COLLECTIVE COMMUNICATION-AWARE HIGH PERFORMANCE NETWORK-ON-CHIP ARCHITECTURES FOR BIG DATA PROCESSING

By

KARTHI DURAISAMY

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

DECEMBER 2017

© Copyright by KARTHI DURAISAMY, 2017
All Rights Reserved
To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of KARTHI DURAIYAMY find it satisfactory and recommend that it be accepted.

Partha Pratim Pande, Ph.D., Chair

Ananth Kalyanaraman, Ph.D.

Deuk Heo, Ph.D.
ACKNOWLEDGMENT

As I finish this dissertation, I would like to acknowledge the people who supported me over the years in many ways and made this dissertation possible.

I express my utmost gratitude to my Ph.D. advisor and an ardent mentor, Dr. Partha Pratim Pande, for his thorough guidance over the last four years. His passion for work is contagious and has definitely kept me focused and motivated over the course of my PhD life. He gave me the freedom to develop my own ideas and has always been available to tackle the various problems that I have encountered.

I would like to thank Dr. Ananth Kalyanaraman for providing invaluable advice and guidance, both at work and otherwise. All the works that I have published with Dr. Ananth have played a vital role in shaping up this thesis.

I thank Dr. Deukhyoun Heo for being a member of my Ph.D. committee and providing precious feedback. He is a favorite teacher of mine and his lectures have taught me invaluable lessons in approaching, learning and teaching a topic.

I sincerely thank all the co-authors of my journals and conferences publications. All their contributions have shaped and enriched this dissertation.

Aside from research, many great people have helped me out in various ways when I worked towards this dissertation. I thank my friends, Wonje Choi, Dr. Ryan Kim and Dongjin Lee for also being amazing colleagues. Whenever I faced a research problem that needed an argumentative discussion or a personal issue that must get a quick resolution or an illness that needed a hospital visit, I have always depended on you three. I would also like to thank
all my fellow colleagues in the Low Power and Robust Nanosystems Lab for all the support and I wish you all the best.

I sincerely appreciate the support and love provided by my mother, father and my extended family. Surely, without them this work would not have been possible.

Finally, I thank the National Science Foundation for the grants CNS 1564014, CCF 1514269 and CCF 1162202 that have helped support my doctoral dissertation research.
COLLECTIVE COMMUNICATION-AWARE HIGH PERFORMANCE NETWORK-ON-CHIP ARCHITECTURES FOR BIG DATA PROCESSING

Abstract

by Karthi Duraisamy, Ph.D.
Washington State University
December 2017

Chair: Partha Pratim Pande

In an era when power constraints and data movement are proving to be significant barriers for high-end computing, manycore architectures offer a low power and highly scalable platform suitable for both data- and compute-intensive applications. The performance of a manycore architecture is highly dependent on the capabilities of its communication backbone, namely the Network on-chip (NoC). An efficient NoC designed for a manycore platform must align the connectivity of the NoC with the application's on-chip traffic patterns.

Analysis on the inter-core traffic patterns exhibited by various Big Data applications reveal irregular memory access behaviors that give rise to long-range on-chip communication requirements. In addition to the irregular memory access patterns, many of the modern applications also necessitate dense collective communication capabilities among the on-chip nodes. Under collective communication, either a single source node transmits
data to all the other nodes in the system (one-to-all) or all the nodes in the system communicate with a single destination node (all-to-one).

Wireless NoC is an emerging paradigm to design high bandwidth and energy efficient communication backbone for manycore chips. Previous works show that the wireless links can establish low-latency data-transfers even between physically distant on-chip nodes. In addition, with its inherent broadcast capability, the on-chip wireless links are highly suited to perform efficient collective-communication. Thus, employing on-chip wireless links one can design efficient communication infrastructures for manycore platforms running high performance Big Data processing.

This dissertation focuses on designing a hybrid (wireline + wireless) network-on-chip architecture (called WiNoC) capable of low-latency collective communication. First, we leverage the knowledge gained from studying small-world graphs to design low hop count WiNoC topologies. Next, we augment the WiNoC with suitable data-transfer mechanisms to ensure a congestion-free high performance NoC.

On overall, this work indicates on-chip communication challenges arise from manycore Big Data processing and proposes a wireless-enabled high performance and energy efficient NoC capable of addressing these challenges.
 TABLE OF CONTENTS

ACKNOWLEDGMENTS .......................................................................................................................... iii

ABSTRACT ........................................................................................................................................ v

LIST OF TABLES................................................................................................................................. x

LIST OF FIGURES............................................................................................................................... xi

ABBREVIATIONS............................................................................................................................... xvi

CHAPTERS

1 Introduction ......................................................................................................................................... 1
   1.1 Manycore Processing: An Efficient Approach for Big Data Analytics ................................. 3
   1.2 Traditional Manycore Platforms and Big Data Processing ................................................... 4
   1.3 Wireless Network-on-Chip for Big Data Processing............................................................... 7

2 Related Works ................................................................................................................................... 9
   2.1 Computation Frameworks and Processing Platforms for Big Data ................................. 9
   2.2 NoCs Incorporating Low-Hop Count Interconnection Topologies ..................................... 10
   2.3 Collective Communication-Aware NoCs............................................................................... 13
   2.4 NoCs Performing Single-Cycle Multi-hop Traversals........................................................... 15
   2.5 Summary of Related Works and Dissertation Organization .............................................. 17

3 Analysis of Big Data Analytics on Manycore Platforms ............................................................ 19
   3.1 Computation Patterns of Big Data Applications.................................................................. 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Performance Evaluation of WiSMART NoC</td>
<td>154</td>
</tr>
<tr>
<td>6.3</td>
<td>Summary</td>
<td>168</td>
</tr>
<tr>
<td>7</td>
<td>Future Research Directions</td>
<td>169</td>
</tr>
<tr>
<td>7.1</td>
<td>Hardware-Software Co-design Addressing Memory Access Complexities</td>
<td>169</td>
</tr>
<tr>
<td>7.2</td>
<td>Heterogeneous Manycore Platforms for Big Data Processing</td>
<td>177</td>
</tr>
<tr>
<td>7.3</td>
<td>Integrating SMART Control Network in Emerging Interconnects</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>179</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1 Phoenix++ MapReduce: applications analyzed and the datasets employed........ 21
Table 3.2 DIMACS10 graphs used for Grappolo analyses.................................................. 27
Table 3.3 DIMACS10 graphs used for Balanced Coloring analyses.......................... 27
Table 4.1 V/F assignments for MapReduce Applications.................................................. 95
Table 5.1 Information needed for MALASH routing.........................................................116
Table 5.2 MALASH: Routing rules that are followed at all routers..............................116
Table 6.1 The rules that must be satisfied by the source WI coordinates to avoid possible deadlocks arising from the re-routing mechanism used for handling WI failures.................135
Table 6.2 SSR fields associated with the WiSMART NoC.................................................139
Table 6.3 All the pre-SSR values for collective communication aware WiSMART.............148
Table 7.1 Evaluation of the approximate update scheme (via ET) on community detection’s runtime and modularity.................................................................173
Table 7.2 Evaluation of the reduction in number of edges traversals (NET) and number of community lookups (NCL), induced by ET..................................................174
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A Traditional 3 Tier Electrical Packet Switching Data Center Network.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Block Diagram of a Tiled Manycore Platform.</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Execution flow of Phoenix++ MapReduce.</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Pseudocodes for Graph Algorithms.</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>Traffic Characteristics: Fraction of total traffic exchanged between the communicating cores separated by x-hops in a 2D mesh NoC.</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Percentage of traffic injected by the top six traffic hotspots in Big Data applications.</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Core utilization values for MapReduce applications on a 64-core platform.</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Heat plots (of order 64×64) showing the traffic intensity between different source-destination pairs.</td>
<td>39</td>
</tr>
<tr>
<td>3.7</td>
<td>Heat Plots Representing traffic matrices (of order 64×64) for Grappolo and Balanced Coloring with Different Input Graphs.</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>Average flit intensity with Hammer and Directory protocols.</td>
<td>42</td>
</tr>
<tr>
<td>3.9</td>
<td>Percentage of traffic arise due to multicast injections.</td>
<td>42</td>
</tr>
<tr>
<td>3.10</td>
<td>Average destination count per multicast injection.</td>
<td>44</td>
</tr>
<tr>
<td>3.11</td>
<td>Destination quadrant counts for Multicasts (averages for all 10 applications).</td>
<td>44</td>
</tr>
<tr>
<td>3.12</td>
<td>Multicast injection pattern: Number of multicast flits injected within a bin width of 10 cycles for (a) FFT and (b) Kmeans applications.</td>
<td>45</td>
</tr>
</tbody>
</table>
Figure 3.13 Cause for congestion: % of multicast injections that arrives before or after another multicast injection (within a period of 5 cycles). .............................................................. 46
Figure 4.1 Wireline and wireless EDPs for transferring one-bit over varying link lengths.. 51
Figure 4.2 Flow chart explaining the operation of distributed MAC protocol. ....................... 57
Figure 4.3 Sample node selection scenarios under distributed MAC. X marks represent channel requests. ......................................................................................................................... 58
Figure 4.4 Distributed MAC: Order of the nodes for request selection.............................. 58
Figure 4.5 Various NoC Architectures illustrating data transfers........................................ 61
Figure 4.6 Flow Chart explaining WiFBF routing.................................................................... 65
Figure 4.7 A simulated annealing based algorithm to identify the placement of wireless interfaces in WiMesh and WiNoC. ......................................................................................... 68
Figure 4.8 Latency and wireless utilization improvement achieved with basic distributed MAC (BDM) over token MAC for varying WI counts. ................................................................. 71
Figure 4.9 Channel Access Time of the two MAC schemes. .................................................. 71
Figure 4.10 Determining the Optimum Number of Wireless Interfaces............................... 73
Figure 4.11 Comparative Analyses of Network Performances. For each NoC architecture, both wireline only and wireless enabled topologies are considered here. ......................... 75
Figure 4.12 NoC Robustness: Increase in EDP against increase in no., of failed WIs........... 76
Figure 4.13 Grappolo: Execution Times: Shown values are normalized with respect to mesh NoC execution times. ........................................................................................................ 78
Figure 4.14 Grappolo Runtime Characteristics........................................................................ 79
Figure 4.15 Balanced Coloring: Execution Times................................................................. 80
Figure 4.16 Balanced Coloring: Traffic Injection Rates........................................................ 81
Figure 4.17 Balanced Coloring- CAD and CAC Runtime Characteristics ......................................................... 82
Figure 4.18 Graph applications: Full System Energy Consumption and EDP Characteristics ......................................................... 84
Figure 4.19 Adopted VFI design Flow for MapReduce Applications .......................................................... 92
Figure 4.20 MapReduce VFI: Comparison between VFI 1 and VFI 2 systems .............................................. 94
Figure 4.21 MapReduce Applications: Core utilization trend ................................................................. 95
Figure 4.22 EDP of the maximized wireless usage relative to the minimized hop count wireless placement methodology .................................................................................................................. 96
Figure 4.23 Normalized execution time of each Phoenix++ execution operation with respect to non-NVFI mesh .................................................................................................................................................. 97
Figure 4.24 MapReduce applications: Full-System Energy-Delay-Product for VFI Mesh and VFI WiNoC with respect to NVFI Mesh ............................................................................................................. 99
Figure 5.1 A sample WiNoC multicast scenario ............................................................................................. 104
Figure 5.2 Multicast-aware WiNoC: Flow diagram for establishing wireline connectivity 105
Figure 5.3 The overall WiNoC design flow to enable MALASH routing ...................................................... 106
Figure 5.4 Sample WiNoC multicast distribution scenarios in quadrant Q2. ............................................. 112
Figure 5.5 Flow chart explaining MALASH routing ...................................................................................... 114
Figure 5.6 Message latency and energy improvements achieved with DPM over the token passing MAC for multicast traffic .............................................................................................................. 118
Figure 5.7 Comparison of multicast-aware mesh and WiNoC network latency ........................................ 121
Figure 5.8 Execution Time comparisons demonstrating WiNoC Efficiency .............................................. 123
Figure 5.9 Energy comparisons demonstrating WiNoC Efficiency ........................................................... 124
Figure 6.1 1D RSMART mesh labeled with the router stages involved in transmitting data from R0 to R2.................................................................128
Figure 6.2 Performance of RSMART mesh NoCs with varying \(LPC_{max}\).................................130
Figure 6.3 Placement of wireless interfaces in a WiSMART NoC..............................................132
Figure 6.4 The pre-SSR values for WIs located in the different positions...............................136
Figure 6.5 Transfer of pre-SSR (S to SWI) in 3 cycles using look-ahead requests. Only necessary pre-SSR wires are shown.........................................................137
Figure 6.6 Look-ahead request based routing in WiSMART NoC.............................................138
Figure 6.7 Transmission of unicast pre-SSRs in the WiSMART NoC.......................................140
Figure 6.8 SMART and WiSMART broadcast distribution......................................................142
Figure 6.9 Collective Communication enabled WiSMART NoC Creation..................................151
Figure 6.10 Zero load latency of 256-core SMART NoCs for different clock frequencies with UR unicast and broadcast traffic .......................................................155
Figure 6.11 Zero load unicast and broadcast distribution latency under UR traffic for different SMART-hop latencies (3 and 4 cycles).................................................................156
Figure 6.12 Comparing the zero-load latency of 256-core 1D RSMART, 1D RSMART with look-ahead pre-SSR and WiSMART NoCs..........................................................157
Figure 6.13 Sample WiMesh multicast scenarios....................................................................159
Figure 6.14 Average flit latencies for the RSMART, WISMART and High Radix NoC s. ..........160
Figure 6.15 On-chip Traffic Distribution under Directory Protocol..........................................162
Figure 6.16 Comparison of full system EDP with different coherence protocols......................164
Figure 6.17 Connection overheads caused by router-bypass setup networks in SMART, RSMART and WiSAMRT NoCs.................................................................167
Figure 7.1 Effect of early termination on the convergence rate of modularity, and the number of vertices processed per iteration ................................................................. 172

Figure 7.2 Ex. Times of FS and FS+ET versions of Grappolo. ................................................................. 175

Figure 7.3 FS and FS+ET versions of Grappolo: Comparison of Full System Energy Consumptions. ........................................................................................................ 176
### ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>1 Dimensional</td>
</tr>
<tr>
<td>2D</td>
<td>2 Dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>3 Dimensional</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>ALASH</td>
<td>Adaptive Layered Shortest Path</td>
</tr>
<tr>
<td>ASJ</td>
<td>as-22july06</td>
</tr>
<tr>
<td>ASTRO</td>
<td>Astro-ph</td>
</tr>
<tr>
<td>BC</td>
<td>Balanced Coloring</td>
</tr>
<tr>
<td>BDM</td>
<td>Basic Distributed MAC</td>
</tr>
<tr>
<td>BLG</td>
<td>Belgium</td>
</tr>
<tr>
<td>CAC</td>
<td>coAuthorsCiteeseer</td>
</tr>
<tr>
<td>CAD</td>
<td>coAuthorsDBLP</td>
</tr>
<tr>
<td>CCR</td>
<td>citationCiteeseer</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal–Oxide–Semiconductor</td>
</tr>
<tr>
<td>CNL</td>
<td>Canneal</td>
</tr>
<tr>
<td>CNR</td>
<td>cnr-2000</td>
</tr>
<tr>
<td>COND</td>
<td>Cond-mat-2003</td>
</tr>
<tr>
<td>CSR</td>
<td>Compressed Sparse Row</td>
</tr>
<tr>
<td>DPM</td>
<td>Distributed Priority Multicast MAC</td>
</tr>
<tr>
<td>EDP</td>
<td>Energy Delay Product</td>
</tr>
<tr>
<td>ET</td>
<td>Early Termination</td>
</tr>
<tr>
<td>FBF</td>
<td>Flattened ButterFly</td>
</tr>
<tr>
<td>FET</td>
<td>Field-Effect Transistor</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FLD</td>
<td>Fluidanimate</td>
</tr>
<tr>
<td>FS</td>
<td>Fully Synchronized</td>
</tr>
<tr>
<td>FS+ET</td>
<td>Fully Synchronized + Early Termination</td>
</tr>
</tbody>
</table>
GHz  Giga Hertz
GP   Grappolo
$H_{avg}$ Average hops
HEP  Hep-th
HIST Histogram
$HPC_{max}$ Maximum Hops Per Cycle
HR-FB High Radix - Flattened Butterfly
$K_{avg}$ Average number of ports per router
KM   Kmeans
$K_{max}$ Maximum number of ports per router
L1 Cache Level1 Cache
L2 Cache Level1 Cache
LASH Layered Shortest Path
$LPC_{max}$ Maximum Hops Per Cycle
LR   Linear Regression
LU decomposition Lower Upper decomposition
MAC  Medium Access Control
MALASH Multicast-aware Adaptive Layered Shortest Path
MM  Matrix Multiplication
mm  millimeter
NC  Network Coding
NCL  Number of Community Lookups
NET Number of Edges Traversed
nm  nanometer
NoC  Network-On-Chip
ns  nano Second
NTH Netherlands
OOK On-Off Keying
OOOK Orthogonal On-Off Keying
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PGP</td>
<td>PGPgiantcompo</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase-Lock Loop</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAD</td>
<td>Radix</td>
</tr>
<tr>
<td>RSMART</td>
<td>Reduced wire SMART</td>
</tr>
<tr>
<td>SMART</td>
<td>Single-Cycle Multi-hop Asynchronous Repeated Traversal</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric MultiProcessor</td>
</tr>
<tr>
<td>SoC</td>
<td>System-On-Chip</td>
</tr>
<tr>
<td>SSR</td>
<td>SMART hop Setup Request</td>
</tr>
<tr>
<td>SW</td>
<td>Small World</td>
</tr>
<tr>
<td>THz</td>
<td>Tera Hertz</td>
</tr>
<tr>
<td>TLB</td>
<td>Translational Look aside Buffer</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-Large-Scale Integration</td>
</tr>
<tr>
<td>WC</td>
<td>Word Count</td>
</tr>
<tr>
<td>WI</td>
<td>Wireless Interface</td>
</tr>
<tr>
<td>WiFBF</td>
<td>Wireless Flattened Butterfly</td>
</tr>
<tr>
<td>WiMesh</td>
<td>Wireless enabled Mesh</td>
</tr>
<tr>
<td>WiNoC</td>
<td>Wireless Network-On-Chip</td>
</tr>
<tr>
<td>WiSMART</td>
<td>Wireless-enabled SMART</td>
</tr>
<tr>
<td>WTR</td>
<td>Water</td>
</tr>
<tr>
<td>ZB</td>
<td>Zetta Bytes</td>
</tr>
</tbody>
</table>
Dedication

I dedicate this dissertation to my mother, Mallika Duraisamy. I thank her for all the love and support that encourages me to chase my dreams. Let this dissertation be a little token of appreciation.
1 Introduction

With an unprecedented growth in computing technology, twentieth century has paved the way for an information revolution. Apart from the gigabytes of data generated by numerous scientific applications such as DNA and RNA sequencing, the usage of cloud services and social networking has also contributed to the boom in the volume of information to be analyzed, classified and stored. Harvesting the endless potential of this information revolution mainly depends on the efficient analysis of large volumes of data (Big Data) generated by it. Enabling fast and energy-efficient Big Data analytics is vital to enable modern discoveries across numerous fields including but not limited to medicinal research, genetic analysis, gravitational wave detection, financial market analysis and astrophysical research.

The volume of Big Data to is only bound to grow further in the upcoming years. A recent white paper by Hewlett Packard Enterprise on Big Data concluded that the global digital information doubles every two years [1]. Cisco Global Cloud Index Forecasts predicts that by 2020, the annual global data center IP traffic will reach 15.3 zettabytes (ZB) growing 3.25 folds from 4.7 ZB per year in 2015 [2]. This growth in data volume forces every web based service provider to add more and more servers to their facilities in the coming years. IDC’s Digital Universe Study further states that more than the high volumes of data that are stored in and retrieved from the cloud, the analysis that is happening around this data is the major
contributor for Big Data [3]. The traditional approach of employing massive data centers for big data analytics has serious limitations in terms of scalability and power consumption to handle the growth expected in data volume [4][5].

Considering all the above facts, a paradigm shift in the approach for designing computation platforms for Big Data analytics is inevitable. Racks-on-chip [6] and data-centers-on-chip [7][8] are a couple of novel approaches proposed to enable new design paradigms for Big Data analytics. Both approaches advocate for integrating a large number of processing elements and the inter-core communication backbone on a single chip designing massive manycore platforms. Towards this, in this dissertation, we specifically focus on enabling high-performance manycore interconnection architectures that are tailor-made to suit the communication needs of manycore Big Data computing.

Figure 1.1 A Traditional 3 Tier Electrical Packet Switching Data Center Network. Here the three levels of switches are Top of the Rack (ToR) switches, aggregation switches and the core switch.
1.1 Manycore Processing: An Efficient Approach for Big Data Analytics

Traditional Big Data analytics employs thousands of servers located in massive server farms to solve complex problems. In server farms, groups of servers are placed in individual racks. All the racks in the farm are then interconnected using multiple layers of electrical switches as shown in Figure 1.1. Such traditional server clusters are associated with several power and data-transfer constraints. Hence, they only offer a limited potential in meeting the ever-growing computational demands of big data analytics. To summarize the limitations of data-farms addressed in the existing literature; i) the servers and the interconnection switches are highly power hungry, ii) with high data loads, the interconnect experiences bandwidth bottle-neck problems, iii) large data centers have costly form factors and require expensive real-estate and iv) reliability of the multi-tier interconnection model is low and hence, redundant switching components are necessary [9].

