidates are computed. If the move does not improve the design, it is rejected and removed from the candidate list and prohibited from future move candidate lists. When there are no more move processing element candidates, the best topology found will be returned.

4.5 Refining routes

Routes are improved by adding new links to provide more direct routes or removing existing links that are infrequently used. Addition of new links is not always profitable as there is a significant cost for opening up a channel between two routers.

Additional router-to-router links are evaluated to determine an estimate of the energy effects. For a link addition to be beneficial, the network volume reduction must be great enough to justify the addition of ports between the two routers. A volume threshold criteria is used to filter out candidates whose network volume reduction is below the installation cost. The addition of some links may cause others to become redundant. Where a redundant link is removed, the installation cost of the new link is not considered.

Component	Energy/Power
NI src dynamic	1.6e-12 J
NI target dynamic	1.2e-12 J
NI src static	0.16 mW
NI target static	0.16 mW

Table 1: Network interface power

The candidate edges for addition are ranked according to their potential benefit to the topology, i.e., how much bandwidth is reduced. We use a lottery system to decide which of the candidate edges will be added. The number of tickets allotted to an edge is equal to network volume that is decreased. Edges that reduce greater bandwidth are given more, while the worst edge receives no tickets (edges that do not reduce total volume in the network). A random ticket is selected and that edge addition is evaluated for energy improvements. Removal of edges is done in a similar fashion to addition. We prohibit the removal of routes that would cause the traffic requirements of the application to be violated.

4.6 Supporting additional constraints

Additional architectural constraints such as a maximum router port constraints can be added to ensure that the topology is implementable. A maximum router port constraint restricts the number of ports in any router in the topology. This requires the selection criteria for merge, move processing element, and add to be modified to filter out candidates that would violate this constraint. A swap processing element operation is also added to allow local partition refinement when router partitions approach the maximum port constraint. In a swap, pairs of moves are considered such that they do not increase the number of router ports past the port constraint.

5. EXPERIMENTAL RESULTS

We conducted experiments to evaluate our topology generation algorithm against the min-cut based algorithm presented by Murali et al. [11]. To allow fair comparison, we re-implement their algorithm using our energy models and the hMetis partitioner [10].

The topology generation algorithm was run on a 2.0 GHz Pentium-M processor. The floorplans were distributed on a cluster of 3 $\times$ four-core Opteron 2.0 GHz servers running the Capo 10.2 placer with the block flipping and rotation option enabled. The chip dimensions were selected for each benchmark relative to the total sum of the core sizes. A fixed capacitance of 500 fF per mm for each bus line is set. Point-to-point network interfaces are located on the perimeter of the core and have a minimum net length of 0.1mm. We set the optimizer to use energy models for 250 MHz operation. In most benchmarks, the core sizes varied between $62500\mu m^2$ to $4mm^2$. We select the best floorplan of one hundred runs of the final topology comparison purposes. The network interface power consumption is shown in Table 1.

5.1 Experiments Conducted

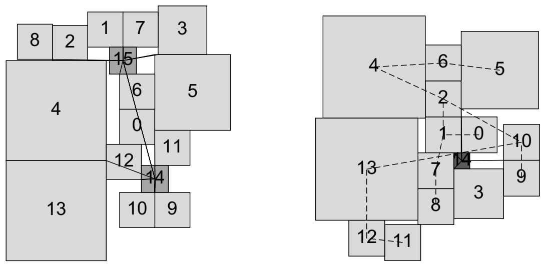
	Graph	PE	Flows	k flits	t (cycles)
G1	263decmp3dec	14	16	23.56	20000
G2	263encmp3dec	12	12	230.2	200000
G3	mp3encmp3dec	13	12	16.52	20000
G4	vopd	13	12	3.116	3000
G5	imp	27	96	11040	1600000
G6	large	14	16	23.56	20000
G7	long	14	16	23.56	100000
G8	broadcast	16	120	120.0	50000
G9	mesh4x4	16	40	48.00	30000
G10	random-a	100	200	283.0	50000

