Application-Specific Network-on-Chip Architecture Synthesis based on Set Partitions and Steiner Trees

Shan Yan and Bill Lin
Department of ECE, University of California, San Diego
La Jolla, CA, 92093
Email: shyan@ucsd.edu, billlin@ece.ucsd.edu

Abstract— This paper considers the problem of synthesizing application-specific Network-on-Chip (NoC) architectures. We propose two heuristic algorithms called CLUSTER and DECOMPOSE that can systematically examine different set partitions of communication flows, and we propose Rectilinear-Steiner-Tree (RST) based algorithms for generating an efficient network topology for each group in the partition. Different evaluation functions in fitting with the implementation backend and the corresponding implementation technology can be incorporated into our solution framework to evaluate the implementation cost of the set partitions and RST topologies generated. In particular, we experimented with an implementation cost model based on the power consumption parameters of a 70nm process technology where leakage power is a major source of energy consumption. Experimental results on a variety of NoC benchmarks showed that our synthesis results can on average achieve a 6.92× reduction in power consumption over the best standard mesh implementation. To further gauge the effectiveness of our heuristic algorithms, we also implemented an exact algorithm that enumerates all distinct set partitions. For the benchmarks where exact results could be obtained, our CLUSTER and DECOMPOSE algorithms on average can achieve results within 1% and 2% of exact results, with execution times all under 1 second whereas the exact algorithms took as much as 4.5 hours.

I. Introduction

Network-on-Chip (NoC) architectures have been proposed as a scalable solution to the global communication challenges in nanoscale SoC designs [1, 2]. They can be designed as regular or application-specific network topologies. Regular topologies have been successfully employed in a number of tile-based chip-multiprocessor projects, e.g. [15, 16], which are appropriate because of processor homogeneity and application traffic variability. On the other hand, for custom SoC applications, the design challenges are different in terms of varied core sizes, irregularly spread core locations, and different communication bandwidth requirements. Therefore, an application-specific network architecture customized to the needs of the application is more appropriate. This synthesis problem is the focus of this paper.

The NoC synthesis problem is challenging for a number of reasons. First, for a large complex SoC design, an optimal solution will likely involve multiple networks since each core will likely communicate only with a small subset of cores. Therefore, a single network that spans all nodes is often unnecessary. Part of the synthesis problem is to partition the set of specified communication flows into subsets and derive a separate optimal physical network topology for each subset. In general, flows may be grouped together even though they don't share common sources or destinations because they may be able to beneficially share common intermediate network resources. Second, besides deciding on the set partition, our synthesis problem must also decide on the physical network topology of each group in the set partition. Finally, depending on the optimization goals and the implementation backend, the appropriate cost function may be quite complex. In particular, in this paper, we consider the power minimization problem that considers both leakage power and dynamic switching power. It is well-known that leakage power is becoming increasingly dominating [10, 12]. Therefore, it is important to properly account for leakage power when adding routers and channels to the synthesized architecture. However, when considering leakage power, the cost function may need to account for possibly discrete as well as nonlinear cost increments of links and routers whereas dynamic switching power may be best modeled as a function of cumulative data rates. Other optimization goals may include minimizing hop-counts along with power minimization.

In this paper, we describe two heuristic algorithms called CLUSTER and DECOMPOSE that systematically examine different set partitions of communication flows. For each set partition considered, we use well-developed Rectilinear-Steiner-Tree (RST) algorithms [18–20] to generate a physical network topology for each group in the set partition. Though the RST problem is in itself NP-hard, well-developed fast RST algorithms are available that can be effectively used, as indicated by the run-times presented in Section VII. For each RST derived, the routes for the corresponding flows and the bandwidth requirements for the corresponding network channels are determined. We then use the specified evaluation function to evaluate the implementation cost of the corresponding set partition and RST-generated physical topologies. At the end of each algorithm, the best solution found is returned. Although we use Steiner trees to generate a physical network topology for each group in the set partition, the final NoC architecture synthesized is not necessarily limited to just trees as RST implementations of different groups may be connected to each other to form non-tree structures.

For the rest of the paper, Section II outlines related work. Section III presents the problem description and our formulation. Sections IV and V describe the CLUSTER and DECOMPOSE algorithms respectively. The router merging algorithm for further performance improvement is discussed in

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

Section VI. The experimental results and the conclusion are presented in Section VII, and VIII respectively.