With the recent enhancements occurring in Very Large Scale Integration (VLSI), technology, manycore architectures offer a low-power and highly scalable processing platform suitable for both data- and compute-intensive applications. As the silicon fabrication technology advances, we can integrate thousands of processing cores on a single die to design manycore platforms capable of massive parallel processing. In addition to delivering a high processing power, when augmented with a suitable data-transfer backbone, the manycore platform also provides an energy-efficient high bandwidth inter-core communication platform. Thus, by designing specialized manycore architectures we can perform a more efficient Big Data computation than employing traditional server clusters for the task.
1.2 Traditional Manycore Platforms and Big Data Processing

A manycore architecture is made of a set of processor tiles and are hence, is also referred as a Tile-processor. All the tiles in the platform can either be homogenous (identical) or heterogeneous. Each processing tile usually consists of one or more processing elements (PEs), a local cache (usually L1) and a network router. A manycore system usually arranges the tiles in multiple rows and columns. Processor tiles are then augmented using an on-chip interconnection network, referred as Network on Chip (NoC). All memory accesses associated with the processor cores happen through the NoC and hence, an efficient NoC architecture is crucial to the performance of any manycore platform. An illustration of Intel’s Knights Landing (also called Xeon Phi™ ) manycore platform [10] is given in Figure 1.2.

Figure 1.2 Block Diagram of a Tiled Manycore Platform. The architecture illustrated here is the Intel’s Knights Landing (also called Xeon Phi™ ) manycore platform. This architecture employs a mesh NoC and each processor tile consists of two CPUs and two vector-processing units (VPUs).
Today’s industry standard manycore platforms such as Intel’s Knights Landing, Tilera and Adapteva Epiphany mainly follow a mesh interconnection topology \([10][11][12][13]\). The advantages of the mesh NoC include i) simple design, ii) regular structure requiring simple routing and iii) easy timing closure as it avoids long-range interconnections. Traditionally, mesh NoCs employ dimension-ordered XY routing \([14]\). In XY routing, whenever a source injects a data flit, the message is first forwarded along the routers in the X-axis, until the message reaches the column of the destination router. From there on, the message is forwarded along the Y-axis routers. Thus, the on-chip communication in a mesh NoC interconnected manycore platform is multi-hop in nature \([15]\). Traditional mesh interconnects follow conventional packet switching architectures that require multiple cycles to traverse each router hop \([15]\).

As we will discuss later in Chapter 3, when run on manycore platforms, Big Data processing gives rise to challenging on-chip traffic patterns. For example, graph analytics applications cause irregular memory access patterns, giving rise to long-range on-chip communication requirements \([16]\). Similarly, the reduce phase in many MapReduce applications require long-range communication \([17]\). In addition to the long-range traffic, many of the modern SoC applications also necessitate collective communication capabilities among the on-chip nodes. Under collective communication, either a single source node transmits data to all the other nodes in the system (one-to-all) or all the nodes in the system communicate with a single destination node (all-to-one). For examples, modern manycore applications such as neuromorphic computing, protein folding analysis and real-time object recognition, all exhibit dense on-chip broadcast/multicast transmissions \([18]\).
Moreover, Big Data analytics is both compute- and data- intensive preferring large-scale manycore platforms integrating hundreds or even thousands of processor-tiles in a single die. For such long-scale manycore platforms, it is essential to identify an appropriate cache-coherence protocol to ensure system scalability. In general, selecting an appropriate cache coherence protocol for a CMP (Chip Multiprocessor) involves analyzing the area and traffic tradeoffs associated with the protocol. Traditional manycore platforms with relatively small number of processor tiles employ the full-bit Directory cache coherence protocol. The Directory protocol is area expensive when implemented and the fraction of Directory cache-coherence overhead (in total silicon area) grows linearly as the number of cores increases. Thus, Directory protocol limits the scalability of the system and hence, is not desirable for large-scale manycore architectures performing Big Data processing. To overcome this issue, modern coherence protocols that involve low area overhead such as AMD's Hammer [19] can be employed in large-scale manycore platforms performing Big Data analytics. However, coherence protocols enabling low-area overhead exhibit multicast, broadcast and acknowledgement bursts, leading to dense collective communication requirements.

To summarize, a specialized NoC architecture for Big Data analytics is expected to efficiently handle long-range on-chip data transfers and must perform congestion-free collective communication. Today's traditional NoC architectures are not highly suitable to handle the complex traffic patterns arise form Big Data processing. As explained above, conventional NoCs are multi-hop in nature. Hence, the long-range data-transfers experience significant network latency. High network latencies eventually lead to long waits for memory accesses,
stalled processor cycles and execution time penalties. In addition, conventional NoCs lack any efficient mechanism to handle dense collective communication traffic.

1.3 Wireless Network-on-Chip for Big Data Processing

Wireless NoC (WiNoC) is an emerging paradigm to design high-bandwidth and energy-efficient communication backbone for many-core chips [20]. The previous works [21] and [22] have already investigated the feasibility of the on-chip wireless communication. The viability of on-chip wireless communication has been already demonstrated through a number of prototype developments [23][24][25]. In addition, a recent study on emerging on-chip interconnects concluded that the radio frequency interconnects such as the millimeter-wave (mm-wave) wireless links and surface-wave interconnects are also more power and cost efficient than the on-chip optical links [26]. Moreover, when compared to surface-wave interconnects, the on-chip wireless technology is more mature and is completely CMOS compatible [20][26], making it the most desirable emerging on-chip interconnection technology for further exploration.

The on-chip wireless links are naturally suited to perform efficient broadcasts and are capable of enabling low-latency data-transfers between physically distant nodes. Because of these features, the wireless links are naturally suited to handle the on-chip data transfers associated with the Big Data processing. Despite the advantages of the on-chip wireless technology, in order to harvest its full potential, more research is required to address various challenges in several areas, including system architecture, circuit design, device fabrication, and computer-aided design [20]. Towards this, in this work, we focus on enhancing the state-
of-the-art on the system-architecture. Inspired by the advantages of on-chip wireless technology, here we demonstrate that by designing novel interconnection architectures incorporating both wireline and wireless links, we can significantly accelerate the Big Data Processing on manycore platforms while also enhancing the energy foot-print of the system. Specifically, the contributions of this dissertation are:

i) An extensive analysis of the on-chip traffic patterns induced by various manycore applications in presence of both Directory and Hammer cache coherence protocols.

ii) Enhancing the state-of-the-art NoCs by integrating on-chip wireless links and identifying the suitable architecture for performing Big Data analytics.

iii) The design of a collective communication-aware wireless NoC incorporating congestion-aware routing and network coding (NC) technique to handle high volumes of collective communication traffic.


v) Thorough comparative evaluations demonstrating the latency, scalability and power efficiency of the proposed wireless NoC architectures.

vi) Introducing the exciting new research problems arise from this work and presenting the preliminary results for the same.
2 Related Works

In recent years, there have been many works exploring the design of high performance low power manycore platforms customized for compute- and data- intensive applications. This chapter summarizes a few of such salient works and their limitations in a few sub-sections.

2.1 Computation Frameworks and Processing Platforms for Big Data

MapReduce is a popular and efficient framework to implement big data computing on large-scale clusters [27]. Phoenix++ [28] framework is a specialized MapReduce implementation targeting shared manycore platforms. Possibilities of reducing on-chip interconnect energy for the Phoenix++ applications through the use of dynamic directories are presented in [11]. Dynamic directories can reduce the number of network control messages in a NoC, leading to energy conservation. The use of a 3Dimensional (3D)-stacked memory-logic system for MapReduce workloads is explored in [3]. This design involves the use of two separate specialized architectures for Map and Reduce phases. The data-intensive Map phase is handled through specialized near-memory processor dies integrated under 3D memory cubes while the compute- and communication-intensive reduce phase is carried out using a separate large-scale manycore architecture.

Designing specialized computation architectures and parallel algorithms for big data analytics has been an area of great interest in recent times [29][30][31]. The Blue Gene
architecture [32] is one such architecture for distributed memory systems, enabling more than $10^{15}$ floating-point operations per second. Blue Gene also enables efficient interconnection of thousands of computing nodes through its advanced 5D torus interconnection topology. A detailed study on using micro-architectures for graph analytics has been presented in [33]. In [34], the performance of Symmetric Multiprocessor (SMP) systems for large-scale graph analytics has been analyzed. There are several previous works exploring the efficiency of SMPs for applications with inherent irregularities [35][36][37]. For two graph operations, community detection and graph coloring, multithreaded implementations for traditional multicore (x86) architectures and also many core architectures such as Tilera have been previously developed [38][39][40][41][42]. In [43], a MapReduce based parallel computing model for large graph analytics has been presented. The model has been shown to be particularly efficient for two popular graph operations, 3-clique enumeration of a graph and computation of clustering coefficient.

2.2 NoCs Incorporating Low-Hop Count Interconnection Topologies

As stated in Chapter 1, in conventional mesh NoCs, the network latency is usually high due to the inherent multi-hop nature of the network. To enable more efficient on-chip communication, a number of previous works have proposed various low-hop count interconnection architectures.

2.2.1 High Radix NoCs

Low-hop count high-radix NoCs are discussed in [44][45] and [46]. In these high-radix on-chip networks, multiple computing cores are connected to a single NoC router and hence a
large number of ports are required in each router. Connecting multiple cores to a single router (called as high radix routers) leads to a lower number of NoC routers and hence enables a low-average hop count among the communicating nodes. The Flattened Butter-Fly (FBF) topology is one such high-radix topology where the maximum hop count between the computing cores can be limited to two. Though high-radix topologies provide a low-hop-count network, designing efficient routers incorporating a large number of router ports is a challenging task [47]. In [46], it is demonstrated that by employing sizzle-switches, it is possible to design four stage routers for the Flattened Butter-Fly topology operating at 3GHz in 32nm technology node. To summarize, the high radix NoCs are limited by the fact that the routers incorporating a large number of ports are complex to design, dissipate high power and require 4-5 pipeline stages.

2.2.2 Insertion of Long-Range Links

It is already shown that by inserting long-range shortcuts in a regular mesh architecture to induce small-world effects[48][49][50], it is possible to lower average hop counts, achieve significant performance gain and lowered energy dissipation compared to the traditional multi-hop mesh networks [48]. The concept of express virtual channels is introduced in [51]. By using virtual express lanes to connect distant cores in the network, it is possible to avoid the router overhead at intermediate nodes, and thereby improve NoC performance in terms of power, latency and throughput. Performance is further improved by incorporating ultra-low latency multi-drop on-chip global lines (G-lines) for flow control signals [52]. Kalray's MPPA architecture [53] uses a modified 2D torus NoC topology (MPPA-NoC) and is specifically designed to establish low hop data transfers between processing cores and the
I/O terminals located along the chip edges. Like any folded torus topology, the MPPA-NoC uses physically long metal wires spanning between the opposite edges of a chip in order to reduce the average inter-router hop counts.

The drawbacks of the above architectures can be listed as follows: i) Despite the significant performance gains achieved, the long-range metal links are usually energy and latency expensive even when incorporated with highly optimized buffers, located at regular intervals. ii) The above-mentioned architectures lack any support for performing efficient collective communication.

2.2.3 Emerging NoCs Incorporating Low-Latency High Bandwidth Communication Links

Emerging on-chip communication methodologies employ photonic or surface-wave or wireless links to enable high bandwidth low latency data-transfers for long-range traffic. Among the three technologies, photonic links involve significant power and area overheads [26]. Moreover, the silicon photonic technology is not fully mature and is yet to be compatible with conventional CMOS (Complementary Metal–Oxide–Semiconductor) and FET (Field-Effect Transistor) fabrications. A surface-wave NoC and a graphene-based wireless NoC are proposed recently to perform single hop collective communication and distribute the long-range traffic in one or few hop(s) [54][55]. The drawbacks of these NoCs include: i) surface-wave and graphene interconnects are not fully CMOS compatible, ii) addition of wireless/surface-wave transceiver at each router can lead to high area overheads and iii) use of a shared-medium broadcast communication will result in high contention as the NoC size and/or traffic intensity increases.
A comprehensive survey regarding various wireless NoC (WiNoC) architectures is presented in [20], shows the possibility of creating novel architectures by inserting on-chip wireless links. On chip wireless links operating in the mm-wave (millimeter-wave) wavelength band is demonstrated in [56][57]. The mm-wave wireless interface is completely CMOS compatible and no new technology is needed for its implementation. Robust, nature inspired, small-world network-based wireless NoC architectures employing mm-wave links are presented in [58] and [59]. These architectures outperform the traditional mesh NoC architecture in terms of network latency and energy dissipation [58][59]. However, these WiNoCs are neither aware of collective-communication transmissions nor enable a congestion aware routing to handle high injection rates.

2.3 Collective Communication-Aware NoCs

Conventionally, NoCs are designed for unicast traffic. Traditional NoCs treat the collective communication as repeated unicast which easily leads to: i) local congestion on the source node, ii) poorly guaranteed QoS (Quality of Service) (iii) global congestion due to competition for the same network resource among repeated unicast packets, and iv) power consumption overhead due to repeated injections of the same packets [60]. To overcome these limitations the following solutions are proposed in the literature.

2.3.1 Path Multicast Mechanism

To eliminate these queuing delays and improve the latency in delivering multicast messages in mesh NoCs, path-multicast methodologies have been proposed [61][62]. In a path multicast methodology, for each multicast message, the overall mesh network is divided into
multiple destination regions. The number of such regions and the size of the each destination region is determined by the source node of the message and the variation of the adopted path multicast algorithm [61]. For each destination region, one copy of the original multicast message is created and this copy is routed across all the destinations of the region in ascending/descending order of the destination node addresses. However, the path multicast methodology can lead to large destination regions that require high numbers of hops to distribute the multicast messages [18]. This ultimately leads to high network latency and poor system performance. A dynamic path multicast mechanism where the network is recursively divided into multiple small destination regions is proposed in [63]. However, for all path-multicast mechanisms, at each destination along a path, the header flit is needed to be repackaged with updated list of destinations. Hence, path multicast mechanism requires complex router architectures.

2.3.2 Tree Multicast Distribution and Acknowledgment Aggregation

The XY-Tree multicast mechanism for mesh NoCs has been proposed in [64]. Under this methodology, the multicast message is first forwarded from the source node to all the intermediate nodes lying in the same row. The message is then replicated at these intermediate nodes and a copy of the original message is forwarded to all the destinations lying along the same columns [64]. Unlike path-based multicast mechanism, XY-Tree multicast does not require a complex router and involves simple message forwarding.

For cache coherence induced traffic patterns, each multicast message is associated with a set of acknowledgement (ACK) messages that are transmitted from each multicast destination back to the source node [14][65]. To meet the requirements of the cache coherence
communication, an XY-Tree multicast NoC incorporating an acknowledgement aggregation network is proposed in [14]. A mesh NoC performing broadcasts over uncongested trees and incorporated with ACK aggregation and single-cycle-multiport replication of broadcast flits was presented in [65]. These works demonstrate that with an efficient replication mechanism and with the use of acknowledgement gathering networks, the performance of the manycore systems incorporating Hammer cache coherence protocol can be greatly enhanced.

However, in these systems, the underlying NoC is still a mesh network. In a mesh NoC, the network latency is usually high due to the inherent multi-hop nature of the system [66]. High network latencies cause undesired delays in forwarding the multicast messages as well as in collecting the acknowledgements, leading to stalled processor cycles. Also, higher injection rates can lead to heavy congestion in a mesh network where the XY-Tree multicast follows a single fixed path to transmit data.

### 2.4 NoCs Performing Single-Cycle Multi-hop Traversals

Recently, the SMART (Single-Cycle Multi-hop Asynchronous Repeated Traversal) NoC paradigm is proposed as an efficient solution to minimize both unicast and collective communication latencies in traditional NoCs [47] [66][67][68]. A SMART NoC can establish single cycle connections even between the physically distant routers, dynamically depending on the instant data-transfer requirements under both unicast and collective communication scenarios. To accomplish such flexible long-range connections, SMART NoCs use router-bypass setup networks along with wires incorporating clockless repeaters. In a SMART NoC,
from each output port spans a set of dedicated SMART-hop setup request (SSR) links that are multi-drop in nature. The collection of all the SSR links in the NoC forms a bypass-request distribution network. The SSR network is of two types, 1-dimensional (1D) and 2-dimensional (2D) and is used to forward the bypass requests from the source router to all the intermediate routers present with $HPC_{\text{max}}$ mesh hops, one cycle before the actual data flit arrives [66]. Here $HPC_{\text{max}}$ is defined as the number of mesh hops that can be traversed in one clock cycle using the SMART architecture.

Despite its advantages, the control mechanism used in the SMART NoC gives rise to high router overheads mainly due to the flit width of the multi-drop SSR links [68]. In order to lower the connection overheads, a SMART NoC with reduced wires (we refer this architecture as RSMART NoC) is proposed in [68]. In a RSMART NoC, the control information is exchanged using a dedicated SSR-mesh network (lying in parallel to the data transfer mesh network) and a set of pre-SSR control wires. The pre-SSR wires are also multi-drop in nature and deliver a pre-cursor to an upcoming router bypass request. However, the flit widths of these pre-SSR wires are much smaller than that of the SSR wires in SMART and hence RSMART NoCs require lower connection overheads than the SMART NoCs.

Integrating the SMART control mechanism can be undesirable for high performance NoCs as it restricts the achievable clock frequency of the NoC. More specifically, with increasing $HPC_{\text{max}}$ values, the clock frequency of the NoC decreases linearly [66]. In addition, the router overheads in a SMART NoC grows quadratically with increasing $HPC_{\text{max}}$ [68].
2.5 Summary of Related Works and Dissertation Organization

Following the existing literature, it is evident that designing customized computation platforms for both MapReduce and graph analytics based Big Data processing is gaining momentum. Despite this, it is also clear that the conventional multi-hop NoC architectures are inefficient in handling the traffic patterns arise from the executions of these modern applications. Hence, there is a need for a low-latency NoC capable of performing energy-efficient multi-hop data transfers and collective-communication. The SMART NoC paradigm can provide an efficient solution to satisfy the needs expressed here. However, in order to do so, it essential to address the frequency scalability issues associated with the SMART NoCs.

The on-chip wireless interfaces can transmit broadcast data over multiple hops within one cycle and hence offers an exciting alternative to the conventional wireline links. Inspired by this, this dissertation explores WiNoC based solutions to handle the traffic patterns arise from Big Data processing on manycore platforms. Towards this the rest of this dissertation is organized as follows:

i) In Chapter 3, we analyze the various characteristics of the on-chip traffic patterns generated by MapReduce and graph analytics applications when run on manycore platforms. In addition, we also discuss the impact of the cache-coherence protocols on the on-chip traffic.

ii) Chapter 4 explores the designs of Wireless NoC architectures necessary to handle the irregular memory access traffic arise from graph analytic and MapReduce applications. In particular, this chapter discusses the efficiency of on-chip wireless links, incorporation of wireless interfaces in different NoCs, an efficient Medium
Access Control (MAC) protocol for on-chip wireless channels and the routing protocols for the different wireless-enabled NoCs.

iii) Chapter 5 presents the design of a small-world based collective communication-aware wireless NoC. In addition, this chapter also discusses a network-coding based congestion-aware routing protocol and a wireless assisted acknowledgement aggregation mechanism.

iv) In Chapter 6, first, we outline the motivation for designing a wireless-enabled SMART (WiSMART) NoC. Next, we present the design of a WISMART NoC augmented with look-ahead requests and region-based collective communication routing

v) Finally, in Chapter 7, we conclude the dissertation and indicate the future research directions that can further enhance the efficacy and applicability of the proposed NoCs. Specifically, we discuss a software hardware co-design to address memory access complexities, augmentation of surface-wave and 3D links in SMART NoCs and extending the design principles to heterogeneous systems.
For an efficient manycore platform, it is essential to align the NoC architecture with the on-chip traffic patterns exhibited by the Big Data applications. The overall on-chip traffic pattern exhibited by an application is usually a collection of i) inter-thread data exchanges that mainly depend on the nature of the application and ii) data exchanges that arise from on-chip memory accesses that depend on the nature of the application as well as the adopted cache-coherence protocol. In this chapter, in order to explain the nature of the applications considered in this work, first we present an overview on their computation patterns. Next, we present an overview on the impact of cache-coherence protocols on inter-core traffic. Following these overviews, we elaborate on the various characteristics of the traffic patterns generated by the considered applications.

In this work, we consider MapReduce and graph analytic applications to study the nature of the on-chip traffic patterns. Both the paradigms are well known for their applicability in Big Data analytics and as illustrated by the works mentioned in Section 2.1, MapReduce and graph applications can be easily ported to manycore platforms.

### 3.1 Computation Patterns of Big Data Applications

Understanding the computation patterns of an application will help us in better recognizing the various inter-thread data exchanges that arise from the execution of the application.
Towards this goal, next we explain the salient computation phases associated with each of the considered applications.

3.1.1 MapReduce

MapReduce is a simple and efficient programming model that targets data intensive computations [27]. These applications involve two distinct execution phases, Map and Reduce. Each MapReduce application is associated with a set of characteristic parameters, called keys and values. Typically, MapReduce applications operate on a set of data and identify the values that are associated with each key [69]. To elaborate on this, we consider one of the MapReduce applications, Word Count. This application counts the occurrence of each unique word in a text document. In this case, each unique word can be considered as a key. The total count of each unique word is the value that is associated with it.

3.1.1.1 Phoenix++ MapReduce Implementation

Phoenix++ is a p-thread implementation of the MapReduce programming model for manycore systems [28]. The Phoenix++ implementation can be split up into four general execution stages, Split, Map, Reduce and Merge. The execution stages and the data flow paths

![Diagram](image_url)

Figure 3.1 Execution flow of Phoenix++ MapReduce. Here, the input data is first divided into multiple Map tasks. Upon the completion of Map phase, the intermediate data generated is fed to multiple Reduce tasks. Finally, Reduce phase outputs are merged to get the desired results.
associated with the Phoenix++ MapReduce framework are shown in Figure 3.1. During the Split phase, the input data is divided into multiple similarly sized sub-units. The number of available cores and the nature of the application determine the number of data units created. The Phoenix++ scheduler then assigns each data unit to one of the cores that performs the Map operation. During the Map phase, each core processes all assigned data units and generates intermediate (key, value) pairs [69]. During the Reduce and Merge execution phases, all values associated with each unique key are processed and combined to generate the final output [69].

For most of the Phoenix++ applications, the execution of the Map phase is significantly longer than the Reduce and Merge phases combined. This is due to the fact that the Map phase involves intensive computation on large amounts of data that are transferred from main memory to the processing cores. Compared to the Map phase, the Reduce and Merge phases are less computationally intensive. However, the intensity of the inter-core communication in the Reduce and Merge phases is high due to the transfers of intermediate Keys and Values.