Table 2: Benchmark Description

Table 2 summarizes the characteristics of the ten benchmarks graphs. The first seven benchmarks (G1-G7) are application volume graphs obtained from [8, 19, 11] which contain a mixture of MP3, H263 decoders and encoders, video object plane decoders and image processors with varying cores sizes. The image processing benchmark *imp*, features twelve processors with private memories and three shared resources with are used equally by each processor. The *large* and *long*

Graph	Mincut					NoCOUT w/o ptp					NoCOUT					Impr
	$\|R\|$	$\|L\|$	Hops	P(mW)	CPU	$\|R\|$	$\|L\|$	Hops	P(mW)	CPU	$\|R\|$	$\|L\|$	Hops	P(mW)	CPU	%
G1	2	23	1.003	7.45	50	4	24	1.068	7.38	130	1	17	0.012	6.01	150	19
G2	2	18	1.000	6.50	40	2	18	1.001	6.50	16	0	12	0.000	4.93	20	24
G3	2	21	1.010	6.43	37	2	22	1.001	6.45	160	1	14	0.020	4.90	310	24
G4	2	23	1.008	7.46	37	3	24	1.110	7.39	200	0	12	0.000	5.14	280	31
G5	8	68	1.236	42.5	200	13	78	1.290	39.5	1050	1	58	0.121	26.6	1200	37
G6	3	24	1.031	7.79	51	3	23	1.026	7.69	165	1	17	0.012	6.24	375	20
G7	1	21	1.000	5.05	50	1	21	1.000	5.05	40	2	19	0.786	4.70	55	7
G8	1	30	1.000	24.0	50	1	30	1.000	24.0	350	1	30	1.000	24.0	375	0
G9	2	34	1.356	15.5	50	2	34	1.356	15.5	200	2	38	1.002	15.5	220	3
G10	20	215	1.389	75.0	250	14	218	1.222	73.5	1500	15	224	0.627	66.5	1800	11

Table 3: Energy consumption of Min-cut and NoCOUT



(a) Mincut (b) NoCOUT with point-to-point

Figure 7: 263decmp3dec floorplans

benchmarks are identical to *263decmp3dec* except that the core area is increased by a factor of nine in G6 and the total application execution time is increased in the *long* benchmark. These two benchmarks illustrate the effect of increasing the relative wire energy, and operating at lower activity rates.

Graphs G8 and G9 show mesh and broadcast type communication patterns used in scientific applications. In the broadcast benchmark, all cores communicate with each other. We generated a synthetic asymmetric volume graph (Graphs G10) with one hundred cores to test the scalability of the algorithms. We control the number of in-edges, out-edges and the distribution of volumes of edges and core sizes.

5.2 Results

Table 3 reports the number of routers $|R|$, links $|L|$, average number of hops, power consumption and algorithm runtime (in seconds) of both the min-cut and NoCOUT algorithms. For the SoC benchmarks, our algorithm generated designs with a mixture of routers and point-to-point networks. Figure 7 shows two floorplan generated by the Mincut algorithm and NoCOUT for the *263decmp3enc* benchmark. Most of the traffic streams use point-to-point links (shown with the dashed lines connecting the centers of cores) to communicate with their neighboring cores resulting in shorter wire lengths. Point-to-point networks were beneficial in these benchmarks (G1-G7) because the cores communicated in a mostly pipelined fashion with an in-degree and out-degree close to one. The *263encmp3dec* and *vopd* applications created topologies containing only point-to-point connections. If we disable the point-to-point generation in our algorithm, it produces designs which are similar to the min-cut solution with small improvements in energy up to 6%. In some cases, we are able to produce a better partition of the router nodes that takes into account energy effects of power/energy model.

For the broadcast benchmark, a single 16-port router was created by both algorithms. As there were many communication partners, no individual stream benefited greatly from point-to-point connections, hence the algorithm correctly chose not to add any. For the *mesh4x4* benchmark, both algorithms created a two router network with half of the mesh nodes connected to each router. Point-to-point links were added between nodes 1 and 2, and 14 and 15, as these reduced the wire energy significantly to justify the extra network interface.