II. Related Work

The work presented in [9] addresses the complementary problem of providing custom network architecture instantiations. Other existing NoC solutions assume a regular mesh-based NoC architecture [3,4], and their focus is on the mapping problem. On the problem of designing application-specific NoC architectures without assuming an existing interconnection network architectures, several techniques have been proposed [5–8]. The optimization method presented in [5] is primarily concerned with providing connectivity under node degree constraints, but it does not consider the dimensioning of links or routers and their implementation and power costs. In [6], techniques were presented for the constraint-driven communication architecture synthesis of point-to-point links by using heuristic-based k-way merging. However, their technique is limited to topologies with specific structures that have only two routers between each source and sink pair. In [7], novel NoC synthesis algorithms were presented, but their solutions only consider topologies based on a slicing structure where router locations are restricted to corners of cores and links run around cores. In addition, as stated in [7], their power minimization problem is one of minimizing the total traffic flowing through the routers of the derived NoC architecture, which corresponds well to process technologies where dynamic switching power dominates, but not necessarily when leakage power is substantial. In [8], an innovative NoC synthesis flow was presented with detailed backend integration. Their method is based on the min-cut partitioning of cores to routers.

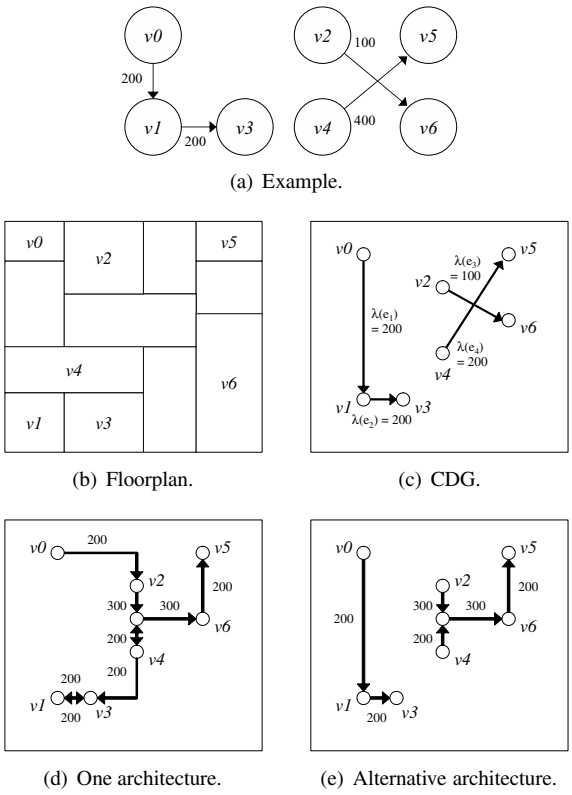
III. Problem Description and Formulation

A. Description

Figure 1 illustrates an example of our application-specific NoC architecture synthesis problem. Consider the small example specification shown in Figure 1(a) where the nodes represent cores, edges represent communication flows, and edge labels represent the bandwidth requirements for the corresponding flows. In our design flow, an initial floorplanning step is performed in advance of NoC synthesis to obtain a placement of the cores. An example floorplan is shown in Figure 1(b).

In general, floorplanning solutions do not have to follow a slicing structure, and the state-of-the-art floorplanning methods allow for non-slicing floorplans to achieve more efficient solutions. The only requirement is that the cores are non-overlapping. Also, in our problem definition, modules in a design do not necessarily have to be attached to the on-chip network. Modules can also be connected by means of conventional routing of interconnects, as shown in the un-labeled rectangles in Figure 1(b). The floorplanning problem has been extensively studied with many well-developed solutions (e.g. [21–24]). Recent work has considered modifications to floorplanning metrics in the context of NoC synthesis [7].

After floorplanning, the (x, y) coordinates of the communication ports of the cores are known, and they are provided as input to our synthesis problem. In addition, the communication bandwidth requirement for each traffic flow is also provided, as



(a) Example. (b) Floorplan. (c) CDG. (d) One architecture. (e) Alternative architecture.

Fig. 1. Illustration of the NoC synthesis problem.

shown in Figure 1(c). Specifically, the input to our synthesis problem is a directed graph $G(V, E, \pi, \lambda)$, called a communication demand graph (CDG), where each node $v_i \in V$ is associated with a communication port of a core, and each directed edge $e_k = v_i \rightarrow v_j \in E$ represents a communication flow from v_i to v_j. The position of each node v_i is given by $\pi(v_i) = (x_i, y_i)$. The bandwidth requirement for each communication flow e_k is given by $\lambda(e_k)$. Based on the optimization goals and cost functions specified by the users, the output of our NoC architecture synthesis problem is a optimized custom network topology with minimum cost with pre-determined routes for the specified communication flows on the network such that the bandwidth requirements are satisfied.

