2013

An Interconnection Network Topology Generation Scheme for Multicore Systems

Bharath Phanibhushana
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/theses

Part of the VLSI and Circuits, Embedded and Hardware Systems Commons

Retrieved from https://scholarworks.umass.edu/theses/1146

This thesis is brought to you for free and open access by ScholarWorks@UMass Amherst. It has been accepted for inclusion in Masters Theses 1911 - February 2014 by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.
AN INTERCONNECTION NETWORK TOPOLOGY
GENERATION SCHEME FOR MULTICORE SYSTEMS

A Thesis Presented
by
BHARATH PHANIBHUSHANA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2013

Electrical and Computer Engineering
AN INTERCONNECTION NETWORK TOPOLOGY
GENERATION SCHEME FOR MULTICORE SYSTEMS

A Thesis Presented
by
BHARATH PHANIBHUSHANA

Approved as to style and content by:

________________________________________
Sandip Kundu, Chair

________________________________________
C Mani Krishna, Member

________________________________________
Wayne Burleson, Member

________________________________________
C.V. Hollot, Department Chair
Electrical and Computer Engineering
ACKNOWLEDGMENTS

I would like to take this opportunity to express my deepest gratitude to my advisor, Prof. Sandip Kundu for his support and guidance throughout this project. I also wish to extend my gratitude to my committee members, Prof. Wayne Burleson and Prof. Mani Krishna for their valuable criticism and suggestions. I would like to thank all my friends who supported me during the entire course. Finally I would like to thank God and my parents for their constant support which helped me in being strong, focussed and motivated all through the journey.
ABSTRACT

AN INTERCONNECTION NETWORK TOPOLOGY GENERATION SCHEME FOR MULTICORE SYSTEMS

SEPTEMBER 2013

BHARATH PHANIBHUSHANA
B.E, VISHVESHWARIAH TECHNOLOGICAL UNIVERSITY
M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sandip Kundu

Multi-Processor System on Chip (MPSoC) consisting of multiple processing cores connected via a Network on Chip (NoC) has gained prominence over the last decade. Most common way of mapping applications to MPSoCs is by dividing the application into small tasks and representing them in the form of a task graph where the edges connecting the tasks represent the inter task communication. Task scheduling involves mapping task to processor cores so as to meet a specified deadline for the application/task graph. With increase in system complexity and application parallelism, task communication times are tending towards task execution times. Hence the NoC which forms the communication backbone for the cores plays a critical role in meeting the deadlines. In this thesis we present an approach to synthesize a minimal network connecting a set of cores in a MPSoC in the presence of deadlines. Given a task graph and a corresponding task to processor schedule, we have developed a partitioning methodology to generate an efficient interconnection network for the cores.
We adopt a 2-phase design flow where we synthesize the network in first phase and in second phase we perform statistical analysis of the network thus generated. We compare our model with a simulated annealing based scheme, a static graph based greedy scheme and the standard mesh topology. The proposed solution offers significant area and performance benefits over the alternate solutions compared in this work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1. PROBLEM FORMULATION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 On-chip router architecture</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Summary of previous works</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Analysis and motivation</td>
<td>7</td>
</tr>
<tr>
<td>1.4.1 Motivational example</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Problem statement</td>
<td>9</td>
</tr>
<tr>
<td>2. DESIGN FLOW AND ALGORITHMS</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Network topology synthesis</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Phase I: Core to router mapping</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Phase II: Path generation</td>
<td>14</td>
</tr>
<tr>
<td>2.2.3 Phase III: Routing</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Example illustrating our algorithms</td>
<td>15</td>
</tr>
<tr>
<td>3. SCHEMES FOR COMPARISON</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Simulated annealing approach</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Static graph based greedy approach</td>
<td>22</td>
</tr>
</tbody>
</table>
4. EXPERIMENTAL RESULTS ........................................ 25

4.1 Experimental setup ............................................. 25
4.2 Benchmark profiling .............................................. 28
4.3 Effect of coefficient of cost function on schedule length ......... 28
4.4 Effect of deadlines on coefficient of cost function ................. 32
4.5 Effect of router count on schedule length and total area ........... 32
4.6 Effect of variation in task execution times .......................... 39
4.7 Comparison with various schemes .................................. 41
  4.7.1 Performance comparison ....................................... 41
  4.7.2 Total router area comparison ................................... 42
  4.7.3 Quality metrics .................................................. 43

5. CONCLUSION ....................................................... 46

BIBLIOGRAPHY .......................................................... 47
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Notations used in Kernighan-Lin module</td>
</tr>
<tr>
<td>3.1</td>
<td>Notations used in Simulated Annealing</td>
</tr>
<tr>
<td>4.1</td>
<td>Benchmarks showing number of cores, nodes and edges</td>
</tr>
<tr>
<td>4.2</td>
<td>Router specifications used in the simulations</td>
</tr>
<tr>
<td>4.3</td>
<td>Processor configuration for benchmark profiling</td>
</tr>
<tr>
<td>4.4</td>
<td>Table showing total router area in $\mu m^2$ for all the schemes and % benefit over all the schemes</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An example task graph showing different tasks and dependencies.</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Virtual Channel router[35]</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Mesh topology for a four core system</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>Topology obtained from our scheme</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Top level design flow</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Our top level final algorithm</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Set of cores and routers</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Kernighan-Lin algorithm maps cores to routers</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>Existing partitions become nodes</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>Recursive Kernighan-Lin iteration 1</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Recursive Kernighan-Lin iteration 2</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Final network generated</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Top level simulated annealing module</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Static graph showing average bandwidth information</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Experimental set up</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Benchmark profiling to obtain communication to computation ratio for various multimedia benchmarks</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Schedule length plotted against k - coefficient of cost function for Benchmark 1</td>
<td>29</td>
</tr>
</tbody>
</table>
4.4 Schedule length plotted against k - coefficient of cost function for Benchmark 2 .................................................. 29

4.5 Schedule length plotted against k - coefficient of cost function for Benchmark 3 .................................................. 30

4.6 Schedule length plotted against k - coefficient of cost function for Benchmark 4 .................................................. 30