<table>
<thead>
<tr>
<th>Application</th>
<th>Input dataset size</th>
<th>Purpose of the Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication</td>
<td>999 x 999 Matrix</td>
<td>Multiplication of large matrices</td>
</tr>
<tr>
<td>Kmeans</td>
<td>Vectors, dimension: 512</td>
<td>K-means clustering of given data points</td>
</tr>
<tr>
<td>PCA</td>
<td>960 x 960 Matrix</td>
<td>Principal Component Analysis in a matrix</td>
</tr>
<tr>
<td>Histogram</td>
<td>Medium (399 MB)</td>
<td>Histogram of color components in an image</td>
</tr>
<tr>
<td>Word Count</td>
<td>Large (100 MB)</td>
<td>Counting the number of words in a text file</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Medium (100 MB)</td>
<td>Modeling the relationship between variables</td>
</tr>
</tbody>
</table>
In this work, we use six Phoenix++ applications, namely Histogram (HIST), Kmeans (KM), Linear Regression (LR), Matrix Multiplication (MM), PCA, Word Count (WC). Among the Phoenix++ applications analyzed, Kmeans and PCA have two MapReduce iterations, whereas all other applications have a single MapReduce iteration. Further information on the applications analyzed is provided in Table 3.1.

3.1.2 Graph Analytics

Graph theory and graph algorithms represent a vast field with more than a century of actively researched problem and topics. While it is not possible to cover the entire graph domain, we note that the two problems we have selected for detailed study and characterization (community detection and balanced coloring) have been carefully chosen, not only because they encapsulate many of the key graph algorithmic traits (irregular computation, data movement, locality issues, etc.), but also because they embody two different paradigms that are prevalent in many of the advanced parallel graph processing frameworks of the modern day. The first paradigm is that of applications that perform vertex-centric calculations followed by synchronized updates. The second paradigm is to apply synchronization during the vertex-centric calculation phase. These find applicability in a number of graph operations such as page-rank, graph matching, clustering, etc.

3.1.2.1 Community Detection using Grappolo

The community detection problem [70] is one of partitioning the given set of vertices in $V$ into “communities” such that the modularity of the partitioning is maximized [71]. Modularity is a measure, between 0 and 1, that reflects the quality of partitioning. More specifically, it is the ratio of between the net weights of the intra-community edges to inter-
community edges. Neither the number of communities nor the size distribution of communities is known a priori. In fact, community detection is used to reveal such natural divisions that exist in real world networks. It is used in a number of scientific applications including (but not limited to) social network analysis, bioinformatics, collaboration networks, and electric power grid [70]. Despite its broad application base, executing community detection over large real world graphs remains to be a challenging problem despite recent developments in multicore processing. The heuristic nature of algorithms alongside the need to access neighborhoods of vertices in an irregular fashion causes irregular data access and movement patterns that impede performance and scalability in traditional multicore environments.

Recently, Lu et al. [72] developed a scalable parallel implementation for executing community detection on conventional multicore architectures. This method, called Grappolo,

<table>
<thead>
<tr>
<th>Algorithm 1: Community Detection $G(V,E,\omega)$</th>
<th>Algorithm 2: Balanced Coloring $G(V,E),\text{Colors}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Repeat</strong></td>
<td><strong>For each iteration</strong></td>
</tr>
<tr>
<td>Init: $C(v) \leftarrow {v}, \forall v \in V$</td>
<td><strong>For each iteration</strong></td>
</tr>
<tr>
<td><strong>Repeat</strong></td>
<td></td>
</tr>
<tr>
<td><strong>For each</strong> $v \in V$ do in parallel</td>
<td></td>
</tr>
<tr>
<td>// <strong>Vertex-centric Computation</strong></td>
<td></td>
</tr>
<tr>
<td>$C_t(v) \leftarrow$ target community that</td>
<td></td>
</tr>
<tr>
<td>maximizes modularity gain for $v$</td>
<td></td>
</tr>
<tr>
<td>// <strong>Synchronized Updates</strong></td>
<td></td>
</tr>
<tr>
<td>$C(v) \leftarrow C_t(v), \forall v \in V$</td>
<td></td>
</tr>
<tr>
<td>$C.info \leftarrow$ Update community sizes</td>
<td></td>
</tr>
<tr>
<td><strong>until</strong> (Net Modularity Gain &lt; $\tau_1$)</td>
<td></td>
</tr>
<tr>
<td>$G'(V', E', \omega') \leftarrow$ compact $G(V,E,\omega)$ using $C$</td>
<td></td>
</tr>
<tr>
<td><strong>until</strong> (Net Modularity Gain &lt; $\tau_2$)</td>
<td></td>
</tr>
<tr>
<td><strong>Output final communities</strong></td>
<td></td>
</tr>
</tbody>
</table>

**a) Community Detection Algorithm**

**b) Balanced Coloring Algorithm**

**Figure 3.2 Pseudocodes for Graph Algorithms.**
implements a multiple-phase, multiple-iteration heuristic to maximize the modularity of the output partitioning. Figure 3.2(a) presents the pseudocode for Grappolo. In what follows, we outline the algorithmic details and data structures of Grappolo.

Given an input graph \( G(V,E,\omega) \), containing \( n \) vertices and \( m \) edges, the algorithm executes multiple phases, and multiple iterations within each phase, until convergence. Within each phase the following steps are carried out:

i) Initially, each vertex is assigned to an individual community of its own.

ii) Within each iteration the vertices are scanned in parallel and for each vertex, a decision is made to determine whether or not to migrate it to one of its neighboring communities (as defined by the communities of its neighbors). To avoid locking, the parallel implementation uses community assignments as of the previous iteration. Other graph heuristics are used to resolve potential conflicts and in effect to ensure that the net modularity gain achieved by the end of the iteration always remains non-negative.

iii) The algorithm proceeds to subsequent iterations until the gain achieved in net modularity becomes negligible. Reaching convergence by this criterion marks the end of current phase and the algorithm constructs a compacted graph \( G'(V',E',\omega') \) by collapsing every community detected in \( G \) to a single meta-vertex in \( G' \), and creating edges reflecting the weights of intra- and inter-community links in \( G \).

iv) Subsequently, the algorithm initiates the next phase on the newly compacted graph \( G' \), until no more appreciable modularity gain is achieved.
The main data structures used in Grappolo are as follows: The graph is stored as two arrays of size $O(m+n)$ in the Compressed Sparse Row (CSR) format [73]. Each vertex entry also stores the current community assignment for that vertex. In addition, a separate array data structure is used to keep track of the degree of each community (i.e., sum of the degree of all vertices belonging to each community). The orders in which the vertices and communities are stored in their respective arrays are arbitrary as it is not possible to predetermine locality properties due to the dynamic nature of the algorithm. All these factors contribute to highly irregular memory access patterns and in turn complex on-chip communication requirements.

### 3.1.2.2 Balanced Coloring

The classical problem of graph coloring can be stated as follows: Given an input graph $G(V,E)$, assign colors to vertices such no two vertices that share an edge between them are assigned the same color [74]. Coloring is a classical graph operation that is widely used in a number of graph-based scientific applications to determine compatible parallel schedules [75]. As edges in graphs typically represent vertex-to-vertex interdependencies, coloring can be used to obtain a parallel schedule that guarantees no two vertices that are interdependent on one another are processed during the same parallel step (i.e., same “color”). However, such an approach needs to also have as few parallel steps (or color classes) as possible, and therefore a second goal for graph coloring is to minimize the number of colors used in assignment. In addition, since concurrency is limited by the number of vertices within each color class, there is also a need to ensure load balanced color distribution and this variant has been extensively studied under the context of equitable and balanced coloring [76]. Recently, coloring
implementations have been proposed to obtain an initial coloring so as to minimize the number of colors and then redistribute the vertices among color classes so as to obtain a balanced coloring [77][78]. The assignment of colors to vertices and their redistributions make the graph coloring operation data movement-bound and lock-intensive in traditional multicore environments. In addition, these characteristics heavily depend on the underlying graph structure and connectivity.

In a recent work, Lu et al. [79] proposed a number of heuristics to generate a balanced coloring – i.e., the number of vertices assigned per color class (aka. color bin) is roughly the same. They implemented a parallel approach whereby the Greedy First Fit approach is used to first obtain an initial coloring of the graph, and a subsequent balancing step redistributes the vertices among the different bins so as to obtain a balanced coloring without increasing the overall number of colors used. Different balancing schemes were explored and one of the more effective schemes, namely Greedy-CLU, works as follows (please refer Figure 3.2(b) for a pseudocode). Color classes are processed one at a time, such that the vertices within every over-full color bin (i.e., those containing more than the mean number of vertices) are redistributed to one or more under-full bins until the number of vertices within the source bin reaches the target mean or no more valid vertex moves are possible. The choice of the under-full bin is performed greedily by selecting a smallest available bin that is also compatible with the vertex being moved (i.e., contains no neighbors of that vertex).

This implementation also uses the CSR format for storing the input graph, while a separate array-based data structure is used to keep track of the color bins and their sizes. The parallel scalability of the overall coloring process is affected by the need to lock the coloring data
structures in order to update the source and destination bin sizes. In addition, the need to perform compatibility checks for vertices requires access to color bins corresponding to the neighbors of every vertex. This step introduces irregular memory access and workloads.

Thus the balanced coloring application involves two distinct phases, an *initial coloring phase* and a *redistribution phase*, between which the redistribution phase is a communication-bound phase with heavy vertex migration traffic and locking. Furthermore, the distribution phase is devoid of any complex computations. Hence, unlike the initial coloring phase, the execution speed of the redistribution phase is highly dependent on the speed of the data migration.

Table 3.2 DIMACS10 graphs used for *Grappolo* analyses.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Graph Size</th>
<th>Number of Vertices</th>
<th>Number of Edges</th>
<th>Number of Communities</th>
<th>Execution Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASJ</td>
<td>0.46 MB</td>
<td>22,963</td>
<td>48,436</td>
<td>31</td>
<td>1.588099</td>
</tr>
<tr>
<td>PGP</td>
<td>0.25 MB</td>
<td>10,680</td>
<td>24,316</td>
<td>99</td>
<td>0.284018</td>
</tr>
<tr>
<td>ASTRO</td>
<td>1.20 MB</td>
<td>16,706</td>
<td>121,251</td>
<td>412</td>
<td>1.724108</td>
</tr>
<tr>
<td>HEP</td>
<td>0.15 MB</td>
<td>8,361</td>
<td>15,751</td>
<td>636</td>
<td>0.292018</td>
</tr>
<tr>
<td>COND</td>
<td>1.30 MB</td>
<td>31,163</td>
<td>120,029</td>
<td>945</td>
<td>2.035129</td>
</tr>
</tbody>
</table>

Table 3.3 DIMACS10 graphs used for Balanced Coloring analyses.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Graph Size</th>
<th>Number of Vertices</th>
<th>Number of Edges</th>
<th>Number of Output Colors</th>
<th>Execution Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTH</td>
<td>34.90 MB</td>
<td>2,216,688</td>
<td>2,441,238</td>
<td>5</td>
<td>1.628099</td>
</tr>
<tr>
<td>BLG</td>
<td>21.30 MB</td>
<td>1,441,295</td>
<td>1,549,970</td>
<td>5</td>
<td>1.244078</td>
</tr>
<tr>
<td>CCR</td>
<td>14.60 MB</td>
<td>268,495</td>
<td>1,156,647</td>
<td>17</td>
<td>0.332021</td>
</tr>
<tr>
<td>CNR</td>
<td>35.10 MB</td>
<td>325,557</td>
<td>2,738,969</td>
<td>86</td>
<td>0.372023</td>
</tr>
<tr>
<td>CAC</td>
<td>09.80 MB</td>
<td>227,320</td>
<td>814,134</td>
<td>87</td>
<td>0.436028</td>
</tr>
<tr>
<td>CAD</td>
<td>12.10 MB</td>
<td>299,067</td>
<td>977,676</td>
<td>115</td>
<td>0.290009</td>
</tr>
</tbody>
</table>
### 3.1.2.3 Test Inputs for *Grappolo* and Balanced Coloring

The evaluation of *Grappolo* is performed with the help of DIMACS10 clustering instance graph data sets [80]. We have considered five DIMACS10 clustering instance graphs, Hep-th (HEP), Astro-ph (ASTRO), Cond-mat-2003 (COND), PGPgiantcompo (PGP) and as-22july06 (ASJ), to evaluate the benefits achieved by the WiNoC interconnect in execution of *Grappolo* applications. The evaluation of the balanced coloring application is carried out with the help of six DIMACS10 graphs. Two street network graphs, Belgium (BLG) and Netherlands (NTH), three citation network graphs, citationCiteseer (CCR), coAuthorsDBLP (CAD), coAuthorsCiteseer (CAC) and one clustering instance graph cnr-2000 (CNR) are used for these evaluations. Due to the shorter runtimes for the balanced coloring application, we were able to test much larger data sets. Further information regarding the datasets such as, size, vertex and edge counts, number of communities and colors and application execution times are provided in Table 3.2 and Table 3.3.

### 3.2 Impact of Cache-Coherence Protocols on the On-Chip Traffic

In a chip multiprocessor (CMP) platform, cache coherence protocols are used to ensure consistency among the multiple cached copies of shared data. Over the years, many cache coherence protocols such as Directory, Hammer, Token etc., have been proposed for CMPs [81]. In general, selecting an appropriate cache coherence protocol for a manycore platform involves analyzing the area and traffic trade-offs associated with the protocol [81]. Protocols requiring low area overheads usually give rise to high volumes of collective communication while the protocols that involve low traffic intensities are expensive to implement. As explained in Chapter 1, for high performance Big Data analytics, large-scale manycore
platforms are preferred. Hence, for such systems, a low-overhead coherence protocol like AMD’s Hammer based Hyper Transport coherence protocol is desired to achieve high system scalability. However, as we will explain later, Hyper Transport coherence protocol necessitates the use of specialized NoC architectures.

In this work, to cover all ends of spectrum (low to high volume of collective communication and low to high area overhead), we have selected Hammer and Directory protocols for a detailed analysis. Specifically, we study the traffic patterns of the considered applications in presence of two highly contrasting cache-coherence protocols, the two-level four-state Directory protocol (henceforth referred to as “Directory”) and the AMD’s Hammer based Hyper Transport coherence protocol (henceforth referred to as “Hammer”) [19]. The two-levels of the Directory protocols refer to the two level cache implementation (L1 and L2) while the four directory states involved are, Modified, Exclusive, Shared and Invalid (MESI). Among the various protocols discussed in [81], Hammer requires the least area overhead while generating a high collective communication traffic. The directory protocol involves mostly unicasts and has the lowest multicast traffic injection rate while occupying a high area overhead [81].

3.2.1 Directory Protocol

In a Directory protocol based CMP, each translation lookaside buffer (TLB) entry is assigned to one of the cache controllers, called as home node. The home node stores the current state of the data block indicating whether the data block is valid or dirty. The home node also maintains a list of processing cores with the most recent copy of the data block. Whenever a processing core $S$ needs to update a data block or encounters a cache miss, it first
communicates with the home node. For a read miss, if the requested data block is present on the chip, the home node forwards this request to the core $D$ that has the most recent copy of the requested data. Core $D$ responds by forwarding the data to the requested node ($S$). In case of the requested data block being not present on the chip, the data is fetched from the main memory. The home-node assignment, by default, does not follow data locality and hence, the Directory protocol data-transfers can give rise to multi-hop unicast traffic.

On receiving a write miss from node $S$, the home node sends an invalidation signal to all the nodes that are having a copy of the data block. Thus, this invalidation signal generates a multicast traffic. Upon receiving the invalidation signal, the cores respond with an acknowledgement (ACK) signal to the home directory. Finally, the state of the data block is updated in the home node and the node $S$ is permitted to write.

The necessity to maintain a list of sharing cores for each data block causes high memory overheads in systems using Directory protocol. Furthermore, the fraction of this overhead (in total silicon area) increases linearly as the number of cores increases and thus limits the scalability of the system. Hence, for large-scale manycore platforms running Big Data processing, employing Directory protocol is not desirable.

### 3.2.2 Hammer Protocol

In Hammer protocol, similar to the Directory protocol each data block is assigned to a home node. However, unlike the Directory protocol, the home nodes in Hammer protocols do not maintain the list of processing cores that are sharing the most recent copy of the data block. Instead, the home node broadcasts the read/write requests to all the cores in the system.
This gives rise to dense on-chip broadcast bursts. In addition, all the nodes that received the broadcast transmit an ACK signal back to the home node, generating high all-to-one traffic.

The avoidance of full-bit directories (to indicate all sharer nodes) enables the Hammer protocol to have a low logic and memory overhead. Hence, Hammer protocol enhances the scalability of the manycore system, as it does not require large memory structures to keep the list of shared cores typically needed in directory-based protocols. However, manycore systems operating with Hammer protocol require collective communication-aware NoCs to handle the heavy broadcast and ACK traffic.

3.3 On-Chip Communication Traits Exhibited by Big Data Processing

In this section, we analyze the nature of the traffic patterns generated by MapReduce, Grappolo and Balanced Coloring applications when mapped on to the 64-core system. We use GEM5 [82], a full system simulator, to obtain detailed processor and network-level information. We keep track of each message injected through the network interface module of GEM5 to extract the multicast traffic traces associated with real applications [83]. We consider a system of x86 processors running Linux within the GEM5 platform for all the experiments. The application codes for Grappolo and Balanced Coloring are parallelized for multicore platforms using OpenMP multithreading with dynamic scheduling. The tasks are parallelized in term of vertices without involving any optimization based on data locality and hence the considered applications are highly scalable with increasing system sizes [38]. MapReduce applications follow the Phoenix++ p-thread framework for parallelization. In GEM5 simulations, the parallelized tasks are mapped to the emulated parallel threads using
basic Linux scheduler. The memory system of the system employed follows private 64KB L1 instruction and data caches and a shared 16MB (256KB distributed per core) L2 cache.

In what follows, the generic traits of the inter-core communications arise from the considered applications are explained first. Without the loss of generality, we consider the MESI Directory cache coherence protocol for this analysis. Later, to ensure a comprehensive analysis, we compare and contrast Directory protocol induced traffic patterns with Hammer coherence induced on-chip traffic patterns.

### 3.3.1 Long-Range Data Exchanges

Figure 3.3 shows the percentage of total traffic exchanged between the communicating cores separated by certain number of hops in a mesh NoC for both MapReduce and graph analytic applications. From this figure, we can observe that all the applications generate significant amount of long-range traffic, although owing to different reasons.

In case of Grappolo (GP), long-range traffic is largely due to shared memory access during the clustering computation; while in the case of Balanced Coloring (BC), it is due to locks generated during the vertex redistribution phase. In GP, computations associated with a single community of vertices can cause a data migration from any other community of vertices, leading to irregular on-chip memory access patterns and heavy long-range communication. In Balanced coloring application, vertexes can migrate from one color bin to other during redistribution phase. In addition, to ensure data consistency at the software level, locks are employed during this process. This in turn leads to irregular memory access
and multi-hop data transfers. It is evident that for both the applications, 70% of the total injected messages are transferred among cores that are more than three hops apart.

Similar to the graph applications, MapReduce induced data transfers are principally long-range in nature. As explained earlier, MapReduce applications involve Split, Map, Reduce and Merge phases. The inputs data for Map phase are loaded directly from the main memory. Reduce phase mostly operates on the output of the Map Phase, available anywhere on-chip. Hence, Reduce phase involves heavy on-chip memory access and gives rise to the most long-range heavy traffic. Merge phase mainly involves data-recombination. Hence, similar to Reduce, Merge phase leads to heavy on-chip memory access. However, the data-recombination between various threads is usually performed in an ordered fashion and hence, Merge phase is traffic is more regular when compared to Reduce. Among the considered MapReduce applications, LR is the most Map phase dominant (a breakdown on the execution of time of each phase for all MapReduce applications is later provided in

Figure 3.3 Traffic Characteristics: Fraction of total traffic exchanged between the communicating cores separated by x-hops in a 2D mesh NoC running Big Data applications. The different values for x are specified in the legend.
Section 3.5). In addition, LR involves heavy false-cache sharing during Map phase, leading to one-hop data exchanges. On the other hand, PCA has the longest Merge Phase. Due to these reasons, LR and PCA traffic involve notable short and medium range data-exchanges. WC and KM involve significant Reduce phases and hence, exhibit mainly long-range data exchanges.

As shown later in Chapter 4, it is more latency and energy efficient to use wireless links than metal wires for link length for multi-hop communications exceeding three hops. Hence, for the long-range communication exhibited by graph analytic and MapReduce applications, wireless links are highly beneficial.

### 3.3.2 Traffic Hotspots

Figure 3.4 shows the percentage of the traffic injected by each of the top six traffic hotspots present in the system, while executing the considered applications. As seen, many of the selected applications indicate the presence of communication hotspots that are responsible for a significant portion of the total traffic in the system. It can be observed from the plot that up to 10% of the total traffic is associated with a single core!
The top six traffic hotspots in *Grappolo* are responsible for about 25% of the total injected traffic. In Balanced Coloring, the top six traffic hotspots are responsible for up to 27% of the total system traffic. In case of *Grappolo*, this injection originates mostly from the master core, which is responsible for the graph compaction phase. Recall that the compaction step is largely serialized owing to the need to gather vertices belonging to communities. In case of Balanced Coloring, the hotspots appear during the redistribution phase. This fact is attributed to the overheads associated with the management of locks. During the redistribution phase, the vertices processed are themselves located randomly on the network, effectively distributing the traffic injection rates at the sources. However, when multiple vertices located at different cores compete to acquire locks, the lock management introduces hotspots during acquisition and release of those locks.

The presence of traffic hotspots in the MapReduce applications can be linked to the presence of a few master cores that are utilized more than the other cores (hence, they inject/revive more traffic during the execution of the application). To elaborate, in Phoenix++ applications, the core utilization pattern is mostly homogeneous among all cores, except for a few master cores that are more active than the other cores. This effect is more prevalent in applications with long library initialization and merge periods. Library initialization occurs once before each Map phase. Library initialization function is responsible for initial task scheduling and storage space allocation for Keys and Values. The master cores are responsible for invoking library initialization and hence, they tend to be more active than the other cores in the above-mentioned period. Activity in the Merge phase can be understood from Figure 3.1. Merge
phase involves multiple sub stages and the numbers of active threads is reduced with each sub stage. Thus, in Merge phase specific cores are more active than the other cores.