For example, Figures 1(d) and 1(e) show two different topologies for the CDG shown in Figure 1(c). Figure 1(d) shows a network topology that interconnects all the nodes in the system. In this topology, the pre-determined route for flow e_1 traverses through node v_0, v_2, v_4, v_3, and v_1. Figure 1(e) shows an alternative topology comprising of two separate networks. Here, flow e_1 is simply transferred over the network channel from v_0 to v_1 with the corresponding dimensioning.

After an optimized NoC architecture has been synthesized with the corresponding routers and links, the floorplanning step can be invoked again to update the placement of cores and communication ports, and the NoC can be resynthesized with the updated floorplanning information. As shown experimentally in Section VII, our NoC synthesis algorithms are fast, making it possible to iterate NoC synthesis with floorplanning.

B. Formulation

In general, the solution space of possible application-specific network architectures is quite large. Depending on the commu-

nication demand requirements of the specific application under consideration, the best network architecture may indeed be comprised of multiple networks. However, the decision on the number of networks and the partitioning of communication of flows among them is not a simple question of partitioning communication flows into groups where the corresponding sets of communication ports are disjoint. Consider again the example depicted in Figure 1(c). The communication flows e_3 and e_4 have disjoint communication ports. However, it may still be best to group them together on the same network, as depicted in Figure 1(e), because both flows must travel a common distance in the horizontal direction, and they are able to share the cost of the network channel that spans the horizontal distance.

In general, the solution space of distinct set partitions of n flows, commonly known as the n^{th} Bell number, is known to grow $\Theta(n \log n)^n$ [17]. B_n grows rapidly, with e.g. B_{10}, B_{11}, and B_{12} equal to 115975, 678570, and 4213597, respectively, and so on. The goal of the heuristic algorithms CLUSTER and DECOMPOSE presented in Sections IV and V, is to significantly reduce the number of set partitions needing to be examined in a systematic manner.

Besides deciding on the set partitioning of flows, physical network topologies must be decided for carrying the communication flows. In current process technologies, layout rules for implementing wires dictate physical topologies where the network channels run horizontally or vertically. Thus, the problem is similar to Rectilinear Steiner Tree (RST) problem. Given a set of nodes, the RST problem is to find a network with the shortest length using horizontal and vertical edges such that all nodes are interconnected. The RST problem is well-studied [18–20] with very fast implementations available [19, 20]. Figures 1(d) and 1(e) show possible RSTs for the set partitions $\{\{1, 2, 3, 4\}\}$ and $\{\{1, 2\}, \{3, 4\}\}$, respectively. We use an RST solver in the inner loop of our heuristic algorithms to generate topologies for the set partitions considered. RSTs are only re-evaluated for the the portion of the set partition that changed at each step, and previously computed RSTs can be cached.

After a physical topology is generated for each group in a set partition, the routes for the corresponding flows and the bandwidth requirements for the corresponding network channels can be readily derived. The routes for the flows follow directly from the tree structure of the RST solution. Correspondingly, the cumulative bandwidth requirements along the edges also follow. Routers are allocated at junctions where either flows from multiple channels must be multiplexed to the same outgoing channel or flows from the same channel must be de-multiplexed to multiple outgoing channels. We can also consider merging of "nearby" routers if merging them will lead to a reduction in cost (this is further discussed in Section VI).

Our RST formulation provides us a way to generate physical topologies. However, depending on the optimization goals and the implementation backend, the appropriate evaluation function may be quite a bit more sophisticated than the total length metric used in RST algorithms. To facilitate different objective functions and implementation backends, the evaluation function can be provided as an input to our algorithms. For example, in this paper, we investigate the NoC synthesis problem with the objective to minimize the power consumption, including both the leakage power and dynamic switching power, as well as satisfy the performance constraints. Other optimization goals may include minimizing hop-counts along with power minimization.

IV. CLUSTER

In this section, we present an algorithm called CLUSTER that reduces the number of set partitions considered from $\Theta(n \log n)^n$ to $\Theta(n^3)$, which is a significantly smaller subset of set partitions. The details of the algorithm is shown in Algorithm 1. The CLUSTER algorithm takes a communication demand graph and an evaluation function as input and generates an optimized network architecture implementation details as output. The algorithm starts by implementing each edge in the communication demand graph separately. The solution for each single edge is a simple RST connecting two terminals. It sets these single edges as the initial set partition, denoted as $P^0 = \{\{e_1\}, \{e_2\}, \ldots, \{e_n\}\}$, as shown in lines 2-5.