4.7 Schedule length plotted against k - coefficient of cost function for Benchmark 5 .................................................. 31

4.8 Schedule length plotted against k - coefficient of cost function for Benchmark 6 .................................................. 31

4.9 k - coefficient of cost function plotted against deadline for Benchmark 1 ............................................................ 32

4.10 k - coefficient of cost function plotted against deadline for Benchmark 2 ............................................................ 33

4.11 k - coefficient of cost function plotted against deadline for Benchmark 3 ............................................................ 33

4.12 k - coefficient of cost function plotted against deadline for Benchmark 4 ............................................................ 34

4.13 k - coefficient of cost function plotted against deadline for Benchmark 5 ............................................................ 34

4.14 k - coefficient of cost function plotted against deadline for Benchmark 6 ............................................................ 35

4.15 Schedule length plotted against various router scenarios for Benchmark 1 ............................................................ 36

4.16 Schedule length plotted against various router scenarios for Benchmark 2 ............................................................ 36

4.17 Schedule length plotted against various router scenarios for Benchmark 3 ............................................................ 37

4.18 Schedule length plotted against various router scenarios for Benchmark 4 ............................................................ 37
4.19 Schedule length plotted against various router scenarios for Benchmark 5 ................................................................. 38

4.20 Schedule length plotted against various router scenarios for Benchmark 6 ................................................................. 38

4.21 Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM1 .............. 39

4.22 Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM2 .............. 40

4.23 Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM3 .............. 40

4.24 Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM4 .............. 41

4.25 Plot showing performance numbers of various schemes for all the benchmarks ............................................................... 42

4.26 Plot showing average port count of various schemes for all the benchmarks ............................................................... 44

4.27 Plot showing average hop count of various schemes for all the benchmarks ............................................................... 44

4.28 Plot showing average hop count of various schemes for all the benchmarks ............................................................... 45
CHAPTER 1
PROBLEM FORMULATION

1.1 Introduction

System-on-Chip (SoC) performance requirements are becoming increasingly complex due to the emerging and evolving applications. Various state-of-art applications like MPEG, video streaming, 3GPP, WiMAX which require large amount of data processing have favored MultiProcessor SoC (MPSoC) implementation. Examples of commercial SoC designs include Intel IXP series (IXP1200, IXP2400, IXP2800) [1], TI TMS320C8x [2], Motorola C-port C-5 [3]. Multicore architecture has proven to be very efficient especially for applications with high parallelism [4]. As the number of cores multiply, several researchers are working towards incorporating thousands of cores on a single chip [6]. The traditional bus architecture which formed the communication back-bone is not scalable as the number of on-chip processing elements is increasing rapidly. Network-on-Chip (NoC) has been proposed as a solution to overcome the scalability issue [7].

Usually in MPSoC implementation, an application is divided into smaller sections of code that execute on a core, known as tasks. In the task graph, a node represents a task and an edge represents the communication between the pair of tasks. The weight of the node represents the execution time of the task while that of an edge represents the time needed to transfer the data between a pair of tasks. Scheduling of tasks onto cores/processor is a key phase in multiprocessor system design, as it directly affects the performance. Multiprocessor task scheduling problem can be defined as: a set of \( m \) tasks, with task \( i \) having execution time of \( e_i \), needs to be scheduled on set of \( p \)
processors such that the total execution time meets the deadline. The task graph has a control dependency for the tasks wherein a task can execute only after all its parent tasks have finished execution. A task graph has a deadline associated with it, before which all the tasks must finish the execution. Slack of a task is defined as the amount of time by which the task can be delayed without missing the deadline. Figure 1.1 shows an example task graph, where $Ti$ represent the tasks connected via communication edges. A four core/processor is assumed. T/C represents a task mapped to a corresponding core. A node represents the task and weight of the node represents the task execution time. The weight of the edge represents the time needed to transfer data between the pair of communicating cores. Values in circles represent the forward and backward task execution calculations which is equivalent to the earliest arrival time and the latest arrival time, values in brackets represent the slack available and the final deadline for the task graph is also shown. Unit for all the values are in cycles. All the scheduled tasks rely on a shared Network on Chip (NoC) connecting the cores for data transfer. Thus the total schedule length not only depends on the processor performance but also on the available NoC resources. With large number of processing cores and hard deadlines, on-chip communication network becomes the bottleneck. If a network does not satisfy the communication requirements, some tasks may have to wait for the shared communication resource to be available, which may lead to increase in schedule length. During runtime the task execution times may change resulting in traffic pattern variations. The designed network should be capable of handling these variations, hence there is a need for a methodology which considers all the communication constraints during network synthesis and models the runtime variations so as to analyse its effect on deadlines. The final goal of such a methodology is to generate a runtime variation tolerant network.
1.2 On-chip router architecture

The network performance is characterized by the interconnection topology and its router architecture. The two major components of a router is the routing strategy and the flow control technique [30]. Wormhole routers and virtual channel routers are the two popular and largely discussed on-chip router designs. In wormhole router there are 3 stages: routing, switch arbitration and crossbar traversal. Switch arbiter receives all the requests from the input ports, checks the status of the output ports, resolves conflicts and grants free ports to the requestors. Then the packet is passed through the crossbar and the for the entire duration of packet transfer the crossbar passage is held, until the tail flit exits the router. On the other hand in a virtual channel router, multiple virtual channels are multiplexed onto a physical channel and instead of reserving passage through the crossbar for the entire duration of packet, the

Figure 1.1. An example task graph showing different tasks and dependencies.
flits of different packets are transferred cycle-by-cycle basis onto the crossbar. Adding virtual channels has proven to deliver increased throughput and better performance [31]. We use a virtual channel router and is shown in Figure 1.2. It consists of 4 pipelined stages [35]:

**Routing Logic:** When the first flit of the packet arrives at the router, a VC buffer is allocated using its VC identifier by the input controller. The routing logic examines the routing field and a set of output virtual channels on which the packet can be routed is produced. The number of output VCs depends on the routing function used by the routing logic.