As examples Figure 3.5, shows the distribution of the core utilization values for four Phoenix++ applications, namely KM, HIST, PCA and MM, while running on a 64-core system. All four applications have notable library initialization periods and indicate the presence of master cores whose utilization values are higher than other cores. The master cores in Figure 3.4 are also responsible for the traffic hotspots shown in Figure 3.5. For PCA and MM, the core utilization pattern is relatively more homogenous than KM and HIST. Hence, traffic

Figure 3.5 Core utilization values for MapReduce applications on a 64-core platform. The dotted arrow indicates the average utilization. Each bar represents the utilization ([0,1]) of cores 1-64 in the order of highest to lowest utilization.
injection patterns are more skewed in KM and HIST than in PCA and MM. Particularly, in the case of KM, the utilization and traffic injection values vary largely over the 64 cores. KM involves multiple MapReduce iterations (two MapReduce iterations for the data set we considered, with each iteration having all the execution stages listed in Figure 3.1) and performs a partitioning operation on the given set of data (usually partitions a large set of numbers into multiple groups). By the nature of the KM application, fewer cores are expected to be more active in the second MapReduce stage as the data partitioned in various groups start to achieve convergence. The non-homogenous core utilization pattern of KM also leads to the imbalanced traffic injections among the processing cores, giving rise to traffic hotspots.

To efficiently handle the traffic hotspots exhibited by graph analytic and MapReduce applications, it is essential to design NoC architectures that can redistribute the messages using adaptive routing methodologies.

### 3.3.3 Traffic Intensity Maps and the Applicability of the Thread Mapping

To visually showcase the irregular nature of on-chip traffic patterns arise from the considered applications, we employ traffic intensity maps. To elaborate more, Figure 3.6(a)-(h) indicate the relative intensities of all the elements in 64×64 traffic matrix. The arrangement of source and destination nodes in these matrices can be understood from Figure 3.6(i). As shown, the elements in each row of a traffic matrix represents the intensity of the traffic from a particular source node to all the destination nodes in the system. The short-range traffic patterns are observed near the diagonal of the intensity maps. The long-range traffic is far from the diagonal and the hot-spot patterns can be observed via the intersection of the dark horizontal and vertical stripes.
Among all the applications considered, only LR and PCA exhibit notable pair-wise traffic in their intensity maps. All other applications indicate a well distributed on-chip traffic, outside of the injection hotspots. As explained earlier, LR and PCA applications involve heavy neighbor interactions during map phase and merge phase, respectively. Hence, PCA and LR indicate high intensity spots close to the diagonal. Grappolo, Balanced Coloring, KM, MM and WC computations lead to irregular on-chip memory access patterns and heavy long-range
Figure 3.6 Heat plots (of order 64×64) showing the traffic intensity between different source-destination pairs. To indicate the arrangement of source and destinations in the plots, Figure (i) shows the zoomed version of the top left corner in (h), along with the source and destination IDs. Hot spots appear as dark and red while low intensity pair-wise traffic are identified through lighter shaded squares. Spots close to the diagonal represent the short-range traffic, whereas long-range traffic intensity is observed from the spots close to the edges of the map.
communication. These characters are observed through the distribution of dark squares and the hotspot stripes in Figure 3.6. For Grappolo and Balanced Coloring, the above-mentioned long-range and hot spot traffic patterns are observed consistently over all the input graphs considered here. From Figure 3.7 it is evident that the probability of data-exchange between two given cores does not significantly vary across data sets but only across applications. However, we acknowledge the fact that the traffic injection-rates (i.e., number of messages injected per unit time) do vary highly from data-set to data-set even for the same application due to variations in the durations of graph analytic application phases (as explained later in detail in Section 4.4.5). Hence, we have used multiple data sets with Grappolo and Balanced coloring for our analyses throughout this dissertation.

There are a number of previous works where applications have been thread-mapped on manycore architectures to reduce long-range data exchanges. However, to achieve such a trend, the applications must exhibit high levels of pair-wise inter-router communication. As seen from the traffic intensity maps, the applications under consideration do not exhibit such pair wise traffic patterns and the traffic is well distributed among all nodes. Hence, employing thread mapping is not beneficial for Big Data applications running on manycore platforms.
Figure 3.7 Heat Plots Representing traffic matrices (of order $64 \times 64$) for Grappolo and Balanced Coloring with Different Input Graphs. The elements in a row represent the traffic intensity from a particular source to all the destination nodes in the system and thus the hotspot cores are indicated with darker bars. It can be noted that for a specific application, the hotspots remain the same across both the graphs.
3.4 Analyzing Directory and Hammer Coherence induced On-Chip Traffic

In order to design an efficient collective communication-aware NoC architecture handling various coherence mechanisms, we must study the nature traffic patterns under different coherence protocols. Hence, we discuss the traffic patterns induced by Directory and Hammer protocols. The applicability of this study extend beyond the applications considered in this work and hence we employ a set of ten applications spanning over different fields and exhibiting a wide range of NoC traffic characteristics. More specifically, we use Canneal (CNL) and Fluidanimate (FLD) applications from PARSEC suite [84], FFT, Radix (RAD), LU and

![Figure 3.8 Average flit intensity with Hammer and Directory protocols](image)

![Figure 3.9 Percentage of traffic arise due to multicast injections](image)
Water (WTR) from SPALSH-2 suite [85], Grappolo (GP) and K-Means (KM), PCA and Linear Regression (LR) from Phoenix MapReduce suite.

Figure 3.8 and Figure 3.9 show the average flit intensity and the percentage of traffic that arises due to multicast injections on a traditional unicast mesh NoC respectively. As explained above, when compared to Directory protocol, the Hammer protocol generates higher amount of broadcast and acknowledgement messages. In the unicast mesh NoC, each broadcast flit is converted to multiple unicast flits. Hence, the overall flit intensity (number of flits injected for every instruction executed) for the Hammer protocol is much higher than that of the Directory protocol (up to 10 times higher flit intensity with Hammer). In presence of Hammer protocol, the multicast-injections (converted to multiple unicast packets) are responsible for about 47% of the total traffic. In contrary, with the Directory protocol, the multicast messages contribute up to 8% of total traffic. Due to the higher multicast volumes (hence with higher traffic intensities), the Hammer protocol experiences much higher queuing latency than the Directory protocol. The average queuing latency associated with the Hammer protocol 133.9 cycles while the messages induced by the Directory protocol only experiences about 4.5 cycles of average queuing latency. These results (Figure 3.8, Figure 3.9 and queueing latency) indicate that multicast support is a necessity to efficiently implement Hammer cache coherence protocol.

Figure 3.10 shows that the average number of destinations associated with a multicast message in both Directory and Hammer protocols. The destination count for Hammer protocol broadcasts is 63. For the Directory protocol multicaasts, the average destination count varies from 12 to 22 among the considered applications. Figure 3.11 presents the
distributed nature of multicast destinations for Hammer and Directory protocols. Since Hammer multicasts are mainly broadcasts, almost all the multicast messages need to get transferred to the 4 quadrants (quadrants are shown later in Figure 5.1(a)). Even with Directory protocol, about 65% of the total multicast messages need to be communicated to destinations in four different quadrants. Thus, the cache coherence induced multicasts create heavy long-range communication requirements. These results (Figure 3.10 and Figure 3.11) indicate the necessity for employing a multicast aware NoC that is efficient for long-range communication. In this scenario, wireless channels provide an efficient solution. A wireless channel is a broadcast communication medium capable of establishing one-hop
links even between physically distant nodes. Hence it can simultaneously transfer the data to destinations lying in all four quadrants with low latencies.

Figure 3.12, as examples, we have shown the temporal distribution of multicast injections in FFT and Kmeans applications, running with Hammer protocol. We use a bin width of ten cycles and it can be seen from the plot that there is a high a number of bins with more than one multicast injection in it. The percentage of multicast injections that arrives before or after another multicast injection (within a period of 5 cycles) is given in Figure 3.13. It once again demonstrates that both the cache coherence induced traffic patterns (particularly Hammer induced traffic patterns) create significant amount of simultaneous multicast injections and hence have a high probability to create congestion. Hence, a congestion aware routing scheme will be helpful in distributing the multicast traffic injected with Hammer and

![Multicast Injection Pattern](image)

**Figure 3.12** Multicast injection pattern: Number of multicast flits injected within a bin width of 10 cycles for (a) FFT and (b) Kmeans applications.
Directory coherence protocols. Moreover, with multiple multicasts occurring in a short window, the network coding technique can be effective in distributing multicast packets.

An NoC architecture that simultaneously addresses the needs of both Directory and Hammer protocol would efficiently handle long-range unicasts, broadcast distributions and acknowledgement aggregation, covering all types of coherence incurred on-chip traffic. Hence, in this work, we aim to design a high performance collective communication-aware NoC architecture capable of achieving low latency data transfers over multiple hops. Since this NoC addresses a wide range of on-chip communication patterns arise from cache coherence mechanisms, we posit that such an NoC would be also be efficient under any other coherence protocol.

3.5 Summary

An efficient NoC architecture must address the communication needs arise from i) the nature of the application and ii) the underlying coherence protocols. Running Big Data applications on the manycore platforms give rise to multi-hop data exchanges and traffic hotspots and
hence, necessitates a low-latency NoC architecture capable of adaptive routing. Owing to its ultra-low area and logic overhead, Hammer coherence protocol can be highly attractive for large-scale manycore platforms running Big Data Processing. However, Hammer coherence mechanism generates dense collective communication traffic necessitating a NoC architecture capable of high performance broadcast distribution and acknowledgement aggregation.
4 Wireless NoC Architectures for Big Data Processing

As explained in Chapter 3, Big Data applications exhibit inherent irregularities due to data movements. Due to this, both MapReduce and Graph analytic applications generate substantial long-range traffic patterns when run on a manycore platform (shown in Figure 3.3), with significant amount of data exchanges involve physically far apart cores. Furthermore, these applications show the presence of one or more hotspot nodes whose traffic injection rates are much higher than that of the average traffic injection rate (as later shown in Figure 3.4). Due to these long-range and skewed traffic patterns, we posit that designing an on-chip interconnect infrastructure that enables low-latency data exchange, even among physically distant cores will be critical to achieve performance at scale in these graph applications. To achieve low-latency long-range data transfers, we propose to use on-chip wireless communication links. Towards this goal, in this Chapter, we undertake a detailed performance evaluation to identify the best-suited wireless-enabled NoC architecture for Big Data analytics. Specifically, we evaluate i) mesh, ii) high Radix and iii) small-world NoC architectures.

4.1 Background on On-Chip Wireless Interfaces

In this section, we highlight the limitations associated with the traditional metal wires and elaborate on the implementation and advantages of on-chip wireless interfaces.
4.1.1 Limitations of Conventional Metal Wires

Traditionally, NoC interconnects employ copper metal wires that are not highly suitable to handle long-range data transfers. When employed, long wires exhibit high parasitic resistance and capacitance parameters (RC parameters). Hence, the rise-time (i.e, \( \tau=RC \)) and in turn, the propagation delay associated with the metal wires are also high leading to poor achievable NoC clock frequency [20]. To overcome this issue, metal wires are usually pipelined by incorporating clocked-buffers at regular intervals. This effectively shortens the propagation length traversed in a single clock period, allowing the NoC clock frequency to be high. However, pipelined wires have the following limitations:

i) Pipelined wires still require multiple cycles to accomplish long-range on-chip data transfers.

ii) In an on-chip communication scenario, delaying a communication (due to the latencies associated with pipelined wires) lead to starved computing cores and hence, ultimately high full system energy consumptions.

ii) Employing clocked buffers at regular intervals can both power and area expensive when flit width of the NoC is high.

4.1.2 Design of an On-Chip Wireless Interface

To overcome the limitations associated with metal wires, on-chip wireless links are proposed. The propagation delay of the wireless links is independent of RC parameters. Hence, wireless data transfers occur at the speed of light regardless of the underlying NoC clock frequency. In literature, electromagnetic waves operating in the GHz-THz spectral range.
range have been considered viable for the implementation of the on-chip wireless links [56][57]. For this work, we consider that the on-chip wireless interfaces (WIs) can transmit data over a 20mm range within one cycle and operating in mm-wave and sub-THz frequency bands (Ex. carrier frequencies centered on 30, 60, 90, 140 and 200 GHz frequencies in [57]). The design of the WIs considered in this work is thoroughly explained in [57]. Here, we summarize the salient features of this wireless interface.

The two principal components of a WI are the antenna and the transceiver. The considered WI uses an on-chip antenna that has been demonstrated to provide the best power gain with the smallest area overhead [86]. The transceiver design follows [57][58]. Both at the architecture and circuit levels of the transceiver, low-power design considerations are taken into account. At the architecture level, on-off-keying (OOK) modulation and non-coherent demodulation are chosen to simplify the design and therefore to avoid the power-hungry phase-lock loop (PLL) in the transceiver. At the circuit level, body-enabled design techniques are employed to further decrease power consumption. For a data rate of 30Gbps, the total power consumption of the on-chip wireless transceiver is 18mW. In [57], WI designs for five carrier frequencies were demonstrated exploring the 30-200GHz spectrum. By using current CMOS technology, both the frequency range (hence, the data rate per carrier) and the number of carrier frequencies can be increased further for WiSMART NoCs, depending on the communication requirement [55]. For example, given a wireless NoC operating at 2.75GHz and a flit width of 16-bits, a wireless carrier transferring data at 44Gbps is required. For applications requiring higher flit widths, we will require wireless channels with even

50
higher data rates. Either using a single carrier or aggregating multiple carriers we achieve the necessary data rate depending on the flit width of the application.

For the WI design considered, the transmitter occupies 200µm×460µm and the receiver occupies 270µm×15µm area overhead in 28nm technology [57]. Thus, considering a 2mm×2mm NoC tile, addition of a WI causes an area overhead of 5.78%. Because of the area overheads, the number of WIs that can be deployed in manycore platform is usually limited.

### 4.1.3 Efficiency of the On-Chip Wireless Links

In this section, we demonstrate that the on-chip wireless links are more efficient than the traditional metal wires to perform long-range on-chip data transfers. Figure 4.1 shows the Energy Delay Product (EDP) per bit of both the wireline and wireless links, transferring data over certain communication lengths. The wireline link is incorporated with uniformly placed and sized repeaters (in 28 nm technology node) for performance optimization. The wireless transceiver [57] considered here does not have any inherent power tuning mechanism that

![Figure 4.1 Wireline and wireless EDPs for transferring one-bit over varying link lengths.](image-url)
can control the radiation energy depending on the communication length. Hence, for wireless links, we consider a constant data rate and a constant bit energy that is recorded over 20 mm on-chip communication range satisfying a typical BER requirement of $10^{-15}$. For the considered transceiver design [57], in 28nm CMOS technology, the wireless link dissipates 1.3pJ/bit for transferring data over 20mm ranges at a data rate of 16 Gbps.

It is evident from Figure 4.1 that for link lengths exceeding 6.25mm, the wireless link has a lower EDP than the wireline link. The EDP of the wireline link increases rapidly with growing communication lengths (both the link latency and the energy increases linearly). On the other hand, the wireless links are capable of sustaining high-data rates regardless of the variation in the communication length as long as the transmitter and the receiver are within the range.

4.1.3.1 High Fan-Out of Wireless Links

The wireless communications are broadcast in nature. Thanks to this omni-directional nature, adding five WIs on a single wireless channel actually enables a total of ten long distant communication shortcuts. Because of this mentioned effect, insertion of few WIs can lead to a high reduction in the overall inter router hop-count. This reduction in hop count ultimately leads to low network latency and energy.

4.2 Distributed MAC Protocols for Wireless NoCs

In wireless NoCs, to achieve low latency long-range data transfers we have to ensure that the wireless channel is utilized properly. Utilization of wireless links and in turn, the achievable performance of any wireless NoC infrastructure, both depend on the efficiency of the Media
Access Control (MAC) protocol that arbitrates between the competing wireless nodes. This section explains the constraints associated with designing a MAC protocol for WiNoC and presents the design of a distributed MAC protocol suitable for WiNoC architectures.

4.2.1 Limitations in adopting the popular MAC protocols for Wireless NoCs

The wireless MAC protocols are required to grant wireless channel access to the WIs at any given instant of time. MAC protocols prevent multiple simultaneous transmissions on a single wireless channel in order to avoid collisions. On-chip networks are usually energy, area and memory constrained and hence the use of complex MAC protocols for wireless NoCs is not suitable. This makes the design of a MAC protocol for the wireless NoCs different from that of the traditional wireless computer networks. Traditional MAC protocols such as CSMA with collision avoidance use long back-off time counters to avoid collision. In wireless NoCs, instead of keeping the packets in the buffer and waiting for wireless access, it is more beneficial to reroute the packet through the alternative wired paths. MAC protocols that involve request and grant transmissions generally contain multiple fields of data to ensure fairness and high throughput. Such MAC protocols with large control data overheads are undesirable for wireless NoCs. The energy constraint associated with wireless NoCs makes the MAC protocols that allow packet collisions unattractive as they involve high number of re-transmissions.

The adoption of popular MAC protocols such as token passing, FDMA and CDMA for wireless NoC environments are explained in [58], [87] and [88] respectively. The CDMA scheme is complex in nature. Moreover, CDMA employs signal spreading causing high area and energy overheads. Limitation on number of available on-chip wireless channels makes the FDMA
scheme inefficient for large system sizes. A hybrid MAC scheme combining both TDMA and FDMA is reported in [87]. This scheme requires multiple wireless transceivers in each node and is undesirable due to the high area overheads associated with it. Among the traditional MAC protocols, the token passing protocol is the most suitable MAC protocol for wireless NoCs due to its simple implementation. Token passing scheme avoids collision, is fair in sharing the medium and involves simple implementation. However as explained later, the token passing scheme is limited in scalability.

A synchronous and distributed MAC (SD-MAC) for the Ultra Wide Band (UWB) wireless NoC is proposed in [89]. UWB wireless NoC uses an impulse-based transceiver where the wireless links carry data over a millimeter range. SD-MAC algorithm follows a binary countdown approach to arbitrate the channel access. The control signal required to manage the contention among the wireless nodes is carried through wired links. In NoCs such as WiMesh and WiNoC, the wireless nodes are separated by long distances. Hence, communication between the wireless nodes can take multiple hops when traversed along the alternative wired links. Therefore, unlike the UWB-based NoC, the usage of wired links to resolve the contention among the wireless nodes is not viable for WiMesh and WiNoC.

Considering all the above factors, we design a distributed MAC protocol suitable for wireless NoCs. The proposed MAC uses simple orthogonally coded request packets, inherits the fairness provided by token passing in request processing and avoids the grant transmission overhead by similarly processing the requests at all wireless nodes. We designate the proposed MAC protocol as distributed MAC due to its distributed decision making nature. To
compare and contrast the performance of the proposed MAC protocol, we also provide a comparative performance analysis with respect to the token passing MAC.

4.2.2 Token Passing MAC

When employing the token passing protocol, a WI must acquire the circulating wireless token to transmit data over the wireless channel. Once acquired, the token is held back until all flits in the WI are transmitted. Upon finishing the transmission of all the flits, the token is released, in the following cycle, to the next WI. Under this MAC, when a WI does not possess the wireless token, all of the packets are re-routed from the WI to the destination node using wireline-only paths [58].

4.2.2.1 Token Passing MAC: Handling Failures

Token passing scheme requires successful forwarding of token among the WIs. The loss of a transmitted token can potentially stall the wireless data transmissions. Moreover, token forwarding happens among the WIs in a specific circular order. WI failures affect this token forwarding, leading to stalled wireless transmissions.

We can identify the stalling of wireless transmissions by constantly monitoring the busy status of the wireless channel and the token rotation period. To recover from these stalled wireless transmissions, the following recovery steps are used.

i) Once a stalling is identified, the token forwarding process is reset and the first node in the circular order restarts the token transmission.

ii) In case of WI failures, resetting the token forwarding is not sufficient to avoid wireless transmission stalling. Once a WI failure is identified, the token passing sequence
should be re-ordered to avoid wireless transmission stalling. While re-ordering, the failed WI nodes are left out of the token passing sequence.

4.2.2.2 Limitations of Token Passing MAC

Under token passing MAC, the token returning period increases with the increase in the number of wireless nodes assigned to a particular wireless channel. When the token is unavailable, either the packets are rerouted through costly wired links or forced to wait until the return of the token. Hence, the token passing scheme only desired when a low number of WIs are employed for each wireless channel. This makes the token passing scheme less ideal for larger system sizes performing large-scale Big Data processing.

4.2.3 Distributed MAC

In the wireless NoCs, since the WIs broadcast the flits, every node inherently possesses the channel status (whether the channel is busy from an ongoing transmission or not). By using this information and a set of simple orthogonal codes, the contention among the wireless nodes can be resolved.

The flow chart explaining the operation of the Distributed MAC protocol is given in Figure 4.2. For the BDM scheme, when there is no message to be transmitted, the wireless channels are used for exchanging request packets. During request transmission times, each cycle period is divided into multiple bit slots. For a set of $N$ nodes sharing the same wireless channel, $N$ bit slots are needed for creating a request packet. Every bit in the request packet represents a particular node. For example, for a set of 4 WIs sharing the same wireless channel, in the request packet of 4 bits ($b_3 b_2 b_1 b_0$), bit $b_0$ represents node 0, $b_1$ represents
node 1 and so on. By transmitting a bit value of 1 at the specific bit slot, a WI broadcasts a request. To clarify, while transmitting a request, node 0 broadcasts 0001 while node 1 transmits 0010 and so on. If a node transmits 0000, then there is no request sent from that node. All the request codes follow orthogonal on-off keying [90] and hence can be transmitted at the same time. Therefore each node receives all the request packets, simultaneously. Once the requests are received, every node processes the requests and one of the WIs acquires the wireless channel and starts transmitting.