Then, in lines 7-18, at each iteration, the algorithm systematically generates new candidate set partitions starting from the set partition chosen from the previous iteration. In particular, in the first iteration, the algorithm starts with the initial set partition $P^0 = \{\{e_1\}, \{e_2\}, \ldots, \{e_n\}\}$ with n groups, each group containing exactly one edge. Then, in lines 8-13, the algorithm generates new candidate set partitions from P^0 by considering all pairwise mergings of groups in P^0. The groups are denoted as g_u and g_v in the pseudo-code. For each pairwise merging considered, an RST solver is called to generate a physical network topology for the merged set of flows, and the cost of this network is calculated using the specified evaluation function C. We do not need to solve an RST problem for the entire set of flows, just the subset of flows in the merged groups is considered. We then, in line 12, compute the total cost of the resulting set partition by summarizing the cost of implementing all the other sets using their own networks. In lines 15-16, we select the merging that achieves the best cost in this iteration and choose it as P^1. In general, we start from the chosen set partition, P^t, from the iteration to generate pairwise mergings of groups from P^t, and the best merging is selected as the new chosen set partition P^{t+1}. At each iteration, the number of groups that need to be considered is reduced by 1, but the size of groups will become increasingly larger. Finally, in the last iteration, we only need to consider the mergings of two groups.

At each iteration in lines 7-18, we maintain the chosen set partition and the associated cost calculations for that iteration. Then, in the end of the algorithm, lines 23-24, we choose the set partition with the minimum cost. Since at each iteration t, there can be at most $(n - t)(n - t - 1)/2$ possible pairwise group mergings, and there are $(n - 1)$ iterations, the number of set partitions considered in the CLUSTER algorithm is $\Theta(n^3)$.

V. DECOMPOSE

The DECOMPOSE algorithm described in this section reduces the number of set partitions considered from $\Theta(n \log n)^n$ to $\Theta(n^2)$. The details of the algorithm is shown in Algorithm 2. This algorithm works in the opposite direction as CLUSTER when generating candidate set partitions and the corresponding RST topologies. It starts by considering all communication demands as a single cluster. In each iteration, the algorithm

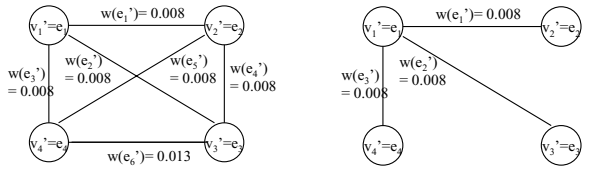
Algorithm 1 CLUSTER $(G(V, E, \pi, \lambda), C, T)$

Input: $G(V, E, \pi, \lambda)$: communication demand graph
C: specified evaluation function for implementation cost
Output: T: synthesized network architecture

```
1: initialize P^0 = ∅
2: for all e_k ∈ E do
3:     P^0 = P^0 ∪ {e_k}
4:     cost({e_k}) = EvaluateCost(T({e_k}), C)
5: end for
6: t = 0
7: while |P^t| > 1 do
8:     for all g_u, g_v ∈ P^t do
9:         g_uv = g_u ∪ g_v
10:        T(g_uv) = SolveRST(g_uv)
11:        cost(g_uv) = EvaluateCost(T(g_uv), C)
12:        β(g_u, g_v) = cost(g_uv) + Σ_{g_i ∈ P^t, g_i ≠ g_u, g_v} cost(g_i)
13:    end for
14:    (u, v) = arg min_{g_u, g_v ∈ P^t} β(g_u, g_v)
15:    P^{t+1} = P^t \ {g_u, g_v}
16:    P^{t+1} = P^{t+1} ∪ {g_u ∪ g_v}
17:    t = t + 1
18: end while
19: for all t ∈ [0, n − 1] do
20:    c(P^t) = Σ_{g_u ∈ P^t} cost(g_u)
21:    soln[P^t] = ⋃_{g_u ∈ P^t} T(g_u)
22: end for
23: t = arg min_{t ∈ [0, n−1]} c(P^t)
24: T = soln[P^t]
25: return T
```