**VC allocator:** After obtaining the candidate set of output VCs, the input controller now submits request to the global VC allocator for corresponding output VCs. The VC allocator takes requests from all the input controllers and allocates available VCs to the requestors. It updates the status of these channels as unavailable.

![Figure 1.2. Virtual Channel router][35]
Switch allocator: After obtaining the output VCs, the input controller submits request to the switch allocator to obtain the physical channel required. The switch allocator takes the requests from all the input controllers and match the VCs request to the output physical channel. If the requested channel is available, the requested channel is granted and the corresponding crossbar signals are activated.

Crossbar: The last stage is the crossbar traversal. After the arbitration and the allocation phases are completed, the flit has secured the output physical channel successfully and it passes the crossbar for the next hop.

1.3 Summary of previous works

A large body of research exists which work towards designing Network on Chip (NoC) in MPSoCs and several techniques have been proposed in the past [5, 10, 11, 12, 13]. Below we present a summary of the relevant works.

Murali et al. presented a tool for automatic topology generation[11]. A library consisting of several standard topologies was built. A three phase design flow was presented. In the first phase for a given routing function and design objectives which include the area and bandwidth are mapped onto different existing standard topologies and the mappings are obtained. In the second phase the best topology is chosen amongst the existing standard topologies based on the design constraints. In the final phase the SystemC components of the network components are generated.

Srinivasan et al. presented a mixed linear programming based technique to generate custom NoCs in [15]. The inputs to the methodology were a directed communication trace graph representing average bandwidth between the processing elements, a router architecture with a given number of input and output ports, maximum bandwidth that a router can support and largest number of routers available. The objective function was to minimize the total power consumption. The network latency was considered in terms of router hops. Few heuristics namely clustering based
heuristic, path based deterministic heuristic and path based randomization heuristic were also presented and a comparison was done between the various approaches.

A promising solution was presented by Murali et.al in [9]. The design flow uses min cut based partitioning to partition the average bandwidth graph and clusters are generated. Heuristics were presented for generating the paths between the routers and minimizing the power was the objective. They also considered the floorplanning constraints and thus a flow incorporating complete solution starting from topology exploration to layout generation was presented. They compared the results with standard mesh and an optimized mesh topologies.

Chan et al. presented a hybrid topology which had both point to point links and packet switched networks [14]. Yu et al. used a min-cost max-flow algorithm to insert switches and network interfaces [16]. Zhong et al. proposed an algorithm for cluster generation and a heuristic for switch and network insertion [17]. Also there have been some work on trying to map the communications on the NoC topology and optimize the communication latency. Hu et. al. proposed an algorithm to schedule tasks and communications to an standard existing topology, but did not consider the problem of network synthesis [19]. Lahiri et al. proposed algorithms to map hardware elements on predefined communication architectures according to a communication profile defined through performance analysis [21]. This analysis enables to discover potential contentions on shared channels. However, the approach is assuming a memory-less communication infrastructure, which prevents using this method to most of the router-based NoCs. Abstract NoC models were defined in [23] which could be integrated into MPSoC design. The standard mesh and torus NoC was considered and NoC synthesis issues were not considered.


1.4 Analysis and motivation

We made the following observations:

• Most of the work in literature mentioned above consider just an average bandwidth graph and a sustained rate of traffic flow. Hence all the communication constraints are not visible to their approaches.

• Very few works consider the communication latency between the cores as an optimization constraint. The runtime variations may affect the traffic patterns assumed during the design phase. None of the work does an analysis on the runtime variations which may affect the final deadlines.

• Moreover, as task communication times become comparable to execution times, the NoC design plays a bigger role in determining the total schedule length. Most of the current work considers a single cycle/fixed cycle router latency and hence their results are inaccurate.

• Conventional NoC architectures like Torus and Mesh are designed for uniform architectures having regular floorplan. Most MPSoCs have non-uniform core sizes making it difficult to map them onto regular standard topologies.

1.4.1 Motivational example

Consider the task graph shown in Figure 1.1. Assume a four core system is used and the following task to processor assignment was done:

• C1 - T1,T5,T7

• C2 - T2,T6

• C3 - T3

• C4 - T4
Figure 1.3. Mesh topology for a four core system

Figure 1.4. Topology obtained from our scheme

Figure 1.3 shows a mesh topology for the interconnection network. A careful analysis of the communication traces show that C1 never communicates with C4; C4 only communicates with C2, C1 to C3 communication is frequent and critical and hence C1 and C3 need to be placed together. An optimal solution does not require four routers and the topology generated from our scheme is shown in figure 1.4. The number of routers and links can be reduced if the communication constraints are thoroughly analysed resulting in significant area benefits. Hence in this thesis we propose a structured and systematic approach for on-chip network generation which aims at solving the above mentioned problems. The main contributions of this thesis are:

- Unlike prior work we consider a task model for application representation; the most important advantage of doing so is that we get a good visibility into all
communication constraints, which are not available in existing solutions which start from average bandwidth graph.

- The final deadline for the task graph is affected by both task execution times and task communication times. During runtime, the task execution times may vary, affecting the final deadlines. To account for this we follow a two phase design flow: in the first phase we design the network assuming mean execution times and in the second phase we model the runtime characteristics of the tasks by doing statistical simulations wherein a distribution for task run times is assumed. This methodology is largely motivated by the process of logic synthesis of gates under design constraints followed by a statistical static timing analysis for timing closure. Thus accounting for runtime variations of task execution times and providing a checking mechanism for the designer.

- We adopt a partitioning approach for network synthesis. We use Kernighan-Lin (KL) algorithm for partitioning the cores and mapping them onto the routers and we have developed a recursive KL algorithm for generating the links between the routers.

- For comparison we have developed a simulated annealing based topology synthesis scheme and a static graph based greedy scheme. Also comparison with standard mesh topology has been done and results have been presented.

- We have defined few metrics to analyse the quality of network generated and comparison with all the schemes have been presented.