Using Figure 4.3 as an example, we explains the process of selecting the node that acquires the wireless channel. Here, the channel is shared by 4 nodes (N1 to N4). The decision making depends on two factors, the last selected node and the list of nodes that are requesting the
Among the competing WIs, the node that lies closest to the previously selected node (closest on the directed graph, in Figure 4.4) acquires the channel and starts transmitting data over the wireless channel. Figure 4.3 represents a few example request and node selection scenarios. During time $T_1$, only node 1 has sent a request packet and hence the wireless channel is acquired by node 1. During time $T_2$, two nodes $N_2$ and $N_3$ have sent request packets. From the last selected node $N_1$, among the two requesting nodes $N_2$ and $N_3$, $N_2$ lies closest to the node $N_1$ on the directed graph. Hence at time $T_2$, node $N_2$ is selected as the node that acquires the wireless channel. If a wireless interface transmits a request and fails to acquire the channel, then the corresponding data flits are forwarded through alternate wireline paths in the following cycles.

### 4.2.3.1 Distributed MAC: Handling Failures

The distributed MAC scheme is inherently protected from WI failures. This is due to the fact that a failed WI would never request for wireless channel access. In case of distributed MAC scheme, erroneous request packets may lead to erroneous channel accesses and collisions. This can be explained with the following scenario: Consider two nodes, node 1 and node 2 require wireless channel access and both have simultaneously transmitted the request
packet. However, due to wireless channel noise, the request packet of node 1 is lost at node 2. With such erroneous request packet, the request processing would result in conflicting results for both the nodes. If node 2 is the last transmitted node (refer Figure 4.3), both nodes 1 and 2 would try to access the wireless channel after request processing, leading to collision.

To handle failures associated with the Distributed MAC, once a collision is identified, data packet transmission is aborted and the request packets are re-transmitted.

4.3 NoC Architectures for Big Data Processing

In this work, we undertake a detailed performance evaluation of three interconnections architectures, viz., mesh NoC, small-world NoC and high radix NoC to run Big Data applications. For each of these architectures, we consider two implementations, i) a conventional fully wireline architecture and ii) a wireless enabled high performance architecture. First, we present a brief tutorial regarding the salient features of these NoC architectures and then we will present the experimental results comparing and contrasting performance of these architectures for Big Data processing.

4.3.1 Mesh NoC

As stated earlier in Chapter 1, mesh NoC architecture is today's industry standard due to various advantages like simple design, regular structure, easy timing closure etc. However, standard Mesh NoCs (example in Figure 4.5(a)), are not capable of handling long-range traffic efficiently, as they involve multi-hop transmissions. Hence, we consider the mesh architecture with long-range shortcuts here.
4.3.1.1 Wireless Mesh

In [48], a mechanism to enhance the performance of the wireline mesh architecture by employing long-range shortcuts is demonstrated. Following [48], in this work, we augment the wireline mesh architecture with long-range wireless shortcuts and this NoC architecture is called as Wireless Mesh (WiMesh) NoC. The architecture is shown in Figure 4.5(b). The algorithm used for the addition of WIs in the WiMesh NoC is later elaborated in Section 4.3.4.

The mesh NoC incorporating long-range wireline shortcuts presented in [48], follows a distributed routing methodology, referred to as South-Last Routing. In this mesh NoC incorporating long range wireline shortcuts, all the wireline ports (including those used for long range connections) are assigned with particular directions such as north, east, south-east, north-west etc., In South-Last Routing method, the routers without any long-range link follow the traditional XY routing methodology whereas the routers with long-range links follow a turn restricted routing methodology. In this turn restricted routing methodology, all the deadlock inducing turns such as 180 degree turns are prohibited. Comparing the directions assigned to the router ports identifies these restricted turns. However, in WiMesh, it is not possible to assign a particular direction to the Wireless Interface ports. Hence, employing the turn restricted routing methodology for routers with WIs is not feasible. In this work, we follow a modified version of South-Last routing for the routers with WIs. In this modified version of South-Last routing, whenever wireless traversal is involved we prevent deadlocks by allowing the message paths to follow a strictly increasing (or decreasing) router IDs. It is already shown in [91] that a routing methodology where the
Figure 4.5 Various NoC Architectures illustrating data transfers: Figures (a-d) showcase 36-tile NoCs, each with two data transfers (R0→R29 & R30→R14). Figures (e) and (f) showcase FBF topologies with 16 routers and one data transfer (R0→R14). In wireless enabled NoCs, the color of the router indicates the wireless channel frequency that is associated with the router.
message path follows a strictly increasing (or decreasing) router IDs is deadlock free. In case of wireless node failures, we resort to standard XY-wireline routing.

### 4.3.2 Small-World NoC

In the SWNoC topology, each core is connected to a router; routers are interconnected using wireline and wireless links. The overall topology is a small-world network where the links between routers are established following a power law distribution [58]. More precisely, the probability $P(i,j)$ of establishing a link between two routers $i$ and $j$, separated by a Euclidean distance $l_{ij}$, is proportional to the distance $l_{ij}$ raised to a finite power as in:

$$P(i,j) = \frac{l_{ij}^{-\alpha} f_{ij}}{\sum_{i} \sum_{j} l_{ij}^{-\alpha} f_{ij}} \quad (4.1)$$

The frequency of traffic interactions between cores, $f_{ij}$, is also factored in, so the more frequently communicating cores have a higher probability of having a direct link inserted between them. This frequency is the percentage of traffic generated by core $i$ that is sent to core $j$ directly. This approach implicitly optimizes the network architecture for a non-uniform traffic scenario.

Getting into details, the parameter $\alpha$ governs the nature of connectivity, e.g., a larger $\alpha$ means a locally connected network with a few, or even no long-range links. By the same token, a zero value of $\alpha$ generates an ideal small-world network following the Watts-Strogatz model [49]– one with long-range shortcuts that are virtually independent of the distance between the cores. It has been shown that a value of $\alpha$ less than $D + 1$, $D$ being the dimension of the network, ensures the small-world property; with $\alpha=1.8$, the average hop count is minimum with a fixed wiring cost $\text{re}$. Figure 4.5(c) shows an example Small World Network on Chip.
(SWNoC) architecture designed following Equation (4.1) and is constituted only of metal wire links. As it is evident from this figure, the SWNoC uses several long metal wires.

### 4.3.2.1 Wireless enabled Small-World NoC

Since the long metal wires are costly both in terms of power and latency, we propose to use wireless links to connect the routers that are far apart in our small world based wireless enabled NoC (WiNoC) architecture (shown in Figure 4.5(d)). In practice, depending upon the available wireless resources, we can only allow a limited number of long links in the WiNoC to be wireless, while the others would still remain wireline. This way, we can make the distant cores “socialize” with each other, and hence reduce the communication costs when running real applications. Using wireless channels, we overlay the wireline small-world connectivity with the wireless links such that a few routers get an additional wireless port. The wireless port of each router is provided with Wireless Interface (WI) tuned to one of the three distinct frequency channels. The WI placement mechanism that is used to overlay the power law based wireline connectivity with wireless nodes is explained later in Section 4.3.4.

### 4.3.2.2 ALASH Routing Protocol

The power-law model based SWNoC and WiNoC principally have irregular network topologies. Irregular networks require topology agnostic routing methods. Hence, we use ALSAH protocol [58] for routing packets in WiNoC and SWNoC. ALASH is built upon the layered shortest path (LASH) algorithm [92]. The LASH algorithm takes advantage of the multiple virtual channels in each port of the NoC routers in order to route messages along the shortest physical paths. In order to achieve a deadlock-free operation, the network is
divided into a set of virtual layers, which are created by dedicating the virtual channels from each router port into these layers. The shortest physical path between each source-destination pair is then assigned to a layer such that the layer's channel dependency graph remains free from cycles and this ensures the deadlock freedom. ALASH protocol improves on the LASH routing scheme, by enabling an adaptive layering function by considering the expected traffic patterns. We follow the priority layering function explained in [58]. The priority layering function allocates as many layers as possible to source-destination pairs with high $f_{ij}$. This improves the adaptability of messages with higher $f_{ij}$ by providing them with greater routing flexibility. In ALASH, if the wireless interface located in the path becomes unavailable due to lack of buffer space or due to failures, then the packet is rerouted from the wireless interface along the wireline only paths.

4.3.3 High Radix NoC

High radix NoCs target low-hop count NoCs and integrate multiple processing cores to a single NoC router [44]. Flattened Butter-Fly (FBF) topology is one such low hop count high radix topology. In this work, we consider a 16 router FBF topology presented in [46]. As shown in Figure 4.5(e), each router in FBF is connected to all other routers in its row and to all other routers in its column. Hence, in this FBF, all the NoC routers in the system can communicate with each other within two hops by following a dimension ordered XY routing [46]. As it can be noted in Figure 4.5(e), the FBF topology incorporates a number of long-range wireline connections that can be costly in terms of delay and energy. In order to efficiently handle the long-range communication, in this work, we incorporate Wireless Interfaces in each of the NoC routers present in FBF. In WiFBF, the overall number of routers
is a small fixed number (only 16) and hence even when each router is provided with a WI, the additional on-chip area overhead is relatively low. This modified network is called as Wireless enabled Flattened Butterfly (WiFBF) and the connectivity is shown in Figure 4.5(f).

4.3.3.1 Routing Protocols for High Radix NoCs

In [46], to route the packets along the wireline FBF network, a minimal XY dimension ordered routing scheme (with a maximum inter router hop count of two) is used. In this work, we expand on this scheme to design the routing methodology for WiFBF. The WiFBF routing protocol followed in a router $R$ with a WI channel frequency of $C_R$ is shown in Figure 4.6. As shown in Figure 4.6, in WiFBF routing, each message is allowed a maximum of one wireless traversal and the wireless routing for any message can be only initiated at the

![Figure 4.6 Flow Chart explaining WiFBF routing](image-url)
source router for that message. Hence there is no opportunity for creating cycles and thus the WiFBF routing is deadlock free. Under WI failures (or when a WI is unavailable), the proposed WiFBF routing simply follows the minimal XY dimension ordered routing along the wireline links and hence the maximum hop count for any communication in WiFBF is only two.

4.3.4 Placement of Wireless Interfaces

As discussed in earlier Section 4.3.3, in the WiFBF architecture, there are only sixteen NoC routers and each router is provided with a WI. However, in WiMesh and WiNoC, based on the allowed area overhead, we can only incorporate a limited number of WIs and hence it is essential to optimize the location of these WIs to ensure performance gain without paying high area overhead. For this purpose, we employ a simulated annealing (SA) based WI incorporation methodology.

The flowchart presented in Figure 4.7 shows a SA-based mechanism used for placing a set of N WIs on each wireless channel in WiMesh and WiNoC. In the WiMesh and WiNoC creation process, the wireless interface placement strategy focuses on minimizing the optimization metric $\mu$, which is defined as:

$$\mu = \Sigma_{i} \Sigma_{j} f_{ij} h_{ij} \quad (4.2)$$

Here $h_{ij}$ denotes the minimum distance in number of hops from router $i$ to router $j$ with the given network connection. The $f_{ij}$ value denotes the frequency of interaction between the routers. The $f_{ij}$ values are determined by analyzing the traffic patterns generated while executing the Grappolo and Balanced Coloring applications with a set of graphs. To optimize
the network for handling multiple applications, a combined $f_{ij}$ metric can be used. Let $L$ denote the number of applications considered. Then, the net interaction frequency is given by:

$$f_{ij} = \frac{1}{L} \sum_{n} \frac{f_{ijk}}{\sum_{\forall i} \sum_{\forall j} f_{ijk}}, \quad k = 1, 2 \ldots L \quad (4.3)$$

where $f_{ijk}$ denotes the frequency of interaction between routers $i$ and $j$ for a specific application $k$. All the $f_{ij}$ elements of an application constitute its traffic matrix. As elaborated earlier in Section 3.3.3, the traffic matrix of an application has a unique signature with constant traffic hotspots that can be observed over all different input graphs. Hence, any set of large real world graphs can be used to identify the $f_{ij}$ patterns. Moreover, as shown earlier in Figure 3.7, these hotspots communicate almost evenly with all the other nodes and do not involve in any high-intensity pairwise communication. Hence, placing Wireless Interfaces (WIs) to minimize the overall traffic weighted hop count would ensure the following:

i) The WIs would be placed nearer to the traffic injection hot spots.

ii) The remaining WIs would be essentially distributed all across the chip such that the overall inter-router hop count is minimized.

It was shown earlier in Figure 4.1 that wireless links are more efficient than traditional metal wires only when the link length exceeds 6.25mm in the 28 nm technology node. Hence, in addition to the simulated annealing mechanism explained in Figure 4.7, we also impose a WI placement constraint where two WIs operating in the same frequency channel are separated.
by more than 6.25mm. Following the mechanism explained in Figure 4.7 and by varying $N$, we can find the optimum number of wireless interfaces for a NoC and their best suitable locations. The experimental results showcasing the optimum value of $N$ for both WiMesh and WiNoC are discussed later in Section 4.4.2.

4.4 Performance Evaluations

In this section, we characterize the network performance of all the NoC architectures considered. In addition, we also evaluate the efficacy of the distributed MAC scheme. For these characterizations, we consider the combined graph analytic inter-core traffic pattern ($f_{ij}$), obtained with Equation (4.3) in Section 4.3.4. Since we are interested in both network
latency and energy dissipation, we use the Energy-Delay Product (EDP) as the relevant performance metric to characterize the NoCs. Average message latency and average message energy values are used in this message EDP computation.

We use GEM5 [82], a full system simulator, to obtain detailed processor and network-level information. We consider a system of 64 x86 processors running Linux within the GEM5 platform for all the experiments. The memory system is a four state (Modified, Exclusive, Shared and Invalid) CMP directory setup with private 64KB L1 instruction and data caches and a shared 16MB (256KB distributed per core) L2 cache. The width of all wireline links is considered to be the same as the flit width, which is considered to be 32 bits in this paper.

For mesh and small world architectures, we employ a generic three stage router architecture explained in [93]. For High Radix architectures, we employ four-stage router architectures as suggested in [46]. For the small world NoCs, we consider an average of four ports per router, which is similar to that of the conventional wireline mesh. Also in small world NoCs, we impose an upper bound on the number of ports attached to a particular router so that no router becomes unrealistically large. The SWNoC and WiNoC achieve highest throughput with lowest energy dissipation when the maximum port count in a router is restricted to seven [58]. As stated earlier, the FBF and WiFBF architectures are having 16 routers and hence each NoC router is connected to 4 computing cores. For all the NoC architectures considered, all the router ports are provided with a buffer depth of two flits. Energy dissipation of the NoC routers, inclusive of the routing and MAC blocks, was obtained from the synthesized netlist by running Synopsys™ Prime Power, while the energy dissipated by wireline links was obtained through HSPICE simulations taking into consideration the length
and layout of wired links. In all the NoC architectures, across both wireline and wireless links, we follow wormhole routing methodology.

### 4.4.1 Efficiency of the Distributed MAC

To analyze the efficiency of the Distributed MAC, we first quantify the number of WIs that it can supports in each wireless without introducing significant MAC overheads. In order to provide a comparative baseline, we also show the optimum number of WIs in each channel under the token passing scheme. Figure 4.8 compares the performance of WiNoC in presence of the two MAC schemes with varying the WI counts across three wireless channels. Here, each channel uses the same number of WIs. In this figure, we also show the improvement achieved in wireless utilization with the Distributed MAC when compared to the token passing MAC. The wireless utilization parameter represents the percentage of total messages that are using the wireless channels. For this analysis, we use the combined $f_{ij}$ traffic metric, given by (4.2).

Under Distributed MAC, the achievable latency improves until the WI count reaches a certain limit (18 in this case, six WIs per channel) and then remains constant. Whereas when we consider the token passing MAC, the latency reduces until the WI count reaches an upper limit (12 here), beyond which it starts to degrade. The Distributed MAC provides sufficient opportunity for all the competing nodes to access the wireless channels and hence the wireless utilization continue to increase with increase in WI count until the number of messages requesting the wireless channel access gets saturated. In case of the token passing MAC, as the number of WIs increase beyond a certain limit, performance starts to degrade
due to the increasing token returning period. The unavailability of the token for long intervals forces the packets to take costly wireline paths, resulting in the latency penalty.

To further explain the difference in the two MAC schemes, Figure 4.8 compares their channel access times. It is evident that as the number of WIs increases, the channel access time for token passing MAC increases more rapidly than the distributed MAC. The token has to pass through multiple nodes before reaching the node that requires wireless channel access. Each token passing involves a token transmission and a processing period,
contributing to higher access time. Distributed MAC avoids this multiple token passing and requires only one request transmission and processing period to acquire the wireless channel.

From this analysis, it can be concluded that the Distributed MAC can sustain a higher number of WIs on the same wireless channel when compared to the token passing scheme and thus provides better scalability needed for wireless NoC architectures running Big Data processing.

4.4.2 Optimum number of Wireless Interfaces Supported

We first quantify the maximum number of WIs that can be added in WiNoC and WiMesh architectures, beyond which the addition of anymore WIs will provide no further significant improvement in performance. Figure 4.10(a) and Figure 4.10(c) show the variation in EDPs of the WiMesh and WiNoC architectures respectively with increasing number of WIs across three wireless channels. Here, each channel uses the same number of WIs. We also show the improvement achieved in wireless utilization (U) with increasing number of WIs. The wireless utilization parameter represents the percentage of total messages that are using the wireless channels. To find the optimum number of WIs, we use a metric δ which is defined below.

\[
\delta(i) = \frac{U(i) - U(i - n_c)}{U(i - n_c)} \times \frac{EDP(i - n_c) - EDP(i)}{EDP(i - n_c)}
\]  

(4.4)

In above equation, \( n_c \) represent the number of wireless channels available and \( i \) denote the number of WIs in the current system. We consider a step value of \( n_c \) in the above difference
equations, since we add $n_c$ wireless node in each step (one more WI per channel per step). The number of wireless interfaces ($i$) for which the $\delta(i)$ approaches zero, can be considered as the optimum WI count.

Figure 4.10(b) and Figure 4.10(d) show the $\delta(i)$ values for WiMesh and WiNoC architectures respectively. From Figure 4.10(a) and Figure 4.10(c), it can be seen that the achievable EDP and wireless utilization improves until the WI count reaches a certain limit (18 in this case for WiNoC and 15 for WiMesh) and then remains constant. WiNoC has a better wireline
connectivity to distribute the long-range traffic when compared to WiMesh NoC. Due to the inherent multi-hop characteristics of the baseline mesh network of WiMesh, more messages try to access the wireless paths to improve the latency. Hence, the wireless channels in the WiMesh system saturate with a lower number of WIs. Thus, we can conclude that the optimum number of WIs for WiNoC is 18 and the optimum number of WIs for the WiMesh is 15, beyond which the $\delta(i)$ values are near zero. Hence, from here on, for all the following experiments, we consider a set 15 WIs for WiMesh (5 WIs for each wireless channel) and 18 WIs for WiNoC (6 WIs operating for each wireless channel). As already discussed in Section 4.3.3 and as shown in Figure 4.5(f), WiFBF follows a fixed WI placement and channel assignment with WIs placed on all of 16 routers (Six WIs on channel 1, five WIs on channel 2 and five WIs on channel 3).

4.4.3 Network Performance

In this section, we present the comparative analysis of the network performances of all the NoC architectures considered in this work. As shown in Figure 4.11(a), the WiFBF and FBF architectures have the two lowest average inter-router hop count values. However, each router traversal in the FBF topology is costlier in terms of energy and latency, compared to that in mesh and small-world NoC topologies. This is further corroborated by the single router traversal energy values presented in Figure 4.11(b). Having a much large number of ports in each router and requiring a hierarchal switching stage contribute to the additional router stage latency and energy experienced by FBF and WiFBF architectures.

The average message latency and EDP values are presented in Figure 4.11(c) and Figure 4.11(d) respectively. From these figures, it is evident that all the wireless enabled NoC
architectures perform better than the corresponding wireline only NoC following the same topology. Among all the topologies, mesh topology achieves the best latency and EDP improvement with the addition of wireless interfaces. This fact can be attributed to the high reduction in average hop count achieved by the mesh NoC with the addition of WIs (Figure 4.11(a)). Among all the wireline only NoCs considered, the FBF NoC achieves the lowest message latency. Similarly, among all the wireless-enabled NoCs, WiFBF achieves the best message latency value. However, due to their high router energy consumptions, in term of
EDP, the WiFBF and FBF topologies perform considerably worse than the WiNoC and SWNoC topologies (38% EDP penalty for WiFBF when compared to WiNoC) respectively. Aside from achieving the best network EDP values, the WiNoC exhibits only a 4.7% higher latency values when compared to WiFBF. From these analyses, we can conclude that for efficient implementations of complex large-scale graph analytics, the WiNoC architecture is the best suitable option.

4.4.4 Robustness of the NoC Architectures

Any good wireless NoC architecture should be able to handle the runtime WI failures as seamlessly as possible. As we have already explained in Section 4.3, in all three wireless NoCs (WiMesh, WiNoC and WiFBF), we handle the runtime WI failures by rerouting the packets along the wireline only paths. Figure 4.12 shows the increase in the EDP of the three above-mentioned wireless NoCs with increasing number of failed wireless nodes (WIs). From this figure, it can be easily seen that compared to WiNoC and WiFBF, the EDP of the WiMesh architecture increases more rapidly with the increase in number of failed WIs. This steeper increase can be attributed to the fact that the wireline only paths (used for rerouting) in the

![Figure 4.12 NoC Robustness: Increase in EDP against increase in no., of failed WIs.](image)
WiMesh topology have much higher average hop counts than that for the WiNoC and WiFBF topologies. In WiFBF the maximum hop count is always two (even under WI failures) and the small world topology of WiNoC ensures a low average hop count even among the physically far apart routers. It can be seen from Figure 4.11(a), that in fact small world architecture with no wireless interfaces (SWNoC) has a smaller hop count than that of mesh architecture with 15 WIs (WiMesh). From these discussions, we can conclude that the WiNoC and WiFBF architectures are better in handling the runtime WI failures when compared to the WiMesh architecture.

### 4.4.5 Full System Performance

In this section, we evaluate the overall runtime of all the NoC architectures considered when running the target Big Data applications. For this purpose, we use Grappolo and Balanced Coloring applications to explain the observed trends in detail.