considers different ways of breaking up an existing group in the set partition chosen from the previous iteration into two smaller ones. Then, the differential cost of splitting a group is evaluated by generating an RST for each sub-group and evaluating their costs using the specified evaluation function. To facilitate this decomposition process, two important graphs are used in DECOMPOSE: Affinity Graph (AG) and its Minimum Spanning Tree (MST). The affinity graph A is built by associating each flow in the communication demand graph to a vertex in the affinity graph. An edge is added between each pair of the vertices in the affinity graph to form a complete graph. A weight is attached to each edge $e' = (v'_i, v'_j)$ and is calculated as $w(e') = cost(\{e_i, e_j\}) + \sum_{e_k \in E, e_k \neq e_i, e_j} cost(\{e_k\})$, where e_i is the flow in the communication demand graph associated with v'_i in the affinity graph. $cost(\{e_k\})$ is calculated by calling on the evaluation function to evaluate the cost of implementing $\{e_k\}$ separately and $cost(\{e_i, e_j\})$ is calculated by evaluating the cost of implementing a generated RST topology for $\{e_i, e_j\}$ together. The weights of the edges in the affinity graph reflect the benefits of implementing flows represented by vertices in the affinity graph together using shared resources. by only negative weighted edges so that the total implementation cost is minimized. The smaller the weight, the less the resulting total cost of clustering the two flows connected by that edge. The motivation is to only cluster flows that are connected by small weighted edges so that the total implementation cost is minimized. Then the minimum spanning tree M of A that contains the minimum number of minimal weighted edges connecting all the vertices in A is derived. The affinity graph and its MST of the example in Figure 1 are shown in Figure 2. The cost considered is the total power consumption based on the 70nm technology power estimations shown in Table I.

Recall that the vertices in the spanning tree M corresponds



(a) Affinity graph. (b) MST of affinity graph.

Fig. 2. Affinity graph and MST of AG for the NOC synthesis example

to *flows* in the communication demand graph G. Since M is initially a spanning tree, it interconnects all vertices, which is interpreted as having all *flows* in a single cluster. During the course of the DECOMPOSE algorithm, we will selectively remove edges from M to create *disjoint set of vertices*, which will correspond to *disjoint sets of flows into groups*, thus forming a particular set partition.

In each iteration shown in lines 5-9, the algorithm systematically generates new candidate set partitions starting from the set partition chosen from the previous iteration. Inside the while loop, new set partitions are generated by temporarily removing one edge from M. This is achieved by calling the routine SelectEdgetoDelete(M). With an edge removed, the corresponding group is split into two sub-groups. We evaluate the cost of this splitting by solving an RST problem for each subgroup and calling on the evaluation function to compute the new costs. In the first iteration of the algorithm, the spanning tree M has $(n-1)$ edges. Thus, $(n-1)$ new candidate set partitions will be generated. The set partition with the best cost will be chosen as the set partition for the current iteration. This set partition, and the corresponding modified M, will be used as the starting point for the next iteration. At iteration t, M will have $(n-t-1)$ remaining edges. Therefore, $(n-t-1)$ candidate set partitions will be generated and considered. The algorithm ends when all flows in the problem have been split into their own individual groups. Then, at the end of the algorithm, at line 10, we choose the set partition with the minimum cost among the set partitions chosen from all iterations. Since at each iteration t, there can be at most $(n-t-1)$ candidate set partitions, the number of set partitions considered in the DECOMPOSE algorithm is $\Theta(n^2)$, which is again considerably smaller than $\Theta(n \log n)^n$.

VI. Router Merging

After the physical network topology has been generated for each set partition of flows, a router merging step is used to further optimize the topology and reduce the cost. Since for each set partition, routers are allocated at Steiner points or terminal points of the RST generated, routers that connect with each other can be merged to eliminate router ports and thus possibly the corresponding costs. Routers that connect to the same ports can also be merged to reduce ports and costs. We propose a greedy router merging algorithm, which works iteratively by considering all possible mergings of two routers connected with each other in each iteration. For each candidate merging, the cost difference of the resulting topology after merging and the one before merging is calculated. Then they are sorted in the increasing order of the cost difference. In the merging step, for each candidate merging from the sorted list, routers are merged if they have not merged yet and the cost is

Algorithm 2 DECOMPOSE $(G(V, E, \pi, \lambda), C, T)$

Input: $G(V, E, \pi, \lambda)$: communication demand graph
C: specified evaluation function for implementation cost
Output: T: synthesized network architecture
1: A = GenerateAffinityGraph(G)
2: M = GenerateMinSpanningTree(A)
3: $t = 0$
4: $n = |E|$
5: **while** $|M| < n$ **do**
6: $(e, soln[t], cost[t])$ = SelectEdgeToDelete(M)
7: remove e from M
8: $t = t + 1$
9: **end while**
10: $T = soln[\arg\min_t cost[t]]$
11: **return** T

SelectEdgeToDelete(M)
1: **for all** $e_i \in M$ **do**
2: temporarily remove e_i from M
3: $components(M)$ = CalculateConnectedComponents(M)
4: **for all** $g_i \in components(M)$ **do**
5: $T(g_i)$ = SolveRST(g_i)
6: $soln[e_i] = soln[e_i] \cup T(g_i)$
7: $cost[e_i] = cost[e_i]$ + EvaluateCost($T(g_i), G, C$)
8: **end for**
9: add e_i back to M
10: **end for**
11: $e = \arg\min_{e_i \in M} cost[e_i]$
12: **return** $(e, soln[e], cost[e])$

improving. After all routers are considered in the current iteration, they are updated by replacing the routers merged with the new one generated. Those routers are reconsidered in the next iteration. The algorithm keeps merging routers until no improvement can be made further.