1.5 Problem statement

Our optimization problem can be formulated as follows:

Given:
• An application task graph

• A deadline for the task graph completion

• Task to processor schedule

**Objective:**

• Obtain an interconnection network for the processing cores which require minimal network resources. Minimize the total on-chip router area:

\[
\sum_{k=1}^{n} A_k
\]

where \( n \) = total no. of routers, \( A_k \) = area of router \( k \)

**Constraint:**

• Meet the deadline

\[
T_i + \sum_{j=1}^{m} D_j \leq T
\]  \hspace{1cm} (1.1)

where

\( T \) = Deadline for the task graph

\( T_i \) = Total task execution time for path \( i \)

\( m \) = Number of routers in path \( i \)

\( D_j \) = Delay of router \( j \)
CHAPTER 2
DESIGN FLOW AND ALGORITHMS

2.1 Introduction

Application specific custom network generation methodologies have proven to perform better than standard topologies [9]. In our network topology generation scheme we adopt a 2-phase design flow where we synthesize the network in first phase and in second phase we do a statistical analysis to verify the quality of network generated. The top level flow for our NoC design methodology is shown in Figure 2.1. Our network synthesis has three phases:

- Phase I: Core to router mapping: We use a partitioning approach to map cores to routers. We use Kernighan-Lin (KL) algorithm, the very powerful and popular two way partitioning algorithm, to partition the cores and assign them to the routers.

- Phase II: Path generation: The second phase of our algorithm is adding links between the routers and generating paths for each pair of communicating cores. We propose a recursive KL algorithm to obtain links between the routers.

- Phase III: Routing: The final phase of our algorithm is Routing and we use the Dijkstra’s algorithm.
2.2 Network topology synthesis

2.2.1 Phase I: Core to router mapping

It is of utmost importance to have a core mapped to the correct router. We adopt a partitioning approach to map the cores to routers. Partitioning approach for generating on-chip networks has been shown to be efficient especially when there are large number of cores [9, 26]. Depending on the number of routers the cores are partitioned and assigned to a particular router. We use Kernighan-Lin(KL) algorithm for partitioning.

Kernighan-Lin Algorithm:

Kernighan-Lin [20] is the most popular two way partitioning algorithm. It is an iterative improvement algorithm which attempts to find optimal series of interchange operations between partitions so that the gain is maximized. KL algorithm has important applications in various fields including VLSI circuit partitioning, placement, load balancing [27, 28, 29].

The algorithm is characterized by a cost matrix and is shown in 2.1. This cost function has two parameters, Bandwidth and Slack between the cores. The idea
Notation | Explanation
--- | ---
$A, B$ | Starting random partitions
$\zeta$ | A 2-tuple consisting of cores and routers ($C, R$)
$C_{ij}$ | Cost between core $i$ and core $j$
$BW_{ij}$ | Bandwidth between core $i$ and core $j$
$S_{ij}$ | Slack between core $i$ and core $j$
$k$ | Coefficient of cost function
$E_a$ | External cost of node $a$
$I_a$ | Internal cost of node $a$
$D_a$ | $E_a - I_a$ Benefit of moving $a$ from $A$ to $B$
$g_{ab}$ | $D_a + D_b - 2C_{ab}$ Gain of swapping $a$ and $b$
$P_m, P_n$ | Intermediate partitions
$\sigma$ | Standard deviation
$\Gamma$ | A set representing all the existing current partitions
$\xi$ | A design point representing partitions obtained from $(\zeta, C_{ij})$

Table 2.1. Notations used in Kernighan-Lin module

behind choosing this cost function is motivated by the fact that if two cores have large bandwidth of data to transfer and less slack available for communication, then the two cores need to be physically close to each other which can be achieved by placing them in a single partition. The coefficient of cost function $k$, decides the contribution of the slack in the cost function.

$$C_{ij} = BW_{ij} - k \times S_{ij}$$  \hspace{1cm} (2.1)

Consider two random partitions $A$ and $B$, node $a \in A$ and $b \in B$. The contribution of node $a$ to the cutset is called as the External cost of node $a$, and is defined as:

$$E_a = \sum_{v \in B} C_{av}$$  \hspace{1cm} (2.2)

Internal cost of node $a$ is defined as:

$$I_a = \sum_{v \in A} C_{av}$$  \hspace{1cm} (2.3)
If we move node $a$ from $A$ to $B$, it will increase the value of cutset by $I_a$ and decrease it by $E_a$. Hence the benefit of moving $a$ from $A$ to $B$:

$$D_a = E_a - I_a$$  \hspace{1cm} (2.4)

If $a$ and $b$ are interchanged, the reduction in the cost, or the gain of swapping is given by:

$$g_{ab} = D_a + D_b - 2C_{ab}$$  \hspace{1cm} (2.5)

The Kernighan-Lin procedure is explained below.

---

**Algorithm 1** Module KERNIGHAN\_LIN()

1: Input: $\zeta, C_{ij}$
2: Output: $\xi$
3: repeat
4: Start with random partitions $A$ and $B$
5: Unmark all the nodes
6: Calculate $g_{ab}$ for all $a \in A$ and $b \in B$
7: Choose an unmarked pair $(a_1,b_1)$ with maximum gain $g_1$ but don’t swap $a_1$ and $b_1$
8: Update $D$ for remaining nodes as if $a_1$ and $b_1$ were swapped and recompute $g$
9: Choose second pair $(a_2,b_2)$ with maximum gain $g_2$. Compute $G = g_1 + g_2$
10: At this point we have pairs $(a_1,b_1), (a_2,b_2) \ldots$ and gains $g_1, g_2 \ldots$
11: The gain for first $k$ swaps: $G_k = \sum_{i=1}^{k} g(i)$
12: If there is no $k$ such that $G_k > 0$ then the current partition cannot be improved; otherwise choose the $k$ that maximizes $G_k$ and make the swappings permanent
13: until $G_k \leq 0$; no further pairwise improvements are possible

---

2.2.2 Phase II: Path generation

After Phase I, which maps the cores to routers, the second phase of our algorithm is adding links between the routers and generating paths for each pair of communicating routers. The area of an on-chip router increases quadratically with the number of ports [32] and hence it is really important to carefully choose the interconnection links for the routers. An incremental path generation methodology is proposed by adopting
a recursive KL algorithm. Here existing partitions become nodes. We update the cost function between the newly formed nodes. The new cost function is the summation of the costs between each nodes in the partition. We feed this set of nodes with an associated cost function into KL partitioner. The output is a set of partitions wherein routers that need to be connected are in same partition. Thus by following this process recursively we obtain interconnection links between the routers. The steps involved in the recursive KL algorithm is shown in algorithm 2.