#### 4.4.5.1 Grappolo: Execution Time

Figure 4.13 shows Grappolo's execution times. Among all the NoC architectures considered, WiFBF and WiNoC achieve the lowest execution times due to their low hop count nature. When compared to the traditional wireline mesh, FBF achieves an average of 16.2% execution time improvement while WiNoC achieves an average of 14.2% execution time improvement. Among the five graphs considered, HEP achieves the highest execution time improvement by using the WiNoC (17.9%) and WiFBF (20.5%), when compared to the wireline mesh. HEP is followed by ASTRO and PGP (with 19.5% improvement with WiFBF and 17.2% with WiNoC), when compared to the wireline mesh. The ASJ graph achieves the least execution time improvements (4.6% improvement with WiFBF and 3% improvement...
with WiNoC, when compared to the wireline mesh. The reason for the variations in execution time improvement can be understood by analyzing the runtime characteristics of Grappolo. Figure 4.14 shows the varying traffic volumes and the average CPU utilizations, over the entire Grappolo execution period with the WiNoC architecture. Here, traffic volumes indicate the volume of flits exchanged through the interconnection network. As discussed earlier, the Grappolo application involves multiple executions of two distinct phases, clustering and compaction. Since all computing cores are highly active during the clustering phase, it is identified with high average core utilizations. However, the master core is the only core that is active during compaction. This serialization within the compaction phase is identified with a lower average CPU utilization.

Since all the cores are active, inter-core traffic volumes associated with the clustering phase are much higher than that of the compaction phase. It can also be noted from Figure 4.14, that the traffic volume decreases with time during the clustering phase. As the execution of the community detection algorithm progresses, the number of clusters reduces, and
consequently the interactions between the cores also begin to reduce, leading to the observed drop in traffic volume. Similar trends were observed with all the other NoC architectures albeit with different execution times (Figure 4.13).

These variations in CPU utilization and network traffic for the above mentioned input graphs lead to the observed variation in runtime improvements achieved with different NoC architectures. By comparing Figure 4.13 and Figure 4.14, we can note that the graphs with longer clustering durations achieve higher execution time improvements through the use of
low hop count NoC architectures such as WiNoC, FBF and WiFBF. When compared to wireline mesh, the low hop count NoCs enable a faster data exchange among the computing cores, leading to a quicker executions of the traffic-intensive clustering phase.

### 4.4.5.2 Balanced Coloring: Execution Time

Figure 4.15 shows the execution times of the Balanced Coloring application on the multicore system interconnected with the six different NoC architectures. These execution times are broken down into the two major phases within computation – viz. initial coloring and redistribution. Among the graphs considered, CAC achieves the maximum improvement in execution time (38% by using the WiNoC and 40% with WiFBF, when compared to the wireline Mesh). CAD follows CAC ((32%) by using the WiNoC and 33% with WiFBF when compared to the wireline Mesh). The two street network graphs, BLG and NTH achieve the least execution time variations with varying NoC architectures. NTH achieves only a 7.5% execution time improvement by using the WiNoC and only 9.3% with execution time improvement by using WiFBF, when compared to using wireline Mesh.
Figure 4.16 compares the traffic injection rates of the initial and redistribution phases. The average number of flits injected per cycle per core gives the traffic injection rate. As explained in Section 3.1.2, the redistribution phase involves heavy data migration. Due to their communication intensive nature, the redistribution phases exhibit up to 4x higher injection rates when compared to corresponding initial coloring phases. Hence, as shown in Figure 4.15, redistribution phases achieve higher execution time improvements with improved NoC architectures when compared to the respective initial coloring phases.

The number of colors associated with each considered graph is provided in Table 3.3. It can be noted from Figure 4.15 that the graphs with a larger number of colors (>80 colors) achieve a better execution time improvement with better NoC architectures when compared to graphs with low number of colors (<20 colors). This disparity in the improvement profile can be explained as follows: The graphs with more colors also spend more time in the redistribution phase, which as explained above stands best to gain from the low hop count NoC architectures. Observing the trends in traffic volumes and CPU utilizations from Figure 4.15 and Figure 4.17 can corroborate this. More specifically, the CPU activity drops during the redistribution phase when executed with the mesh topology. With higher average hop
count, mesh NoC forces the CPUs to exhibit prolonged waits for the necessary data, leading to a drop in CPU activity. In contrast, WiNoC maintains a fairly high CPU utilization throughout the redistribution period.

### 4.4.5.3 Full System Energy Consumptions

Figure 4.18(a) and Figure 4.18(b) shows the energy consumptions of the multicore systems executing *Grappolo* and Balanced Coloring, employing the six different NoC architectures considered in this work. The processor-level statistics generated by the GEM5 simulations are incorporated into McPAT (Multicore Power, Area, and Timing) to determine the
processor-level power values [94]. The energy computations associated with the different NoC components were explained earlier in Section 4.4.

The overall energy consumption of the multicore system is the sum of the energies consumed by the cores, interconnects and network router. As stated earlier in Section 4.3.3, the FBF and WiFBF architectures dissipate high router energies and hence consume higher network energies when compared to SWNoC, WiNoC and WiMesh architectures. The variation in network energy consumption among the mesh and small world networks can be explained as follows: The average hop count of the wireline mesh network is higher than WiMesh and the small world NoC architectures (Figure 4.11(a)). Hence, each injected message stays longer in the wireline mesh network than in the WiMesh, SWNoC and WiNoC architectures. This in turn leads to higher network energy consumptions with the wireline mesh NoC. As an example, we can consider the execution of CAD Balanced Coloring with mesh and WiNoC architectures. By comparing figures 15(a) and 15(b), it can be seen that the average traffic volume associated with mesh is higher than the average traffic volume associated with WiNoC. Moreover, compared to the wireline NoCs (wireline mesh, SWNoC and FBF) architectures, the WiMesh, WiNoC and WiFBF NoCs use more energy efficient wireless links for long-range communication. This enables WiMesh, WiNoC and WiFBF to achieve a better interconnect energy consumption than their wireline counterparts. Finally, when compared to the wireline mesh, all the other architectures enable a lesser execution time, leading to improved CPU energy consumptions.

Overall, the WiNoC architecture achieves lowest energy dissipation. More specifically, WiNoC has the lowest network energy dissipation and the second lowest core energy
Based on the graphs considered, Balanced Coloring achieves a maximum of 44.3% reduction in energy consumption by using WiNoC when compared to wireline mesh.
Grappolo achieves a maximum of 35.5% energy savings. The CPU energy savings are more pronounced for the Balanced Coloring application due to the communication intensive nature of redistribution phase and hence Balanced Coloring application achieves higher full system energy savings than Grappolo.

4.4.6 Most Suitable NoC for Big Data Processing

In this section, we evaluate the full system energy delay product (EDP) all the NoCs and identify the most suitable architecture for Big Data processing. Figure 4.18(c) and Figure 4.18(d) show the full system EDPs of the multicore systems executing Grappolo and Balanced Coloring, employing the six different NoC architectures considered in this work. We use full system execution times and full system energy consumptions to compute full system EDP. Due to the tradeoffs associated with reducing energy at the cost of the execution time performance, EDP is a suitable metric for analyzing the full system performance profile. To summarize, among all the considered NoC architectures, WiNoC achieves the best full system energy delay product. WiNoC is followed by WiFBF. Compared to the WiFBF, the WiNoC achieves a maximum of 12.5% EDP enhancement (an average of 9% EDP savings for WiNoC over WiFBF). For the graph applications considered here, when compared to the traditional wireline mesh and the wireline FBF NoCs, the WiNoC achieves on an average of 38% and 18% EDP enhancements respectively. Moreover, it is already shown in Section 4.4.3, that the WiNoC is also a high robust architecture that can more efficiently handle the wireless link failures. From these discussions, we can conclude that the small-world network-enabled WiNoC is the most suitable NoC architecture for implementing complex large-scale graph analytics on the multicore platforms.
4.5 Customized Voltage Frequency Islands for MapReduce

To efficiently scale up Big Data operations, minimizing the energy consumption of the manycore platforms is essential. Thankfully, MapReduce applications involve multiple unique phases giving rise to a wide variety of processor core and network utilization patterns (explained in Chapter 3). Exploiting this diversity, the energy efficiency of the manycore platforms running MapReduce can be highly improved by employing voltage-frequency islands.

Multiple Voltage Frequency Island (VFI)-based designs are common for designing low-power high performance manycore chips [97][98][99][100]. The basic principle of a VFI-based design is that the entire chip is divided into multiple clusters depending on the computation and communication requirements of the cores [97]. Each cluster is then assigned a suitable voltage/frequency (V/F) pair according to the cluster’s computation and communication characteristics. By fine-tuning the V/F pairs of each cluster it is possible to achieve significant energy savings within a certain performance constraint.

As the execution of Big Data applications on a manycore platform generates varying workload patterns (explained in Chapter 3) depending on the execution stages, we can accordingly map the cores into several V/F domains. Hence, VFI-enabled manycore design is capable for implementing energy efficient Big Data processing. The design of VFI-enabled manycore platforms can be highly optimized depending on the nature of the application. As a case study, here we showcase the potential of such an implementation for Phoneix++ MapReduce applications.
4.5.1 Design of VFIs for MapReduce

In this section, we describe how we create various VFI clusters in a multicore chip. Then, we discuss the additional analysis needed to fine-tune the VFIs for Phoenix++ applications. This includes adjusting the V/F of certain clusters to accommodate bottleneck cores that exist in Phoenix++ applications and task stealing modifications to account for the VFI creation.

An ideal VFI-based clustering encapsulates most significant inter-core communication within VFIs so that the overall network latency can be improved. It is also expected to group cores of similar utilization into the same cluster so that per-island voltage/frequency scaling can be more effective. Therefore, when implementing VFI-based clustering, we take both processor utilization and inter-processor communication into account. The baseline architecture consists of $n$ homogeneous cores, which are clustered into $m$ equally sized VFIs.

In this work, we consider 64 homogeneous cores clustered into four $4 \times 4$ equally sized VFIs.

We start by introducing some clustering notations. Per-core utilization is denoted by $u \in [0,1]^{n \times 1}$, which is defined by the number of instructions committed on average in each cycle normalized with respect to the issue width of the core. The traffic data between every source-destination pair, $f_{ip}$, denotes the number of packets sent from core $i$ to core $p$ per unit time.

We also use $X_{ij}$ to indicate whether or not a core is assigned to a cluster, namely $X_{ij} = 1$ if core $i$ is assigned to cluster $j$, else $X_{ij} = 0$. Since $f_{ip}$ and $u$ vary for different benchmarks, the creation of VFIs is benchmark specific. We define the combined objective of cost as:

$$
\min \left\{ \omega_c \sum_{i,j,p,q} X_{ij} X_{pq} f_{ip} \varphi_{comm}(j, q) + \omega_u \sum_{i,j} X_{ij} (u_i - \bar{u}_j)^2 \right\} 
$$

(4.5)
subject to

$$\forall i: \sum_j X_{ij} = 1 \text{ and } \forall j: \sum_i X_{ij} = n/m, \text{ where } X_{ij} \in \{0,1\}$$

where $\varphi_{comm}$ is the normalized inter-cluster communication cost function, defined as:

$$\varphi_{comm}(j, q) = \begin{cases} 
1 & \text{if } j \neq q \\
1/\sqrt{m} & \text{if } j = q
\end{cases} \quad (4.6)$$

$\bar{u}_j$ denotes the mean in each $m$-quartile of the utilization values. Intuitively, the first half of the objective evaluates total communication cost and the later half evaluates the utilization cost via inter-cluster core utilization variation. $\omega_c$ and $\omega_u$ are the weights for communication and utilization costs, respectively.

The communication cost function $\varphi_{comm}$ assumes that intra-cluster communication cost is evaluated as $1/\sqrt{m}$ of inter-cluster communication cost. This is because, on average, the number of hops required for inter- vs. intra-cluster communication comes as a ratio of $\sqrt{m}$:1 if an $m$-cluster (i.e., $m$ equal partitions) implementation is assumed and the $m$ clusters form a square grid. The parameters $\omega_c$ and $\omega_u$ can be arbitrary real numbers and only the relative magnitude of these two parameters matters. Indeed, if $\omega_c$ is significantly greater than $\omega_u$, the solution will be very effective on grouping significant inter-core communication within clusters. On the contrary, if $\omega_c$ is significantly less than $\omega_u$, the optimization approaches minimize variation of intra-cluster core utilization. In our setup, we normalize the elements in $f$ and $u$ with respect to their maximum values and choose $\omega_c = \omega_u = 1$. The objective in (4.5) is a quadratic function in $X$; the problem is therefore a 0-1 quadratic programming
which is NP-hard and needs to be solved using a branch-and-bound approach since the variables are all discrete. We use a commercial solver, Gurobi [101], to solve it.

4.5.1.1 Modifying V/F Assignment

In Section 3.3.2, we already explained the core utilization pattern seen in Phoneix++ MapReduce applications indicating that except for a few master cores, the utilization is nearly homogeneous among all processing cores. With this type of nearly homogeneous core utilization, depending on the impact of the traffic patterns, the bottleneck cores may fall within a cluster with a lower than required V/F leading to a high execution time penalty. The lower V/F values will slow down the bottleneck cores leading to slowing down the execution of entire application.

For PCA, MM and HIST, the core utilization pattern can lead to suboptimal V/F allocation for the master cores, for the reasons discussed above. As the number of master cores is much less than the size of the cluster, creating a separate cluster for the bottleneck cores is not beneficial due to the associated overhead. Moreover, creating a specific cluster for the bottleneck cores will deviate from our baseline uniform VFI clustering discussed in Section 4.5.1.

To address the execution time penalty created by the slowing down of bottleneck cores, a reassignment of V/F for the designed clusters is carried out specifically for PCA, Matrix Multiplication and Histogram. During the V/F reassignment, clusters in which the bottleneck cores reside are assigned with higher V/F values, while the V/F values of the other clusters remain unchanged. The locations of the bottleneck cores are not shifted to another cluster so that the traffic patterns remain unchanged. Among the other MapReduce applications
considered in this work, Linear Regression has very little library initialization period and does not have any merge period. Word Count has a non-homogeneous core utilization pattern like Kmeans. Hence, for these two applications also we do not require any V/F reassignment.

4.5.1.2 Modifying Task stealing

The necessity for modifying the default task stealing mechanism can be understood from the following case study of the Word Count application. For the Word Count application, among the four VFI clusters created, two clusters operate with one V/F value while the other two operate with another V/F value. Thus, half the cores are assigned to operate with a frequency of $f_1$ while the remaining cores are assigned to operate with a frequency of $f_2$ (with $f_2<f_1$).

While executing Word Count on the 64-core system with a large input dataset (100 MB input), the default phoenix scheduler creates 100 map tasks at the beginning of the Map phase and each core starts to perform Map operation on one of the 100 tasks. For each core, the duration spent on this initial Map task depends on the data unit assigned to it and the frequency of operation. The run time range (in seconds) for the initial Map task is given by:

- Cores with frequency $f_1$: 0.268 to 0.284 (average: 0.270)
- Cores with frequency $f_2$: 0.280 to 0.342 (average: 0.320)

Thus, certain cores with the low frequency ($f_2$) may finish earlier than the cores with the high frequency ($f_1$), depending on the data assigned. With the default task stealing mechanism, the low frequency cores that finish their initial Map task before 0.284 seconds would steal one of the unfinished tasks from cores running with $f_1$, leaving specific high frequency cores with no task to perform after 0.284 seconds. However, more often than not,
such a stealing would result in an execution time penalty, as the average task execution
duration on cores with frequency $f_2$ is higher than that on cores with frequency $f_1$. Hence,
for VFI systems, it is necessary to modify the default task stealing mechanism in such a
manner that the cores with higher V/F do not remain unnecessarily idle. To prevent the cores
with lower V/F from performing an undesired task stealing, we restrict the number to tasks
performed by cores with lower V/F to $N_f$ defined by (4.3).

$$N_f = \left\lfloor \frac{N}{C} \times \left(1 - \frac{f_{\text{max}} - f}{f_{\text{max}}} \right) \right\rfloor, \quad \forall \ f < f_{\text{max}} \quad (4.7)$$

where $N$ is the total number of tasks to be performed, $C$ is the number of cores, $f$ is the
frequency assigned to the core and $f_{\text{max}}$ is the maximum frequency of operation. Figure 4.19
presents the complete VFI platform design flow. The system with initial clustering is denoted
as VFI 1. The reassignment of V/F values for clusters in VFI 1 results in the final VFI system,
named VFI 2 system.

### 4.5.2 Wireless NoC Architecture for VFIs

Now, most of the existing VFI-partitioned designs use the conventional multi-hop mesh-
based Network-on-Chip (NoC) architecture. For large-scale systems, the inter-VFI data
exchanges through traditional mesh NoCs introduce unnecessary latency and energy
overheads. Consequently, finding new architectures that can significantly improve the
energy consumption and data rates of massively integrated multicore platforms is of great
interest. As described in Section 4.4.9, the WiNoC interconnected manycore platforms
provide the best full system performance for Big Data computing by efficiently handling the complex memory access patterns. Hence, we employ a customized WiNoC architecture to support efficient data exchanges among various VFI domains running MapReduce. This is done by creating the wireline network, mapping threads and placing the wireless links using the knowledge of the VFI domains and their traffic characteristics.

As explained in Section 4.3.2, the wireline links in WiNoC are designed using a power-law model. We assume an average number of connections, $\langle k \rangle$, from each NoC switch to the other switches. The value of $\langle k \rangle$ is chosen to be four so that the WiNoC does not introduce any additional switch overhead with respect to a conventional mesh. Also, an upper bound, $k_{max}$, is imposed on the number of ports attached to a particular switch so that no switch becomes unrealistically large. This also reduces the skew in the distribution of links among the switches. There is no specific lower bound on the number of ports attached to a switch but a fully connected network implies that this number must be at least 1. Both $\langle k \rangle$ and $k_{max}$ do not include the local NoC switch port to the core.

Due to the nature of the VFI clustering, additional constraints need to be applied to the connectivity of the WiNoC. The distribution of links is divided into two steps: VFI intra-cluster connections need to ensure each cluster’s connectivity, and VFI inter-cluster
connections, to enable communication between the clusters. This is to ensure that both intra-cluster and inter-cluster communications have sufficient resources and none of them becomes a bottleneck in the overall data exchange.

For each switch, \( \langle k \rangle \) is divided into two parts, \( \langle k_{\text{intra}} \rangle \) and \( \langle k_{\text{inter}} \rangle \), the average number of intra-cluster and inter-cluster connections to other switches respectively. For the VFI intra-cluster connections, each cluster is treated separately. A network is created for each cluster such that the connectivity follows the power-law model; the network cluster is fully connected and has an average intra-cluster connectivity, \( \langle k_{\text{intra}} \rangle \).

The VFI inter-cluster connections are created such that the connectivity also follows the same power-law model as the intra-cluster connections, has an average inter-cluster connectivity, \( \langle k_{\text{inter}} \rangle \). The number of links going from one cluster to another is decided by the inter-VFI traffic. The proportion of links allocated between two clusters is directly related to the proportion of inter-cluster traffic between the two clusters in total inter-cluster traffic.

### 4.5.3 Performance Evaluation

We evaluate the performance of a 64 core multiprocessor system running Phoenix++, in the presence of the above-discussed VFI clustering and WiNoC interconnection architecture. For this evaluation, the same experimental setup explained in x is followed. The applications considered are PCA, Word Count (WC), Kmeans, Histogram (HIST), Matrix Multiplication (MM) and Linear Regression (LR).
4.5.3.1 Impact of Master Cores and Efficiency of V/F Reassignment

In this section, we analyze the impact of master cores on the execution time of these applications and evaluate the benefits achieved by V/F re-assignment. Figure 4.20 presents the execution times and Energy Delay Products (EDP) of PCA, HIST and MM for the two VFI systems, the initial VFI system (VFI 1 in Figure 4.19) and final VFI system with V/F reassignment (VFI 2 in Figure 4.19). The shown execution time and EDP values are normalized with respect to a baseline non-VFI system with mesh interconnection (NVFI Mesh). For all these three applications, the V/F value of the cluster with bottleneck cores was increased to 1.0V/2.5 GHz in VFI 2 system from 0.9 V/2.25 GHz in VFI 1 system. The V/F values of all other clusters remain unchanged. Table 4.1 shows the V/F values for all the benchmarks in both the VFI configurations (reassigned V/F values are highlighted).

From Figure 4.20(a), we can observe that PCA benefits most by re-assigning the V/F values, followed by MM. The reason for the execution time improvement in VFI 2 system can be understood from Figure 4.21 which presents the average and bottleneck cores’ utilization values for PCA, HIST and MM. PCA has the highest bottleneck core to average core busy
utilization ratio, followed by MM. Higher this ratio, higher is the impact of bottleneck core on the execution time. As explained in section 4.5.1, with the initial V/F assignment, bottleneck cores were placed in clusters with low V/F values. This slows down the bottleneck cores, leading to VFI 1 system experiencing a higher execution time penalty when compared to the NVFI mesh. However, by increasing the V/F values of the whole cluster we may lose energy saving opportunity. We can observe from Figure 4.20(b) that this execution time improvement in VFI 2 system has been achieved without noticeable increase in the energy consumption. EDPs of the PCA and MM are improved for the VFI 2 system compared to the VFI 1 system whereas HIST pays no EDP penalty. Hence, we can conclude that the

<table>
<thead>
<tr>
<th>Cluster 1 (V/GHz)</th>
<th>Cluster 2 (V/GHz)</th>
<th>Cluster 3 (V/GHz)</th>
<th>Cluster 4 (V/GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFI 1</td>
<td>VFI 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
</tr>
<tr>
<td>HIST</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
</tr>
<tr>
<td>Kmeans</td>
<td>0.6/1.5</td>
<td>0.6/1.5</td>
<td>0.8/2.0</td>
</tr>
<tr>
<td>WC</td>
<td>0.8/2.0</td>
<td>0.8/2.0</td>
<td>1.0/2.5</td>
</tr>
<tr>
<td>PCA</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
</tr>
<tr>
<td>LR</td>
<td>1.0/2.5</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
</tr>
</tbody>
</table>

Figure 4.21 MapReduce Applications: Core utilization trend
improvement in execution time due to V/F re-assignment is more than the decrease in energy savings.