VII. Results

A. Experimental Setup

We have implemented our two proposed algorithms CLUSTER and DECOMPOSE in C++, using a fast public domain Rectilinear Steiner Tree solver called GeoSteiner4.0 [19, 20] to generate the physical network topologies in the inner loop. The proposed router merging algorithm has been integrated into the two algorithms as well.

Three sets of benchmarks were used to evaluate these algorithms. The first set of benchmarks are four different video processing applications obtained from [9], including a Video Object Plane Decoder (VOPD), an MPEG4 decoder, a Picture-In-Picture (PIP) application, and a Multi-Window Display (MWD) application. The next set of benchmarks were obtained from [3] and [7]. They correspond to different encoder/decoder combinations of a H.263 video codec, a MP3 audio codec, and a generic MultiMedia System (MMS). Finally, to generate larger benchmark instances, we generated synthetic benchmarks from the above video applications.

All experimental results were obtained on a 1.5 GHz Intel P4 processor machine with 512 MB of memory running Linux.

B. Method of Evaluation

In our experiments, we aim to evaluate the performance of the two proposed algorithms CLUSTER and DECOMPOSE on all benchmarks with the objective of minimizing the total power consumption of the synthesized NoC architectures. The total power consumption includes the dynamic switching power which is a function of data rate passing through each component and the leakage power which is related to all the components in the NoC architecture. To estimate power consumption different router configurations considered during the synthesis process, we used a state-of-the-art power simulator called Orion [11, 12] that considers both leakage and dynamic power. We set the operational frequency to be 1 GHz, the buffer size to be 4 flits, and the size of each flit to be 128 bits. The leakage power and switching bit energy of routers with different example port configurations in 70nm technology are showed in Table I. For the link power parameters, we used RC wires with repeated buffers and a minimum global wire pitch. The static power and switching bit energy parameters in 70nm technology were obtained using the models from [14] and are listed in Table I.

TABLE I
Power consumption of NoC components [11, 14]

(a) Power consumption of routers

Ports (in x out)	2x2	3x2	3x3	4x3	4x4	5x4	5x5
Leakage power (W)	0.0069	0.0099	0.0133	0.0172	0.0216	0.0260	0.0319
Switching bit energy (pJ/bit)	0.3225	0.0676	0.5663	0.1080	0.8651	0.9180	1.2189

(b) Power consumption of links

Wire length (mm)	1	4	8	12	16
Leakage power (W)	0.000496	0.001984	0.003968	0.005952	0.007936
Switching bit energy (pJ/bit)	0.6	2.4	4.8	7.2	9.6

For evaluation, fair direct comparison with previously published NoC synthesis results is difficult in part because of vast differences in the power parameters assumed[1]. Therefore, to evaluate the performance of our proposed algorithms, we have designed two sets of experiments. In the first set of experiments, we generated mesh topologies for the benchmarks by modifying the design procedure to synthesize NoCs based on mesh structure. To obtain mesh topologies, we generated a design with each core connecting to a single router and restricted the router sizes to have 5 input/output ports. We also generated a variant of the basic mesh topology, optimized mesh (opt-mesh), by removing those ports and links that are not used by the traffic flows. These experiments are designed to show the benefits of application-specific NoC architectures. In the second set of experiments, we implemented an exact algorithm, referred to as EXACT, that exhaustively enumerates all distinct set partitions. These experiments are designed to show how close our heuristic algorithms are to exact results.

C. Comparison of Results

The synthesis results of CLUSTER and DECOMPOSE on all benchmarks at 70nm with comparison to mesh and opt-mesh topologies are shown in Table II. The power results show that both algorithms can achieve substantial reduction in power consumption over the standard mesh and opt-mesh topologies in all cases. In particular, a 6.92× and a 6.77× reduction on average in power consumption over standard mesh topologies can be achieved by CLUSTER and DECOMPOSE respectively, with a 2.68× and a 2.60× reduction over the optimized mesh topologies for each of them. The execution times reported

[1]We use the Orion simulator to evaluate the NoC power consumption [11, 12]. The power estimates from Orion are consistent with another published power-optimized NoC implementation described in [13]. The power estimates are on the same order of magnitude for the same router configuration in the same technology.