**Algorithm 2** Module RECURSIVE_KERNIGHAN_LIN()

Input: $\xi, C_{ij}$

repeat

Existing set of partitions $\xi$ become nodes

for all partitions do

Update the cost function between partition $P_m$ and $P_n$

$C_{mn} = \sum C_{ij} \quad \forall i \in P_m, \forall j \in P_n$

end for

$\xi = \text{KERNIGHAN_LIN}(\Gamma, C_{mn})$

until $P_m = P_n = \text{Corecount}$

---

2.2.3 Phase III: Routing

The final phase of our network synthesis is creating routing paths. We use Dijkstra’s shortest path algorithm to find the routing paths. We set the cost function as the hop count. Here the objective is to find the path with minimum hops between each pair of routers. The steps involved is shown in the algorithm 3.

2.3 Example illustrating our algorithms

Let us consider an example to illustrate our algorithms. Consider a set of cores and a set of routers shown in figure 2.3. Let us further assume that a task graph is mapped to cores and the cost function defined in Equation 2.1 is calculated between each pair of cores. The first phase of our algorithm is to map the cores to routers using KL algorithm. Figure 2.4 shows the cores mapped to the routers. The second phase
**Algorithm 3 Module DIJKSTRA()**

1. Set Cost between nodes = Hop Count
2. Set root node value=0 and all other nodes = infinity
3. Mark all nodes unvisited
4. For the current node consider all the neighbors and update their tentative distance
5. If the distance is less than previously recorded value then overwrite that distance
6. When all the neighbors are done, mark the current node as visited node and remove it from unvisited set
7. Move to the unvisited node which has smallest distance, mark that as current node and move to step 4
8. If the destination node is marked visited then the algorithm stops

---

**Figure 2.2.** Our top level final algorithm
of our algorithm is to generate links between the routers. The existing partitions become nodes. Then using the KL algorithm the links are obtained. We run this recursively until the entire partition count becomes equal to the core count.

**Figure 2.3.** Set of cores and routers

**Figure 2.4.** Kernighan-Lin algorithm maps cores to routers
Figure 2.5. Existing partitions become nodes

Figure 2.6. Recursive Kernighan-Lin iteration 1
Figure 2.7. Recursive Kernighan-Lin iteration 2

Figure 2.8. Final network generated
3.1 Simulated annealing approach

We have developed a simulated annealing (SA) based scheme for topology generation. SA is a random-search technique which exploits analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure. SA’s major advantage over other methods is an ability to avoid becoming trapped in a local minima. The algorithm employs a random search methodology which attempts to find a global minima for a given minimization problem. An acceptance criterion is defined wherein an increase in the objective function is accepted with a probability. The basic elements of a SA implementation includes:

1. A representation of all possible configurations
2. A generator function which produces a change to the existing solution and returns the new configuration
3. A cooling schedule with an initial temperature and a final temperature, a depreciation factor and a limit to the number of tries at each temperature value
4. An acceptance criterion which states that a new solution is accepted with a probability
5. A depreciation factor for the temperature
6. A randomly generated initial solution
Figure 3.1. Top level simulated annealing module

<table>
<thead>
<tr>
<th>Notation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(C \Rightarrow R)$</td>
<td>A mapping relation where core C is mapped to Router R</td>
</tr>
<tr>
<td>$F_{ij}$</td>
<td>A function which returns routers in flow between core i and core j</td>
</tr>
<tr>
<td>$\chi$</td>
<td>A configuration representing M and F</td>
</tr>
<tr>
<td>$\chi_{initial}$</td>
<td>Initial random configuration</td>
</tr>
<tr>
<td>$Temp_{initial}$</td>
<td>Initial temperature</td>
</tr>
<tr>
<td>$Temp_{final}$</td>
<td>Final temperature</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Depreciation factor</td>
</tr>
<tr>
<td>$N_{tries}$</td>
<td>Number of tries to find a valid solution at each temperature</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Solution validity Indicator</td>
</tr>
<tr>
<td>$A$</td>
<td>Total router area</td>
</tr>
<tr>
<td>$\chi_{final}$</td>
<td>Final configuration</td>
</tr>
</tbody>
</table>

Table 3.1. Notations used in Simulated Annealing
We start with a random configuration wherein cores are randomly assigned to routers. We make sure that the partition size is balanced. The STEP function shown in algorithm 6 generates a core to router mapping and a path for each flows. Number of hops for each flow is randomly picked between 1 and a maximum value depending on the cost of the flow. Routers for the flow is randomly picked. Thus generated configuration is checked to see if it meets the deadline using SCHEDULE_LENGTH function in VALIDITY_CHECK module shown algorithm 5. If the configuration meets the deadline, it is accepted otherwise an INVALID is returned. At each temperature $N_{\text{tries}}$ are performed to obtain a VALID solution. If a solution is valid, then the total router area is calculated and if it is less than the previously obtained value, the solution is made the final, otherwise it is accepted with a probability shown in algorithm 4. A geometric depreciation factor was used as it is known to give good results [36]. The goal for the simulated annealing module is to minimize total router area.