### 4.5.3.2 Network Performance

In this section, we evaluate the optimal network parameters for the WiNoC to be used with the MapReduce applications under consideration. To do so we focus on the network parameters, wireless methodology, $\langle k_{\text{intra}} \rangle$ and $\langle k_{\text{inter}} \rangle$ parameters discussed in section 4.5.2. We vary each of these parameters and use their network energy-delay-product (EDP) as the performance metric. Due to the nature of fully connected networks, our VFI clusters of 16 cores require a $\langle k_{\text{intra}} \rangle$ of at least 1.875. As the value of $\langle k \rangle$ is 4, the values of $(\langle k_{\text{intra}} \rangle, \langle k_{\text{inter}} \rangle)$ can be either $(3,1)$ or $(2,2)$. We have found that $(3,1)$ configuration always performs better than $(2,2)$. Hence, this network configuration is used in all further performance analysis. Figure 4.22 demonstrates the reduction in EDP when using the maximized wireless utilization over the minimized hop-count strategy. For all benchmarks considered, the system with maximized wireless utilization and $(\langle k_{\text{intra}} \rangle, \langle k_{\text{inter}} \rangle)$ values equal to $(3, 1)$ perform consistently better than other alternatives.
For VFI-based designs, one of the main goals is to reduce energy while maintaining an acceptable level of performance degradation. Therefore, in order to evaluate our VFI-enabled system, we present both execution time and the whole system EDP parameters.

First, we consider the overall execution time of the designed system with respect to the baseline non-VFI (NVFI) mesh-based system. Figure 4.23 shows the execution time relative to NVFI mesh for both VFI mesh and VFI WiNoC. As expected in the presence of VFI, traditional mesh-based designs suffer a degradation of up to 10.5% of execution time due to the lowered V/F values. Since the WiNoC is able to transfer data across the network faster giving rise to faster non-local L2 access, the performance degradation is lowered and eliminated for certain benchmarks under consideration. MM, WC, LR, and Kmeans were all able to execute quicker using WiNoC-based VFI than NVFI mesh.

Among all the applications, WC and Kmeans achieve the highest execution time improvements with an improved interconnection network (15% and 14% respectively).
Both applications have a high number of MapReduce Keys and also exhibit high amount of traffic interaction between the cores that are distant. With an improved network, the latencies in transferring the Keys and Values among the cores get highly reduced, leading to better execution times. Among all the applications studied, LR application has the highest traffic injection rate as it exchanges large data units with nearer cores in the Map phase. Due to its nearer-core heavy communication pattern and dependency on core-main memory interaction, the improvement achieved for LR with WiNoC is limited (4%).

Due to the tradeoffs associated with reducing energy at the cost of performance, the energy-delay-product (EDP) is the most relevant metric when analyzing the energy profile. By delay we consider the execution time here. Figure 4.24 shows the EDP for the VFI configurations in relation to NVFI mesh. The variation of the differences between the VFI Mesh and VFI WiNoC EDP arise from each application’s core interaction rates. For example, LR has the greatest core interaction rate and difference between the VFI mesh and VFI WiNoC EDP. This can be attributed to WiNoC having lower energy consumption compared to the Mesh architecture.

Furthermore, it can be observed from Figure 4.24 that Kmeans achieve the highest EDP savings with respect to NVFI mesh. As explained in Section 3.3.2, the core utilization values for Kmeans varies highly over the 64 cores. It can be seen in Figure 3.5, that about 32 cores have less than 50% utilization when compared to the average utilization. Such cores can be operated with significantly a lower V/F, without affecting the execution time leading to higher energy savings.
In summary, we can see that for all of the benchmarks considered, we are able to save significant EDP by using WiNoC over NVFI mesh, (maximum 66.2% and 33.7% on average). These large reductions in EDP are due to enhancement of the energy dissipation profiles of both the processing cores and the interconnection network without noticeable compromise in execution time. This demonstrates the ability for WiNoCs to enable VFI-based designs, greatly reducing energy without impacting the performance of traditional mesh-based systems.

4.6 Summary

Implementing advanced graph analytic and MapReduce operations on state-of-the-art manycore platforms requires significant redesign of the on-chip interconnect network topologies to mask the adverse effects of irregular data movement characteristics. Towards this end, in this work we explore the suitability of three different NoC topologies, mesh, small world Network on Chip and High-Radix, for implementing multicore graph analytics. Incorporated with long-range shortcuts, the small world and Flattened Butter-Fly topologies enable a fast data exchange among the computing cores leading to improved system
performance when compared to the traditional wireline mesh. Among all the NoC architectures considered in this chapter, the wireless enabled small-world Network on chip (WiNoC) architecture achieves the best full system Energy Delay Product (EDP). In an effort to achieve energy efficiency on both processing cores and interconnect fabrics, we have integrated VFI and WiNoC in a single manycore platform running MapReduce applications. With efficient V/F assignments, it is possible to achieve significant energy reduction for MapReduce applications while keeping the execution time penalties to a minimum.
Collective communication, i.e., multicast/broadcast, lies on the critical path of many real world applications with intensive data exchange requirements on many-core platforms. Neural network based computing [102], real-time object recognition processing [103], genetic algorithm based protein folding analysis [104] and neuromorphic computing [105] are few of the SoC applications that exhibit significant multicast traffic patterns. Moreover, many SoC control functions such as passing global states, managing and configuring the on-chip networks and implementing (a) operand exchange networks, (b) non-uniform cache architectures and (c) cache coherency protocols also require efficient multicast data exchange. To elaborate on this, as examples, we consider the traffic patterns associated with two different cache coherence protocols, Directory [81] and Hammer protocols [19]. Hammer and Directory cache coherence protocols present two vastly varying yet challenging cases of NoC traffic injections. As explained in Chapter 3, both Directory and Hammer protocols give rise to traffic hotspots and generate significant amount of long-range traffic patterns warranting low latency NoC links, even among the physically far apart processing cores. However, unlike the Directory protocol that only involves sparse multicasts, Hammer protocol involves dense broadcast injections and hence, stresses the NoC affecting the overall performance when implemented on systems predominantly designed for unicast-
based transmissions. Moreover, the multicast traffic arising due to the cache coherence protocols is inherent to the manycore platform and can be observed irrespective of the executed applications. On the other hand, the on-chip area overhead associated with implementing the Hammer protocol is very low and hence it is highly preferable for large manycore chips performing Big Data computing than the traditional full-bit Directory protocol. Hence, designing a high performance multicast-aware NoC that enables efficient implementation of Hammer cache coherence protocol is a highly relevant research topic [14]. In addition, cache coherence protocols also exhibit multicast and ACK bursts at a particular instances of time and thus, necessitating a collective communication and congestion aware NoC.

In this Chapter, we present the design of a collective communication-aware wireless NoC (WiNoC) incorporating congestion-aware routing and network coding technique to handle high volumes of multicast injections. The resultant WiNoC improves the overall execution time of the manycore chip and lowers the total energy dissipation. We demonstrate the efficacy of the proposed WiNoC through case studies using the above-discussed Hammer cache coherence protocol.

5.1 Multicast-aware WiNoC Network Design

As discussed earlier, in this work, we strive to design a congestion-aware low hop count multicast-enabled WiNoC. In this context, we present the design of a small world network-based multicast enabled WiNoC architecture. For this WiNoC, we employ the same wireless interface design explained in Section 4.1.2.
5.1.1 WiNoC Design Constraints

In our WiNoC, each processing core is connected to a router and the routers are interconnected following a power-law based small-world (SW) connectivity. Essentially, SW networks incorporate multiple short-range links and a few long-range shortcuts so as the overall inter router hop count \( (H_{avg}) \) is minimized. However, while adding links in small world NoCs, we need to follow certain restrictions. First, we should restrict the average number of ports \( (K_{avg}) \) per router to four so that the WiNoC does not introduce any additional router port overhead when compared to a conventional mesh. Next, we should restrict the maximum number of ports in a router \( (K_{max}) \) so that no particular router becomes unnecessarily large. For a system size of 64 cores, the small world based networks achieve highest throughput with lowest energy dissipation when the maximum port count in a router is restricted to seven [58]. Finally, as we explained in Section 4.1.2, for link lengths exceeding 6.25mm in 28nm CMOS technology node, wireless links are more efficient than the traditional metal wires in terms of energy-delay product (EDP). Hence, WiNoCs should prefer using wireless links instead of wireline links for data exchanges exceeding a communication distance of 6.25mm. However, depending on the available wireless resources and allowed area overhead, we can only make a limited number of the longest links wireless, while the others need to remain wireline.

Considering these facts, for our WiNoC, we follow a region based wireless node placement strategy. In this strategy, the whole system is divided into multiple equal sized non-overlapping regions and each region is provided with a set of Wireless Interfaces (WIs). Principally, in this design, the short range communication (within a region) is handled
through the wireline links while the long range communication (inter region data exchange) is handled using wireless links. In each considered region we place the WIs such that the

![Diagram](image)

(a) WiNoC Multicast Routing
(b) Connectivity of Quadrant Q2
(c) Primary distribution links
(d) Secondary distribution links

Figure 5.1 (a) A sample WiNoC multicast scenario where a source node (0) in quadrant Q1 is transmitting a multicast message to destinations across three different quadrants. The destination nodes are in red and the routers with the WIs are indicated in green. The color of the square that is surrounding a WI indicates the channel assignment. Wireline communication is indicated with solid lines and the wireless data transfer is indicated with dotted lines. (b-d) The Wireline connectivity and the two orthogonal distribution paths associated with quadrant Q2 of the WiNoC is illustrated through figures b-d. The wireline network is created with a $K_{avg}=4$ and $K_{max}=7$. 
communication distance from all the non-wireless nodes to their nearest WIs is minimized. The number of WIs allowed in a region is equal to the number of available non-overlapping wireless channels. For a system of 64 nodes, it is already shown that placing the WIs at the center of each region minimizes the communication distance from any node to its nearest WI and hence it ensures that the utilization of wireless medium is maximized [59]. Figure 5.1(a) shows an example where a system of 64 cores is divided into 4 regions (quadrants), with each quadrant having three WIs in its center.
5.1.2 Orthogonal Paths in WiNoC

In order to facilitate the congestion aware multicast routing (discussed in Section 5.2), we propose that each non wireless node in the system is connected to all the WIs in its own region via two orthogonal wireline paths, where none of the links in these two paths are being the same. This can be achieved by creating two orthogonal spanning trees for each region from the existing wireline connectivity.

In order to more clearly explain the orthogonal paths concept, Figure 5.1 shows the entire wireline connectivity of a region and the two orthogonal spanning trees associated with it. Figure 5.1(b) illustrates the entire wireline connectivity associated with quadrant Q2. Figure 5.1(c) and (d) show the two orthogonal spanning trees created from the wireline connectivity shown in Figure 5.1(b). The links constituting the connectivity shown in Figure 5.1(c) and (d) are called the primary distribution links and the secondary distribution links respectively. Among these two sets of links, the set of primary distribution links (Figure 5.1(c)) provide minimal hop count paths between all non-wireless nodes and their nearest WI. Hence, as explained later in Section 5.2, these links are usually preferred for distributing messages from a WI to its nearest non-wireless destinations and thus have the name primary.

Figure 5.3 The overall WiNoC design flow to enable MALASH routing.
distribution links. The other set of links, the secondary distribution links will be used for this distribution in case the primary distribution links are not available due to congestion. Moreover, as it can be observed from Figure 5.1 that the primary and secondary distribution links are merely subsets of the existing intra quadrant wireline connectivity and hence, employing orthogonal wireline paths does not introduce any additional wireline overheads.

Finally, following the above-discussed design principles and constraints, we employ a simulated annealing-based (SA) mechanism to place the wireline links so as the average inter router hop count ($H_{avg}$) is minimized (shown in Figure 5.2). In this SA-based mechanism, at the start of each cycle, a perturbation is created in the current network connectivity to create a new network configuration. In this perturbation, it is also ensured that the input constraints such as $K_{max}$, $K_{avg}$ are preserved. Then, the wireline connectivity of each region is divided into pairs of orthogonal spanning trees (e.g. each pair has two spanning trees named $T1$ and $T2$). For each region, the links of the spanning tree (say $T1$) that provide the minimal hop count paths between all non-wireless nodes and their nearest WI constitute the set of primary distribution links. The links in the spanning tree which is its orthogonal pair ($T2$), constitute the set of secondary distribution links. It should be noted that this wireline link addition mechanism will work for a WiNoC with any number of regions and also for any region size. The only constraint that should be met is that for a region with $N$ nodes, there should be at least $2(N-1)$ intra region links in order to establish the two orthogonal spanning trees (this is because each spanning tree is constituted of $N$-1 links).
5.2 Routing and MAC Protocols

The WiNoC has an irregular network topology and hence requires a topology agnostic routing method. Thus, principally, we follow ALASH routing with priority layering for data exchanges in WiNoC (introduced in Section 4.2.2.2). In the multicast-aware WiNoC, for all source-destination pairs residing in the same region, we perform the priority layering for shortest paths along both the primary and the secondary distribution links. The entire WiNoC design flow to achieve a layered routing enabled WiNoC is shown in Figure 5.3. In this figure, $F_{ij}$ denotes the matrix containing frequency of traffic interactions between all the routers; $N_{VL}$ denotes the number of available virtual layers. In order to perform efficient multicast routing, we propose to expand on the basic protocol and design multicast-aware ALASH protocol (called as MALASH). The salient features of MALASH collective communication routing is explained next.

5.2.1 Transmission of Messages

In MALASH routing, the multicast messages are distributed using a region-based approach (as shown in Figure 5.1(a)). To distribute the message to destinations that are present in the same region as that of the source, MALASH uses wireline links. To distribute the message to the destinations located in the non-source regions, MALASH uses a combination of wireless and wireline links. First, the multicast message is forwarded from the source node to the nearest WI. Then, from this WI, the message is broadcast using the wireless channel. Finally, WIs residing in the non-source regions receive this broadcast message and distribute it to the intended destinations using wireline links. This wireline distribution is carried out using the tree multicast mechanism where the message is first forwarded to a set of intermediate
nodes. The message is then replicated at these intermediate nodes before being forwarded to the intended final destinations [107]. Following the congestion aware routing mechanism, (explained later in Section 5.2.2), either the primary or the secondary distribution links are used to reach a final destination node from the destination WI.

5.2.1.1 Example

As an illustrative example, let us consider the situation shown in Figure 5.1(a) where the whole system is divided in to four quadrants (regions). Here, node 0 present in Q1 is transmitting a multicast message to destinations 11, 15, 25, 39 and 62. To deliver the messages to nodes 11 and 25, MALASH follows the wireline distribution. However, to reach nodes 15, 39 and 62, MALASH utilizes one specific wireless channel. To transmit the multicast message to these non-source quadrant destinations, the injected message first gets routed to the nearest WI (node 9) in Q1. From this WI, the message is broadcast using the wireless channel to simultaneously reach the WIs 14 and 46 in Q2 and Q4 respectively. Again, from nodes 14 and 46 we follow the wireline paths to reach node 15 in Q2 and nodes 39 and 62 in Q4. We illustrated this example with the help of one particular wireless channel. The same method will be repeated using other wireless channels depending on the locations of the source-destination pairs.

The following two subsections describe how the multicast message is distributed from the destination WI in each region to the final destination nodes using the wireline links.
5.2.2 Congestion Aware Routing

As explained earlier in Section 3.4, for Hammer cache coherence protocol induced traffic patterns, multiple broadcast injections can occur within short windows. Such injection can specifically lead to heavy wireline traffic congestion caused by forwarding of multicast packets to their final destinations from the destination WIs. MALASH addresses this issue by employing a congestion aware routing.

As explained in Section 5.1, in WiNoC for all source destination pairs within a region, two different paths are available (along the two orthogonal spanning trees). For a source \((i)\) and destination \((j)\) pair, the shortest among these two paths is referred as path \(P_{ij1}\) while the other path is referred as \(P_{ij2}\). A multicast message has a higher impact on the overall network performance than a unicast message. Hence, in MALASH, the multicast messages are treated with a higher priority over unicast messages. With this in mind, the congestion aware routing for forwarding messages from WIs is implemented using the following rules.

i) For a message \(M\) with priority \(p\), the path \(P_{ij1}\) is selected if \(P_{ij1}\) is free or occupied with a lower priority message. If \(P_{ij1}\) is occupied with a higher or equal priority message then \(P_{ij2}\) is selected.

ii) A message is not allowed to switch paths from \(P_{ij2}\) to \(P_{ij1}\). Once received along \(P_{ij2}\), the message must be forwarded along \(P_{ij2}\). This condition ensures deadlock freedom.

Following above rules, usually the shortest paths \((P_{ij1}~\text{paths})\) are always preferred for distributing a multicast, first from the destination WI to all the intermediate nodes and then from the intermediate nodes to the final destinations. However, when a WI has two multicast
messages to distribute, then the oldest waiting message is forwarded using primary distribution links and the other message is forwarded using secondary distribution links.

5.2.3 Employing Network Coding

In a region when two different WIs distribute two multicast messages $M1$ and $M2$, there can be scenarios where both the messages need to traverse through common intermediate nodes simultaneously. To efficiently distribute multiple multicast messages that are received at the same time, MALASH combines orthogonal paths with network coding technique. In such case, instead of forwarding both messages to the final destinations, the intermediate node creates a coded message $M3$ and forwards it to the final destination. The message $M3$ is created by XOR-ing individual bits of $M1$ and $M2$. At the final destinations, the desired message is recovered by XOR-ing one of the original messages with $M3$.

As stated, network coding is only needed when two messages $M1$ and $M2$ start their wireline distribution from different destination WIs in the region. This is a necessary condition to employ network coding. In case of $M1$ and $M2$ starting their wireline distribution from the same destination WI, the congestion can be avoided simply by forwarding $M1$ strictly along primary distribution links and forwarding $M2$ strictly along secondary distribution links. For clarity, the usefulness of network coding is further explained below through the example demonstrated in Figure 5.4(c).

5.2.4 Example Scenarios

We have demonstrated three sample scenarios in Figure 5.4 to show the application of congestion aware routing and network coding. In Figure 5.4(a) the WI node 22 receives a
multicast message $M1$ which has to be distributed to all the nodes in the region. With no other multicast message to compete in the region, $M1$ is simply forwarded along the $P_{ij}$ paths.
employing tree multicast mechanism. Destinations 4, 5, 13 and 29 are reached along the secondary distribution links while other destinations are reached along the primary distribution links. Also while distributing $M1$, the intermediate node 14 receives a unicast message $U1$ whose final destination is node 15. Since $M1$ is prioritized over $U1$, $M1$ is distributed from node 14 to destination 15 along the $P_{ij1}$ (present in set of the primary distribution links) while $U1$ is forwarded along the path available in the set of secondary distribution links.

In Figure 5.4(b) WI node 22 simultaneously distributes two multicast messages $M1$ and $M2$ along primary and secondary distribution links respectively. Figure 5.4(c) demonstrates the usefulness of network coding in the multicast distribution. In this example, at time $t$, WI nodes 14 and 21 start distributing two multicast messages $M1$ and $M2$ respectively, along the $P_{ij1}$ paths to the immediate intermediate nodes, 13, 14, 21 and 22. At time $t1 > t$, all the intermediate nodes (13, 14, 21 and 22) have the two multicast messages each to replicate and forward. In such case, network congestion and intermediate queuing is avoided by employing network coding. The coded message is forwarded along secondary distribution links form these intermediate nodes to the final destinations. At the same time, one of the original messages (both $M1$ and $M2$ are valid), is forwarded along the primary distribution links. For the illustration shown in Figure 5.4(c), the destination sets of $M1$ and $M2$ can be any two sets with common final destination nodes along the quadrant edges. Since the network coded packets are routed along secondary distribution links from the intermediate nodes (13, 14, 21 and 22 in Figure 5.4(c)) to the final destinations, they require one extra hop to reach the final destination nodes when compared to the paths along the primary
distribution links. Hence, considering $M_1$ and $M_2$ consisting of $F$ flits each, for a NoC employing routers with $S$ stages, use of network coding saves $F-S$ cycles of message queuing.
latency. Figure 5.5 summarizes the overall MALASH routing flow.

5.2.5 Aggregation of Acknowledgements

Similar to the XY-Tree based multicast NoC (mentioned in IV.A), in WiNoC, the acknowledgements are aggregated at the intermediate nodes. Initially, each destination forwards its acknowledgement message to the same intermediate node that forwarded the original multicast message. The ACKs are then aggregated at each region WIs and finally get forwarded towards the source node. The use of ACK aggregation greatly reduces the overall traffic. For a 64 core system with unicast mesh, 63 ACK messages are returned to the source for each broadcast. In WiNoC, first, only one aggregated ACK is generated for each quadrant. Following this, the source WI combines the aggregated ACKs generated in all quadrants using wireless links and forwards it towards the multicast source node.

5.2.6 Distributed Routing and Deadlock Freedom

Though MALASH multicast routing includes multiple features such as forwarding towards destination region WIs, congestion aware wireline distribution and network coding, it is distributed in nature and simple to implement. We use two additional fields $C_m$ and $R_m$ in the multicast headers (a total of 4 additional bits per message) to accomplish the distributed routing. $C_m$ is a two bit field that indicates the channel ID of the source region WI to which the message is first forwarded from the source node. $R_m$ is a 2 bit field that guides the wireline transmissions associated with the message. An $R_m$ value of 0 would indicate that the message is being forwarded from the original source node to the source region WI. All $R_m$ values that greater than 0 indicate that the message is undergoing regional wireline distribution. Table
### Table 5.1 Information needed for MALASH routing

<table>
<thead>
<tr>
<th>$R_m$</th>
<th><strong>Message Status</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Forwarded towards the source region WI</td>
</tr>
<tr>
<td>1</td>
<td>Wireline distributed along $P_{ij}$ paths, Network coding allowed</td>
</tr>
<tr>
<td>2</td>
<td>Wireline distributed, strictly along primary distribution links</td>
</tr>
<tr>
<td>3</td>
<td>Wireline distributed, strictly along secondary distribution links</td>
</tr>
</tbody>
</table>

For WI, $C_{own}$ = 1, 2 or 3; for non wireless routers, $C_{own}$ = 0

### Table 5.2 MALASH: Routing rules that are followed at all routers

<table>
<thead>
<tr>
<th>Header Values</th>
<th><strong>Rules</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_m=0$, $C_m \neq C_{own}$</td>
<td>Forward message to the nearest WI with channel $C_m$</td>
</tr>
</tbody>
</table>
| $R_m=0$, $C_m = C_{own}$ *(Source region WI)* | (a) Set $R_m=1$  
(b) Perform wireless replication |
| $R_m=1$, $C_m = C_{own}$ *(destination region WI)* | (a) Following the information given in Table. 5.1, set $R_m=1$ or 2 or 3 using congestion aware routing  
(b) Perform wireline distribution |
| $R_m>0$, $C_m \neq C_{own}$ *(Intermediate node)* | Continue wireline distribution by following $R_m$ and the information given in Table. 5.1 |

5.1 and Table 5.2 list the routing rules followed in MALASH to enable a distributed routing.