TABLE II
NOC SYNTHESIS RESULTS COMPARISON WITH MESH TOPOLOGIES

			CLUSTER				DECOMPOSE				mesh	opt-mesh
Appli.	|V|	|E|	Power (W)	Time (sec)	Impro. CL/ mesh	Impro. CL/ opt	Power (W)	Time (sec)	Impro. DE/ mesh	Impro. DE/ opt	Power (W)	Power (W)
VOPD	12	14	0.042	1.35	6.42	3.60	0.042	0.34	6.42	3.60	0.272	0.152
MPEG4	12	13	0.039	1.09	7.07	2.35	0.040	0.30	6.84	2.27	0.276	0.092
PIP	8	8	0.021	0.24	8.65	2.93	0.021	0.09	8.65	2.93	0.178	0.060
MWD	12	13	0.028	1.57	9.24	4.31	0.031	0.32	8.44	3.94	0.260	0.121
H263	7	10	0.037	0.37	4.31	2.24	0.036	0.14	4.18	2.17	0.155	0.080
G5	12	12	0.028	0.82	9.11	3.76	0.031	0.23	8.29	3.42	0.258	0.106
mp3enc	7	8	0.023	0.09	4.28	2.16	0.023	0.05	4.28	2.16	0.100	0.050
mp3dec	6	6	0.016	0.11	6.02	2.60	0.016	0.04	6.02	2.60	0.099	0.043
h263dec	7	8	0.026	0.19	6.77	2.47	0.026	0.07	6.77	2.47	0.178	0.065
MMEnc	14	19	0.068	3.89	5.50	1.91	0.073	0.77	5.12	1.78	0.376	0.131
MMDec	11	14	0.054	1.88	4.74	2.24	0.054	0.32	4.74	2.24	0.257	0.122
MMS	25	33	0.123	32.07	5.21	2.11	0.132	3.96	4.85	1.96	0.642	0.260
V+M	24	27	0.086	15.47	7.39	2.35	0.090	1.30	7.04	2.24	0.633	0.202
M+P	20	21	0.053	7.45	9.29	2.89	0.053	0.96	9.29	2.89	0.495	0.154
V+M+M	36	40	0.120	73.02	8.37	2.49	0.124	7.54	8.13	2.42	1.008	0.300
4in1	44	48	0.139	130.50	10.25	2.47	0.142	12.68	10.02	2.42	1.425	0.344

TABLE III
NOC SYNTHESIS RESULTS COMPARISON WITH EXACT SOLUTIONS

	CLUSTER			DECOMPOSE			EXACT	
Appli.	Power (W)	Time (sec)	Improv. CL/EX	Power (W)	Time (sec)	Improv. DE/EX	Power (W)	Time (sec)
VOPD	0.042	1.35	n.a.	0.042	0.34	n.a.	t.o.	t.o.
MPEG4	0.039	1.09	n.a.	0.040	0.30	n.a.	t.o.	t.o.
MWD	0.028	1.57	n.a.	0.031	0.32	n.a.	t.o.	t.o.
MMEnc	0.068	3.89	n.a.	0.073	0.77	n.a.	t.o.	t.o.
MMDec	0.054	1.88	n.a.	0.054	0.32	n.a.	t.o.	t.o.
MMS	0.123	32.07	n.a.	0.132	3.96	n.a.	t.o.	t.o.
V+M	0.086	15.47	n.a.	0.090	1.30	n.a.	t.o.	t.o.
M+P	0.053	7.45	n.a.	0.053	0.96	n.a.	t.o.	t.o.
V+M+M	0.120	73.02	n.a.	0.124	7.54	n.a.	t.o.	t.o.
4in1	0.139	130.50	n.a.	0.142	12.68	n.a.	t.o.	t.o.
mp3enc	0.023	0.09	**1.00**	0.023	0.05	**1.00**	0.023	1.00
mp3dec	0.016	0.11	**1.00**	0.016	0.04	**1.00**	0.016	1.00
h263dec	0.026	0.19	**1.00**	0.026	0.07	**1.00**	0.026	7.00
PIP	0.021	0.24	**1.00**	0.021	0.09	**1.00**	0.021	9.00
H263	0.037	0.37	**1.03**	0.036	0.14	**1.00**	0.036	293.00
G5	0.028	0.82	**1.00**	0.031	0.23	**1.10**	0.028	14440.00

show that both algorithms work very fast, in just seconds. As can be seen, CLUSTER can achieve better results than DECOMPOSE because it examines more set partition candidates in its solution space, but it requires longer run times.