3.2 Static graph based greedy approach

We have developed a methodology based on a static graph which consists average bandwidth information between nodes. An example static graph is shown in figure 3.2. The motivation behind this approach is to greedily assign lower hops/direct links to the flows which have high average bandwidth and lower average bandwidth flows get higher hops. The deadline or the slack is not visible to this approach. We use a Breadth First Search(BFS) algorithm to search paths between routers. We start with no. of routers equal to no. of cores. The communication flows are sorted in non-increasing order of average bandwidth information. Higher bandwidth flows get lower hops and lower bandwidth flows get higher hops. Algorithm 7 shows the static graph based greedy network topology generation.
Algorithm 4 Module SIMULATED_ANNEALING()

Input: $Temp_{initial}, Temp_{final}, \alpha, N_{tries}, \chi_{initial}$
Output: $A_{final}, \chi_{final}$

Begin:
\[ temp = Temp_{initial} \]
\[ \chi = \chi_{initial} \]
\[ \chi_{final} = \text{INVALID} \]
\[ A_{final} = \text{INVALID} \]

while $temp > Temp_{final}$ do
\[
\text{for } i \text{ in } 1..N_{tries} \text{ do }
\]
\[ \chi_{new} = \text{STEP()} \]
\[ \Phi = \text{VALIDITY_CHECK}(\chi_{new}) \]
if $\Phi$ is VALID then
\[ \delta = A_{new} - A_{current} \]
if $(\delta < 0)$ OR $(e^{-\delta/temp} > \text{RANDOM}(0,1))$ then
\[ \chi = \chi_{new} \]
end if
\[ \chi_{final} = \chi \]
end for
\[ temp = temp * \alpha \]
end while
return $\chi_{final}$

End

Algorithm 5 Module VALIDITY_CHECK()

Input: $\chi$
Output: $\Phi$

if $\text{SCHEDULE\_LENGTH}(\chi) \leq \text{DEADLINE}$ then
return $\Phi$ VALID
else
return $\Phi$ INVALID
end if

Algorithm 6 Module STEP()

Output: $\chi$
\[ M = \text{SWAP\_CORES()} \]
for all (flows between core$_i$ and core$_j$) do
\[ \text{MAXHOP}_{ij} = \text{ceil}(\text{MaxCost}/\text{cost}_{ij}) \]
\[ F_{ij} = \text{RANDOM}(1,\text{MAXHOP}_{ij}) \]
end for
return $\chi(M, F)$
Figure 3.2. Static graph showing average bandwidth information

Algorithm 7 Module STATIC_GRAPH_GREEDY()

1: Map each core to a router
2: Sort average bandwidth in non-increasing order
3: Higher bandwidth flows get lower hops.Map bandwidth to hops for each flow
4: for hop_ij in 1..MaxHopCount do
5:     for all flows between core i and core j do
6:         if hop_ij is 1 then
7:             Open a direct link between the router_i and router_j
8:         else
9:             found=BFS(router_i,router_j)
10:            if found is 0 then
11:                Add links between router_i and router_j via existing connections with
12:                   maximum hops = hop_ij
13:           end if
14:        end if
15:     end for
16: end for
17: Use Dijsktara algorithm to find shortest path
CHAPTER 4
EXPERIMENTAL RESULTS

4.1 Experimental setup

To validate the proposed methodology experiments were conducted using synthetic benchmarks. The experimental set up is shown in Figure 4.1. The algorithms were implemented in object oriented perl. TGFF [25] was used to generate a number of task graphs with varying complexity. Ganeshpure[34] presented a task graph simulator and we used this for our task graph related calculations. Router delay models were obtained from [35]. Table 4.1 shows the list of benchmarks used. It shows the number of cores in the design, number of nodes and edges in the task graph. The baseline represents the best case schedule length with a point to point connection between the cores. The computation to communication ratio represents the ratio between the task execution time and the maximum value of the corresponding communication time. The number of passes for KL and recursive KL algorithm was set to 100. Each case of router count was sampled 100 times and the best result was chosen. The runtime variation of task execution times was modelled by assuming a Gaussian distribution, with $3\sigma$ variation defining the maximum and minimum value and a mean value midway between them. Each run was sampled for 1000 cases and the effect of standard deviation on the schedule length was analysed. We used Orion simulator [32] to calculate the total router area.
Figure 4.1. Experimental set up

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Cores</th>
<th>Nodes</th>
<th>Edges</th>
<th>Baseline</th>
<th>Deadline</th>
<th>Communication ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark1</td>
<td>8</td>
<td>20</td>
<td>24</td>
<td>8100</td>
<td>8300</td>
<td>1:0.01</td>
</tr>
<tr>
<td>Benchmark2</td>
<td>6</td>
<td>15</td>
<td>16</td>
<td>1700</td>
<td>4200</td>
<td>1:0.3</td>
</tr>
<tr>
<td>Benchmark3</td>
<td>9</td>
<td>35</td>
<td>49</td>
<td>3450</td>
<td>8000</td>
<td>1:0.5</td>
</tr>
<tr>
<td>Benchmark4</td>
<td>30</td>
<td>71</td>
<td>100</td>
<td>4459</td>
<td>20000</td>
<td>1:0.6</td>
</tr>
<tr>
<td>Benchmark5</td>
<td>16</td>
<td>52</td>
<td>83</td>
<td>4748</td>
<td>22000</td>
<td>1:1</td>
</tr>
<tr>
<td>Benchmark6</td>
<td>49</td>
<td>96</td>
<td>134</td>
<td>3238</td>
<td>25000</td>
<td>1:1</td>
</tr>
</tbody>
</table>

Table 4.1. Benchmarks showing number of cores, nodes and edges

<table>
<thead>
<tr>
<th>Specification</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual channels per input port</td>
<td>2</td>
</tr>
<tr>
<td>Flit width</td>
<td>32</td>
</tr>
<tr>
<td>Buffer depth</td>
<td>4</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>1GHz</td>
</tr>
<tr>
<td>Technology node</td>
<td>45nm</td>
</tr>
</tbody>
</table>

Table 4.2. Router specifications used in the simulations
**Figure 4.2.** Benchmark profiling to obtain communication to computation ratio for various multimedia benchmarks

<table>
<thead>
<tr>
<th>Specification</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU clock</td>
<td>1.2GHz</td>
</tr>
<tr>
<td>L1 I-cache</td>
<td>32KB</td>
</tr>
<tr>
<td>L1 D-cache</td>
<td>32KB</td>
</tr>
<tr>
<td>L2 cache</td>
<td>512KB 2-way</td>
</tr>
<tr>
<td>Technology node</td>
<td>45nm</td>
</tr>
</tbody>
</table>