5.2.1 Runtime and Scalability of algorithms

In the benchmarks presented above, the number of generations needed to produce the final solution is typically limited to about 100. Each generation can take about five to ten seconds to evaluate due to the need to floorplan every design many times. The complete NoC topology solution is generated in minutes. This runtime can be further reduced with the use of a larger cluster running floorplanning problems. In future, we plan to investigate analytical floorplanning techniques that will remove the need to run the floorplanner multiple times. Although the min-cut partitioner is more efficient at producing solutions, as more parameters such as link sizes are added in, it requires the min-cut procedure to be run in the inner loop of any optimization algorithm. We believe our method of exploring the design space iteratively will be competitive in runtime when other parameters are added and constrained further. In very large cases, for example greater than 1000 cores, multi-level partitioning approaches similar to that used in floorplanning should be considered. We found that in the fine and route refinement stages, the average processor-to-router and router-to-router wire lengths remain relatively consistent. It may be possible to estimate the potential gain in these stages without running the floorplanning for every design iteration, reducing the number of runs.

6. CONCLUSIONS

The topology generation problem contains multiple NP-hard problems. These problems are often treated separately and solved using approximation algorithms. In this paper, we presented a technique for the generation of an energy efficient application specific NoC topology with awareness of floorplan and characterized energy and area models. We show that using a guided local search, it is possible to find solutions of similar quality to a min-cut partitioner on various benchmarks. Our algorithm allows a mixture of both point-to-point networks and packet-switched networks to allow further reduction in energy consumption through shorter point-to-point wires and fewer routers.

References

[1] S. N. Adya and I. L. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on VLSI*, 2003.
[2] T. Ahonen, D. A. Sigenza-Tortosa, H. Bin, and J. Nurmi. Topology optimization for application-specific networks-on-chip. In *SLIP*, 2004.
[3] J. Castro and N. Nabona. An implementation of linear and nonlinear multicommodity network flows. *European Journal of Operational Research*, 1996.
[4] J. Chan and S. Parameswaran. NoCGEN: A Template Based Reuse Methodology for Networks on Chip Architecture. In *VLSI Design*, 2004.
[5] J. Chan and S. Parameswaran. NoCEE: energy macro-model extraction methodology for network on chip routers. In *ICCAD*, pages 254–259, 2005.
[6] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. mPL6: enhanced multilevel mixed-size placement. In *ISPD*, 2006.
[7] J. Hu, Y. Deng, and R. Marculescu. System-level point-to-point communication synthesis using floorplanning information. In *ASP-DAC/VLSI*, January 2002.
[8] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *ASP-DAC*, January 2003.
[9] Y. Hu et al. Physical synthesis of energy-efficient networks-on-chip through topology exploration and wire style optimization. In *ICCD*, 2005.
[10] G. Karypis and V. Kumar. Multilevel k -way hypergraph partitioning. In *DAC*, 1999.
[11] S. Murali et al. Designing application-specific networks on chip with floorplan information. In *ICCAD*, 2006.
[12] S. Murali and G. D. Micheli. SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs. In *DAC*, pages 914–919, 2004.
[13] Ü. Y. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In *ICCAD*, pages 246–253, 2005.
[14] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli. Synthesis of on-chip interconnection structures: From point-to-point links to networks-on-chip. Technical Report UCB/EECS-2006-147, UC Berkeley, 2006.
[15] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Efficient synthesis of networks on chip. In *ICCD*, 2003.
[16] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut floorplacement. *IEEE Trans. on CAD*, 25(7):1313–1326, July 2006.
[17] K. Srinivasan and K. S. Chatha. A low complexity heuristic for design of custom network-on-chip architectures. In *DATE*, 2006.
[18] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures. In *ICCD*, 2004.
[19] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Trans. VLSI Syst.*, 2006.
[20] D. Starobinski, M. Karpovsky, and L. A. Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Trans. Netw.*, 2003.