In the next set of experiments, we compared our heuristic algorithms with an exact algorithm that enumerates all distinct set partitions. As the number of distinct set partitions grows $\Theta(n \log n)^n$, the CPU times for generating the exact solutions increase very quickly. We set a CPU timeout period of 5 hours. The results are compared in Table III. Out of the 16 benchmarks tested, we were able to determine the exact solutions for 6 of the benchmarks. Of these 6 benchmarks, both algorithms were able to achieve the exact solution in 5 out of the 6 cases, and on average, the results are within just 1% and 2% of the exact results for CLUSTER and DECOMPOSE respectively. Moreover, the CPU times for the 6 benchmarks are all under 1 second whereas the EXACT algorithm took as much as 4.5 hours to achieve similar results.

VIII. CONCLUSIONS

In this paper, we proposed a formulation of the application-specific NoC synthesis problem based on the decomposition of the problem into the inter-related steps of finding a good set partition of communication flows, deriving a good physical network topology for each group in the partition, and calculating the cost of the set partitions and topologies by means of an evaluation function. We proposed two heuristic algorithms called CLUSTER and DECOMPOSE for systematically examining different possible set partitions, and we proposed the use of Rectilinear Steiner Tree algorithms for deriving good physical network topologies. Although we use Steiner trees to generate a physical network topology for each group in the set partition, the final NoC architecture synthesized is not necessarily limited to just trees as Steiner tree implementations of different groups may be connected to each other to form non-tree structures. Experimental results on a variety of benchmarks using a power consumption cost model show that our algorithms can produce effective solutions with fast execution times comparing to both mesh implementation and EXACT solutions.

REFERENCES

[1] W. J. Dally and B. Towles, "Route packet, not wires: On-chip interconnection networks," *DAC,* 2001.

[2] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, pp. 70-78, January 2002.

[3] J. Hu, R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," *ASP-DAC*, 2003.

[4] S. Murali and G. De Micheli, "Bandwidth constrained mapping of cores onto NoC architectures," *DATE,* 2004.

[5] R. Ravi, M. V. Marathe, et al., "Approximation algorithms for degree-constrained minimum-cost network-design problems," *Algorithmica*, 31(1):58-78, 2001.

[6] A. Pinto, L. P. Carloni, A. L. Sangiovanni-Vincentelli, "Efficient synthesis of networks on chip," *ICCD,* 2003.

[7] K. Srinivasan, K. S. Chatha, G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on VLSI Systems*, Volume 14, Issue 4 (April 2006), pp. 407-420.

[8] S. Murali, et al., "Designing application-specific networks on chips with floorplan information," *ICCAD,* 2006.

[9] D. Bertozzi, et al., "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113-129, Feb. 2005.

[10] A. S. Grove, "Changing vectors of Moore's law," Keynote presentation, *International Electron Device Meeting*, December 2002.

[11] H. Wang, X. Zhu, et al., "Orion: A Power-Performance Simulator for Interconnection Networks", *MICRO 35*, Istanbul, Turkey, November 2002.

[12] X. Chen, L.-S. Peh, "Leakage power modeling and optimization in interconnection networks," *ISPLED*, 2003.

[13] R. Mullins, "Minimising dynamic power consumption in on-chip networks," *Intl. Symp. on System-on-Chip*, Tampere, Finland, Nov 2006.

[14] L. Zhang, H. Chen, et al., "Repeated On-Chip Interconnect Analysis and Evaluation of Delay, Power, and Bandwidth Metrics under Different Design Goals", *ISQED 2007*

[15] M. B. Taylor et al., "The RAW microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 6, pp. 25-35, Mar./Apr. 2002.

[16] K. Sankaralingam et al. "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," *ISCA*, 2003.

[17] D. E. Knuth, The art of computer programming. Pre-Fascicle 3B. A draft of Sections 7.2.1.4-5: Generating all partitions. Addison-Wesley.

[18] D.-Z. Du, J. M. Smith, J. H Rubinstein, Advances in Steiner Trees. *Dordrecht, Netherlands: Kluwer*, 2000.

[19] D. M. Warme, P. Winter, M. Zachariasen, "Exact algorithms for plane Steiner Tree problems: A computational study," Advances in Steiner Trees, pp. 81-116, Kluwer Academic Publishers, 2000.

[20] `http://www.diku.dk/geosteiner/`

[21] N. A. Sherwani, Algorithms for VLSI Physcial Design Automation, 3rd edition, Kluwer Academic Publishers, Norwell, MA, 1998.

[22] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," *DAC*, 1998.

[23] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. on VLSI Systems*, vol 11(6), pp. 1120-1135, December 2003.

[24] Z. Feng, B. Yao, and C.K. Cheng, Floorplan Representation in VLSI, Handbook of DATA Structures and Applications, by D.P. Mehta and S. Sahni, Chapman and Hall, pp. 53-1: 53-29, 2004.