**Table 4.3.** Processor configuration for benchmark profiling
4.2 Benchmark profiling

We used synthetic benchmarks generated by TGFF. To make sure that these benchmarks are close to the real world application, we ran various multimedia benchmarks and observed the computation cycles. We used the Intel CE4100 multimedia SoC [37] configuration and we used Gem 5 simulator [38]. We used a single core processor and the goal in doing such a profiling was to obtain the total cycles spent in computation and the total cycles spent in communicating with the main memory. This ratio allows us to carefully craft the communication costs in the synthetic benchmarks generated by TGFF. Figure 4.2 shows the ratio obtained for various multimedia benchmarks. We observed that few benchmarks are computation dominant like the fft and ifft which have very low ratio of 0.002 and benchmarks like crc and susansmoothing have a ratio of about 0.2 indicating a good mix of computation and communication. For our benchmarks a ratio was picked randomly between 0.01 to 1.

4.3 Effect of coefficient of cost function on schedule length

In Figure 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 the schedule for best router case is plotted against the k - coefficient of cost function for various benchmarks. As we increase k, the contribution of slack in the cost function increases. Lower slack and higher bandwidth communication flows get priority and lower hops are assigned to them. This becomes evident in the total schedule length getting better with increase in k. The cost function cannot have a negative value and hence k is swept between a valid range starting from 0. The value which finds the best schedule length is chosen for the entire set of simulations. For BM5 the deadline was set too far from the baseline. All the communications get huge slack and the schedule length becomes independent of k. As we can see in Figure 4.7 the variation in schedule length is very small and almost independent of k.
Figure 4.3. Schedule length plotted against $k$ - coefficient of cost function for Benchmark 1

Figure 4.4. Schedule length plotted against $k$ - coefficient of cost function for Benchmark 2
Figure 4.5. Schedule length plotted against k - coefficient of cost function for Benchmark 3

Figure 4.6. Schedule length plotted against k - coefficient of cost function for Benchmark 4
Figure 4.7. Schedule length plotted against $k$ - coefficient of cost function for Benchmark 5

Figure 4.8. Schedule length plotted against $k$ - coefficient of cost function for Benchmark 6
4.4 Effect of deadlines on coefficient of cost function

$k$, which is the coefficient of cost function in equation 2.1 is a weighting parameter which dictates the contribution of the slack in the cost function. Since cost is a positive entity, we need to make sure that the cost function doesn’t go negative, so an upper bound on the coefficient is needed. The minimum slack available for a given benchmark depends on the deadline, if the deadline is tending towards the baseline then the slack tends to zero and if the deadline is placed far away from the baseline the slack available will be high. When the slack available is high, the coefficient of cost function needs to take small value so as to protect the cost function from dropping to a negative value. Figure 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 shows the plot of upper bound on the coefficient of cost function plotted against deadlines for various benchmarks.

![Graph showing upper bound on coefficient of cost function against deadline for Benchmark 1](image)

**Figure 4.9.** $k$ - coefficient of cost function plotted against deadline for Benchmark 1

4.5 Effect of router count on schedule length and total area

Our design flow is shown in Figure 2.2. The number of routers are varied from 1 to a value equal to number of cores. After fixing the number of routers the cores are mapped to routers using Kernighan-Lin and the topology is generated using recursive
Figure 4.10. $k$ - coefficient of cost function plotted against deadline for Benchmark 2

Figure 4.11. $k$ - coefficient of cost function plotted against deadline for Benchmark 3
Figure 4.12. $k$ - coefficient of cost function plotted against deadline for Benchmark 4

Figure 4.13. $k$ - coefficient of cost function plotted against deadline for Benchmark 5
Figure 4.14. k - coefficient of cost function plotted against deadline for Benchmark 6

KL. In simulated annealing based scheme we follow a similar procedure wherein the router count is varied from 1 to number of cores. The simulated annealing based scheme is shown in figure 3.1. The results for various benchmarks are shown in figure 4.15, 4.16, 4.17, 4.18, 4.19, 4.20 and DL represents the deadline used for each benchmark. It was observed that as we increase the no. of routers, each communication flow observes a greater no. of hops which results in an increase in the total schedule length which eventually leads to deadlines being missed. The cases which don’t meet the deadline are discarded. Small no. of routers have smaller no. of hops for each communication flow but end up having large no. of ports per router. Hence even though they meet the deadline, these candidates are discarded if they violate the maximum no. of ports allowed per router. The maximum no. of ports allowed per router was set to 5 and we used mesh topology as a reference to set this value. So we observed that we end up with 2 to 3 potential candidates for the final best topology selection. To further converge on to the best topology we use statistical modelling of task execution times to find the topology which maximizes the percentage cases meeting the deadline.
Figure 4.15. Schedule length plotted against various router scenarios for Benchmark 1

Figure 4.16. Schedule length plotted against various router scenarios for Benchmark 2
Figure 4.17. Schedule length plotted against various router scenarios for Benchmark 3.

Figure 4.18. Schedule length plotted against various router scenarios for Benchmark 4.
Figure 4.19. Schedule length plotted against various router scenarios for Benchmark 5.

Figure 4.20. Schedule length plotted against various router scenarios for Benchmark 6.
4.6 Effect of variation in task execution times

The runtime variation of the task execution times were modeled as a normal distribution and each run was sampled for 1000 cases. The statistical simulations are run on the potential candidates for best topology obtained after applying constraint 1 and constraint 2 (refer figure 2.2). Figure 4.21,4.22,4.23,4.24 shows the variation for different benchmarks plotted against standard deviation. The motivation behind this experiment was to analyse the effect of runtime variation and to determine the capability of the network to handle such a variation. Our algorithms make sure that critical communications are given smaller hops and even if there is a runtime variation for task execution time in the critical paths, deadlines will not be missed. Percentage cases meeting the deadline was observed and network is marked as a good topology if this percentage meets the design requirement. This serves as an important feedback in the design process which can help in tuning the design parameters.

![Figure 4.21](image_url)

Figure 4.21. Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM1
Figure 4.22. Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM2

Figure 4.23. Plot showing percentage cases meeting the deadline for different standard deviation under different deadlines for BM3
4.7 Comparison with various schemes

4.7.1 Performance comparison

Figure 4.25 show the schedule length in cycles the with mean execution times for the tasks for all the benchmarks. The dimension of the mesh for each benchmark is shown on the x-axis. For BM1 all schemes gave approximately same performance. This is because it was computation dominant benchmark, the communication cost was too low (refer table 4.1). For BM2, mesh performed better among all the schemes. Since BM2 is a small benchmark with just 6 cores, mesh was found to be the best network topology. For rest of the benchmarks our scheme was the best among all the 4 schemes. Static graph based greedy approach gave the worst performance. This is because the scheme assigned lower hops to flows which had higher average bandwidth. But these flows may not be critical resulting in a higher schedule length. A best case performance benefit of 35% over mesh topology, 28.25% over SA based scheme and 37% over static graph based greedy approach was observed.
4.7.2 Total router area comparison

Table 4.4 shows total router area obtained from various schemes. For our scheme best topology was chosen based on the statistical analysis: the topology which maximizes the percentage cases meeting the deadline. Router area is quadratic with no. of ports and hence links/ports have to be carefully added. Our scheme was extremely effective in adding links to the routers due to which significant area benefits were observed over all the other schemes. A best case benefit of 53% over mesh, 85% over SA and 57% over static graph based greedy approach was observed. For SA based scheme the quality of the solution depends on the runtime. As we can observe from the table 4.4, for the same runtime the bigger benchmarks give bigger benefit over SA. This is because as the benchmark size increases the search space of the solution increases and SA needs longer runtime to find a good solution.

4.7.3 Quality metrics

We have defined few metrics to analyse the quality of the generated network topology. The three quality metrics defined are average port count, average hop
Table 4.4. Table showing total router area in $\mu m^2$ for all the schemes and % benefit over all the schemes

<table>
<thead>
<tr>
<th></th>
<th>Our scheme</th>
<th>SA</th>
<th>Mesh</th>
<th>Greedy</th>
<th>%SA</th>
<th>%Mesh</th>
<th>%Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM1</td>
<td>45913.08</td>
<td>75753.8</td>
<td>91826.16</td>
<td>112114.58</td>
<td>49.12</td>
<td>50</td>
<td>59.04</td>
</tr>
<tr>
<td>BM2</td>
<td>38536.45</td>
<td>67033</td>
<td>65012.96</td>
<td>85301.38</td>
<td>42.51</td>
<td>40.72</td>
<td>54.82</td>
</tr>
<tr>
<td>BM3</td>
<td>55799.71</td>
<td>92858</td>
<td>109424.46</td>
<td>118145.26</td>
<td>39.90</td>
<td>49</td>
<td>52.77</td>
</tr>
<tr>
<td>BM4</td>
<td>228939.92</td>
<td>1075775.1</td>
<td>437071.76</td>
<td>488123.72</td>
<td>78.71</td>
<td>47.61</td>
<td>53.09</td>
</tr>
<tr>
<td>BM5</td>
<td>113952.85</td>
<td>312903.7</td>
<td>215845.76</td>
<td>265682.91</td>
<td>63.58</td>
<td>47.20</td>
<td>57.10</td>
</tr>
<tr>
<td>BM6</td>
<td>343767.22</td>
<td>2377048.5</td>
<td>746289.26</td>
<td>782324.33</td>
<td>85.53</td>
<td>53.93</td>
<td>56.05</td>
</tr>
</tbody>
</table>

count and worst case hop count. Average port count is defined as:

$$APC = \frac{1}{N} \sum_{i=1}^{N} P_i/N \quad (4.1)$$

where

$P_i = \text{No. of ports in router } i$

$N = \text{Total no. of routers}$

The average port count metric defines the uniformity of the network. Figure 4.26 shows the average port count for various benchmarks. This metric makes sure that all the routers have approximately same no. of ports which will help in smooth place and route process. The average port count obtained from our scheme was comparable to the mesh. For BM4, BM5 and BM6, SA scheme had a large average port count. This shows that SA scheme tried to assign equal priority hops to all the flows. The second metric is the average hop count which is the average no. of hops of the network and is shown for all the benchmarks in Figure 4.27. It was observed that the average no. of hops from our scheme is comparable to Mesh topology. For BM3 our scheme performed better over the mesh. The third metric is the worst case hop count and is shown for all the benchmarks in Figure 4.28. It was observed that SA scheme had the least worst case hop count. Our scheme did comparable to mesh, the greedy
approach had bigger worst case hops as the scheme assigns bigger hops for low average bandwidth flows.

Figure 4.26. Plot showing average port count of various schemes for all the benchmarks
Figure 4.27. Plot showing average hop count of various schemes for all the benchmarks.

Figure 4.28. Plot showing average hop count of various schemes for all the benchmarks.
CHAPTER 5

CONCLUSION

A methodology for interconnection network topology generation for multicore system-on-chip was presented. Given a task graph, a corresponding task to processor schedule and a deadline, we have developed a recursive partitioning methodology based on Kernighan-Lin algorithm to generate an efficient interconnection network for the cores. A 2-phase approach was adopted wherein the first phase was generation of the network and in the second phase runtime variations of the task execution times were modeled as Gaussian distribution and results from statistical simulations were presented. Also, effect of delay models and task communication times on the deadlines were analyzed. To best of our knowledge this is the first approach wherein a complete design methodology for network synthesis which includes statistical modelling of runtime variations has been presented. The results from proposed solution were compared against simulated annealing based approach, static graph based greedy approach and standard mesh topology. The proposed solution offers significant area and performance benefits over the alternate approaches.
BIBLIOGRAPHY


[21] K. Lahiri et al., *Design space exploration for optimizing on-chip communication architectures* IEEE Trans. on CAD, 23(6), 2004


[37] Intel Corporation, Intel CE 4100 Media Processor

[38] www.m5sim.org