As it can be observed from Table 5.2, the MALASH routing can be simplified to four parallel conditional cases.

Aside from enabling the distributed routing, the $C_m$ field can also be helpful in identifying the necessity for network coding. It was stated in Section 5.2.3 that two messages must start their wireline distribution from different destination WIs in order to employ network coding. This condition can be easily verified by comparing the $C_m$ values of the two messages.

As explained earlier, in MALASH routing, each multicast message is allowed a maximum of one wireless traversal (at the nearest source region WI). Restricting multiple wireless traversals avoids creation of deadlock loops that can arise due to forwarding between
different WIs. This wireless traversal is followed by the final wireline distributions. As explained in Section 5.2.2, the congestion aware routing already ensures deadlock freedom in this wireline distribution. Hence, at all points, MALASH avoids creation of any cycles in the multicast message routing and is inherently deadlock free.

5.2.7 Wireless Medium Access Control (MAC) Protocol

The WiNoC uses the distributed MAC protocol explained in 4.2.3. to resolve the channel access contention among the wireless nodes. To support multicast transmission, we propose a modified version of this distributed MAC called as Distributed Priority Multicast MAC (DPM). DPM allows us to prioritize multicast packets over unicast packets to take advantage of the wireless channels’ broadcast nature. For this priority-based distributed MAC protocol, a $2N$ bit request packet is used. Each set of $N$ bits in the request packet delivers information similar to the request packet used in the distributed MAC protocol. The first set of $N$ bits is used for unicast transmission requests while the second set of $N$ bits represents multicast/aggregated ACK transmission requests. In DPM, a multicast request always has a higher priority than a unicast request. If no multicast requests are received, then the received unicast requests are processed similar to the basic distributed MAC protocol.

5.2.7.1 Efficiency of DPM

In this section, we evaluate the performance of the DPM scheme against the traditional token passing MAC in handling multicast traffic. Figure 5.6 shows the improvements achieved in latency and energy dissipation by adopting the DPM scheme over the token passing scheme. The simulated traffic injection pattern follows a uniform distribution where the injected messages are both unicast and multicast. The percentages of multicast messages in the total
simulated messages are varied from 10% to 30%. Under token passing MAC, a significant portion of the multicast traffic is handled using wireline links. This happens due to i) long channel access times and ii) the lack of priority for multicast flits in accessing wireless channels. Unlike the token passing scheme, DPM policy prioritizes multicast and allows most of the multicast transmissions to be carried by the wireless channels. Thus the DPM scheme reduces the amount of transmissions required, resulting in the enhancement of both network latency and message energy. It can be seen in Figure 5.6 with increase in multicast messages, DPM becomes more efficient than the token passing scheme. When 30% of the total messages are multicast, DPM provides 20% latency reduction and 40% energy enhancement over the token passing scheme.

5.3 Performance Evaluation

To evaluate the effectiveness of our WiNoC architecture against mesh NoC in handling collective communication and long-range unicasts, we employ Grappolo application with five
different data sets under Hammer and Directory cache coehrence mechanism. Specifically, we use a set of five DIMACS10 clustering instance graph data sets, Hep-th (HEP), Astro-ph (ASTRO), Cond-mat-2003 (COND), PGPgiantcompo (PGP) and as-22july06 (ASJ). We compare the performance of WiNoC against an industry standard NoC, the mesh architecture.

In mesh wireline networks, it is not possible to create the two sets of orthogonal links as in our WiNoC and hence it is not possible to employ the congestion-aware MALASH routing described earlier. This is because, to create two sets of orthogonal links, at least $2(N - 1)$ wireline links are needed in a region with $N$ nodes. However in mesh only $2(N - \sqrt{N})$ intra region wireline links are available. Hence, for mesh NoC, we use dimension ordered XY routing for unicast packets and XY-tree broadcast mechanism for Hammer broadcasts [14]. We also use ACK aggregation for the mesh following the method proposed in [14].

We consider a 64–tile x86 manycoressystem operating with a frequency of 2.5GHz and having a die-size of 20×20 mm$^2$. The memory system is comprised of private 64KB L1 instruction and data caches and one shared 16MB L2 cache (256KB distributed L2 per core). All the processing cores and memory system are interconnected using an NoC fabric with 64 routers. For all the NoC architectures considered, we employ a generic three stage router architecture modifying [93]. This architecture has three functional stages, namely, input arbitration, routing/switch traversal, and output arbitration with link traversal. Following routing, a multicast flit is replicated to all necessary output VCs simultaneously. The flits at output VCs are then handled by the output-arbiter. The additional cycles required for congestion aware routing and MAC protocol processing are accounted for while determining the network latency. Energy dissipation of the network routers, including the routing, MAC and network
coding blocks, was obtained from the synthesized netlist using 28nm commercial FD-SOI technology by running Synopsys™ Prime Power. In both NoC architectures, across both wireline and wireless links, we follow wormhole routing methodology. The router ports are provided with a buffer depth of two flits. The width of all wireline links is same as the considered flit width (32 bits). Each wireline is designed with the optimum number of uniformly placed and sized repeaters in 28 nm technology node. The energy dissipation of the wireline links was obtained through Cadence SPECTRE™ simulations.

We use GEM5, a full system simulator, to obtain detailed processor and network-level information. We use a modified GARNET interconnection network along with GEM5 to model the multicast supported NoCs in full system simulations performed to obtain execution times. The processor-level statistics generated by the GEM5 simulations are incorporated into McPAT (Multicore Power, Area, and Timing) to determine processor power values [94].

#### 5.3.1 Network Performance

The advantages achieved by employing WiNoC for performing graph analytics under Hammer coherence can be understood from the NoC latency plot shown in Figure 5.7. Compared to mesh, WiNoC show an average of 47% and 38% reductions in message latencies with Hammer and Directory protocols, respectively. The reasons for WiNoC gain elaborated thoroughly in the following text.

---

1 We have used 32-bit flits since Directory protocol only requires smaller flit widths [95]. However, when a NoC is specifically designed for dense broadcasts (such as Hammer protocol broadcasts), employing higher flit widths [65][96] can be beneficial for enhancing bandwidths at the network edges.
As noted earlier, Hammer protocol involves heavy broadcast-based data exchange. Compared to traditional wireline mesh NoC, WiNoC highly lowers the average number of hops needed to transfer a Hammer broadcast message. This hop-count value is determined by the number of hops taken from the source node to reach the last destination node that receives the broadcast message. XY-Tree mesh NoC only requires on an average of 12.1 hops from the source node to distribute the broadcast message. WiNoC performs far better and only requires on an average of 4.35 hops to distribute a broadcast message.

As explained in Section 3.3.2, Grappolo shows a high hot-spot heavy traffic. In addition, Hammer protocol involves heavy broadcast/ACK bursts (Sections 3.2 and 3.4). Aside from high hop counts, the wireline mesh NoCs also experience network congestion that is induced by the heavy simultaneous broadcast/ACK injections. The considered mesh NoC does not have any adaptivity in the routing and hence the messages need to wait in intermediate nodes under high traffic injections. On the other hand, WiNoC employs fast wireless data transfers over multiple quadrants and uses a congestion-aware wireline distribution along with the network coding technique. This efficient wireline routing mechanism in WiNoC
ensures that even for applications with heavy simultaneous multicast injections, the congestion induced intermediate queuing delays are eliminated and the messages are quickly distributed.

Aside from having an efficient multicast distribution mechanism, by using wireless links, WiNoC also enables a more efficient long distance unicast communication than the wireline mesh-based architecture.

Thus, on overall, WiNoC employs a more sophisticated routing algorithm (with congestion-awareness) and uses a small world wireline connectivity that reduces the overall hop count and hence, achieves better unicast and multicast message latencies.

5.3.2 Execution Time Evaluations

Figure 5.8 compares the execution times of the mesh and WiNoC interconnected manycore systems running Grappolo versions with Hammer coherence protocol. All the values in Figure 5.8 are normalized with respect to the execution times observed in the baseline system with mesh NoC. The WiNoC achieves on an average of 22.4% and 15.96% reduction in execution times with Hammer and Directory protocols respectively, when compared to the mesh NoC. The reason for the variation in gain achieved for the two cache coherence protocols can be understood from the NoC latency plot shown in Figure 5.7. When compared to the Directory protocol, the Hammer protocol induced on-chip messages give rise to a higher network latency in the baseline mesh NoC because of the fact that about 25% of the injections are broadcasts. Thus, in a mesh NoC, memory accesses cause higher penalties on the overall execution time with the Hammer protocol than with the Directory protocol.
Hence, accelerating the memory accesses using the WiNoC leads to more gain in execution time with Hammer than with the Directory protocol. Among the considered input graphs, ASJ and ASTRO involve the lowest and highest traffic injection rates, respectively. Hence, ASJ and ASTRO achieve the lowest and highest execution time gains, respectively, using WiNoC (Figure 5.8).

5.3.3 Full System Energy Consumptions

Figure 5.9 compares the energy consumptions of the manycore platforms executing Grappolo, employing Mesh and WiNoC architectures. All the values in this figure are normalized with respect to the energy consumption of the baseline architecture employing the mesh NoC.

The overall energy consumption of the manycore system is the sum of the energies consumed by the cores and the network elements. The multicast-aware WiNoC enhances both the facets of the system energy when compared to the XY-Tree multicast mesh NoC. WiNoC enhances execution times when compared to XY-Tree multicast mesh NoC. This leads
to lower core energy consumptions. In addition, the use of long-range wireless shortcuts ensures low hop count long-distance unicast message transfers in WiNoC. This eventually leads to lower number of router and link traversals in WiNoC when compared to the XY-Tree multicast mesh NoC and hence WiNoC achieves lower network energy consumption. Due to the communication-intensive nature, the Hammer protocol achieved a higher improvement than the Directory protocol by using the WiNoC. On overall, the WiNoC provides 22% and 30% full-system energy savings on average, over the XY-Tree multicast mesh NoC for running Grappolo with Directory and Hammer coherence protocols.

5.4 Summary

Big Data applications require today’s manycore architectures to handle a highly diverse and challenging on-chip traffic patterns. The Hammer cache coherence protocol is one such system-on-chip application that stresses on chip networks with heavy broadcast and acknowledgement (ACK) injections. In this Chapter, we explained the designing of a collective communication-aware wireless NoC (WiNoC) architecture that can efficiently handle cache
coherence communications. Incorporated with a congestion-aware multicast routing, network coding and a region based ACK aggregation, WiNoC eliminates the initial and intermediate queuing latencies seen in conventional wireline mesh NoCs. Moreover, using wireless shortcuts, the WiNoC achieves significant reductions in network latencies leading to improved system performances.
6 High Performance Single-Cycle Multi-Hop Traversal NoC Architectures

Traditional Network-on-Chip (NoC) interconnects follow conventional packet switching architectures that require multiple cycles to traverse each router hop. Recently, the SMART NoC paradigm is proposed as an efficient solution to minimize both unicast and collective communication latencies in traditional NoCs [66]. SMART (Single-Cycle Multi-hop Asynchronous Repeated Traversal) NoC architectures are targeted at ensuring near-uniform memory access latencies in CMPs and alleviating the necessity for optimization required to achieve good data-locality and hence, are highly desirable for large-scale Big Data processing. Below, the text from [66] is reproduced:

“This work opens up a plethora of research opportunities in circuits, NoC architectures and many-core architectures to optimize and leverage SMART NoCs. We see SMART paving the way for locality oblivious CMPs, easing the burden on coherence protocol and/or software from optimizing for locality.”

The gains achieved by the SMART NoC is mainly attributed to its control network connectivity. Using an efficient router-bypass mechanism and wires with asynchronous (or) clockless repeaters, SMART NoCs enable single cycle data transfers between nodes that are multiple hops apart in a two-dimensional (2D) mesh NoC (here, only repeaters are clockless
while routers and the processing cores are synchronous). The maximum number of hops that can be traversed in a single clock cycle in a SMART NoC is given as $HPC_{max}$ (Maximum Hops per Cycle).

Despite its benefits, integrating the SMART control mechanism can be undesirable for high performance NoCs. SMART control mechanism restricts the achievable clock frequency of the NoC and involves high overheads. More specifically, with increasing $HPC_{max}$ values, the clock frequency of the NoC decreases linearly [66]. In addition, the router overheads in a SMART NoC grows quadratically with increasing $HPC_{max}$ [68]. Hence, in order achieve a high NoC clock frequency and low overheads, it is necessary to design NoC architectures with low $HPC_{max}$. However, as shown later, designing SMART NoCs with the conventional wireline mesh topology and using a low $HPC_{max}$ limits the achievable performance and scalability.

To reduce the control network overheads, a SMART NoC with reduced wires (RSMART NoC) is proposed [68]. Despite addressing the power, area and connection overheads associated with the original SMART control network, RSMART NoC also experiences the same frequency scalability issues seen in the wireline mesh based SMART NoC. In what follows, we first explain the effect of $HPC_{max}$ values on the clock frequency and the message latency of the wireline SMART NoCs and make a case for integrating on-chip wireless links in SMART NoCs. Next, we demonstrate that by integrating WiNoC and a RSMART control mechanism with low $HPC_{max}$ values, it is possible to design a low-overhead high-performance NoC that is more scalable with increasing system sizes and clock frequencies than the wireline RSMART NoCs.
6.1 Wireless enabled RSMART NoC

In this section, first we outline the motivation for designing a wireless-enabled SMART NoC through the analysis of the data transfer times in the RSMART NoC with varying $HPC_{\text{max}}$. Following this, the interconnection network and the routing protocols associated with the WiSMART NoC are explained in detail.

6.1.1 Architecture of the Conventional RSMART NoC

As explained in Figure 6.1, there are four router pipeline stages in a RSMART NoC; local arbitration, pre-SSR transmission and arbitration, SSR transmission and single-cycle multihop link traversal [68] (at zero loads [108] local arbitration can be avoided to save one cycle). Similar to traditional NoC data transfers, in stage 1, the message competes and wins the local...
arbitration at the source node R0. At stage 2, a single-bit pre-cursor request is forwarded from S0 to S1 and S2, along the pre-SSR wires. In the same stage, this pre-cursor request competes with other received pre-SSRs at S1 and S2. Upon successful pre-SSR arbitrations, a detailed bypass request is forwarded during stage 3 from S0 to S1 and S2 along the SSR wires. Using received SSR, routers R1 and R2 setup the desired bypass connectivity at stage 4, enabling the message to travel straight from port C\textsubscript{in} of R0 to C\textsubscript{out} of R2. Traversing all these four stages is called as traversing a SMART hop. More details on the SMART hop and functionality of RMSART NoC are in [68].

6.1.2 RMSART mesh NoC: \(LPC_{\text{max}}\) Vs Latency

In the RMSART NoC, \(LPC_{\text{max}}\) is defined as the maximum wire length traversed per cycle and is given by Equation (6.1).

\[
LPC_{\text{max}} = HPC_{\text{max}} \times \text{Link length per hop}. \quad (6.1)
\]

Figure 6.2(a) and Figure 6.2(b) show the average SMART hops and the zero-load latency [108] under uniform random traffic for 1D and 2D RMSART mesh NoCs with varying \(LPC_{\text{max}}\) values. We have considered a standard 16×16 mm die with four different system sizes (8×8, 16×16, 24×24 and 32×32). As shown in Figure 6.2(b), for all the considered system sizes, the message transfer latencies monotonically increase with increasing \(LPC_{\text{max}}\). This trend is mainly attributed to the fact that for increasing \(LPC_{\text{max}}\) values the NoC clock frequency decreases linearly, starting from the clock frequency values for \(LPC_{\text{max}}=4\) mm (2.75GHz with 1D and 2.25GHz with 2D following [66]). The reduction in SMART hops achieved by increasing \(LPC_{\text{max}}\) (as shown in Figure 6.2(a)) does not compensate the increase in the
network latency caused by the reduced clock frequency. Similar trends were observed with traffic arising from real applications (shown later in Section 6.2.3) and also with the original SMART NoC proposed in [66]. Hence, from this analysis we can conclude that it is more beneficial to design SMART NoCs with high clock frequencies and low $HPC_{max}$ values than with low clock frequencies and high $HPC_{max}$ values. In addition, we should note that there are a number of previous works employing NoCs with high frequencies, ranging 2.5-3.6GHz [109][110][46]. Thus, the clock frequency of the NoC should only be restricted by the
allowed power budget and the \( HPC_{\text{max}} \) must be determined using the given NoC clock frequency (instead of determining NoC clock frequency restricted by the opted \( HPC_{\text{max}} \) value).

However, as shown in Figure 6.2(a), the data-transfers in SMART/RSMART NoCs designed with low \( HPC_{\text{max}} \) values involve high number of SMART hops when a conventional mesh topology is employed. For such NoCs, we posit that it is beneficial to further enhance the architecture by incorporating additional on-chip wireless communication shortcuts (single-cycle long-range communication links). Next, we explain the design of the WiSMART NoC architecture.

### 6.1.3 Architecture of the WiSMART NoC

As shown in Figure 6.2(b), the network latencies of the 1D and 2D RSMART NoCs are close to each other for small \( HPC_{\text{max}} \) values. However, the 1D RSMART NoCs require much lower connection overheads than the 2D RSMART NoCs [66]. Moreover, the broadcast distribution in SMART NoCs follow a dimension ordered approach where the broadcast message is first distributed along the nodes in a row (or column) and then along all the nodes in parallel columns (or rows) [67]. Such routing only requires a 1D RSMART mechanism. Considering all the above reasons, we employ a 1D RSMART control network for the WiSMART NoC. The WiSMART NoC primarily follows the 1D RSMART control network except for the incorporation of wireless interfaces, a look-ahead pre-SSR enabled routing mechanism and the use two bit pre-SSR wires.

In a WiSMART NoC, few of the routers are provided with a Wireless Interface (WI). For the integration of WIs, we follow a region based WI placement strategy that minimizes both the
number of WIs and the average SMART hop count. Minimizing the number WIs reduces the area and power overheads and also enables fast channel access with low MAC (Medium
Access Control) overhead (explained in Chapter 4). In this region based WI placement strategy, we first divide the NoC into multiple non-overlapping squares and then place a set of WIs in each square (at least one WI for each wireless channel) so that the average hop count is reduced. Since each square requires at least one WI per channel, the number of non-overlapping squares used is limited by the number of WIs allocated to a wireless channel. This in turn depends on the adopted MAC scheme, the distributed MAC protocol. Figure 6.3 shows the placement of WIs in a 64-core WiSMART NoC employing five wireless channels and four squares of size 4×4.

By employing squares of side $HPC_{max}$ and placing WIs in the center of each square, the communication between any two nodes of the NoC can be accomplished within three hops (2 SMART wireline hops and a wireless hop). For larger systems, we further reduce the number of WIs by forming super-squares (64-tile super-square in Figure 6.3(b)). In this strategy, sets of 4-squares are grouped together to form super-squares and for each super-square a maximum of two WIs per channel is used. However, using super-squares increases the average communication hops. For example, in a 576 core NoC made of four super-squares and employing $HPC_{max}=6$ (thus 16 squares of side 6), a maximum of four SMART hops is required to communicate between any two nodes.

For systems exhibiting intense collective communication needs, we propose a wireless-assisted congestion-free broadcast routing in WiSMART NoC (explained later in Section 6.1.5). In such systems, aside from focusing on minimizing the hop counts, the placement of the WIs should also be aligned with the parallel broadcast distribution paths employed. The maximum number of simultaneous broadcast distributions that can be performed in a NoC
is limited by the number of the incoming wireline ports in the router. Thus, using the WiSMART routing, a maximum of four broadcast distributions can be performed in parallel and each of this broadcast is expected to occupy a different wireless channel. Figure 6.3(c) shows an example WI placement that enables simultaneous distribution of four broadcast packets in a system with four super-squares and four wireless channels. Further explanation on the collective communication enabled WiSMART NoC is provided later in the Section 6.1.5.

6.1.4 Unicast Data Transfer in WiSMART NoC

In WiSMART NoC, the unicast data transfers are of two types; short range and long range. The short range WiSMART data-transfers (in which communicating nodes present in the same or adjacent squares) are handled only through the wireline links. For the long-range communications, the message is first forwarded from the source node to a source WI using wireline links. The source WI then broadcasts the message using the wireless channel. Finally, a WI residing in the destination square (or super-square) receives the message and forwards it to the final destination node using wireline links. The wireline transfers in the WiSMART are similar to that of the XY routed 1D RSMART data transfers [68], except for the incorporation of look-ahead request mechanism (Section 6.1.4.2).

6.1.4.1 Selection of WIs for a Communication

In the WiSMART NoC, whenever a source WI becomes unavailable (due to failures or busy wireless channel), the data flits are re-routed from the source WI to the final destination along wireline-only XY paths. Depending on the selection of the source WI, this re-routing mechanism can cause deadlocks in long-range data transfers. For example, if a message transmitted from node 11 to 45 in Figure 6.3(a), selecting node 10, which is the physically
closest WI to node 11 as the source WI can lead to a deadlock. In case of a WI failure at node 10, the message will simply move back and forth between nodes 10 and 11. With this in mind, given a set of source and destination co-ordinates, the rules that must be met by the source WI co-ordinates are given in Table 6.1. Following this table, we can understand that WiSMART routing only allows two of the four possible turns in the NoC and hence avoid creation of cycles (deadlocks). Thus, the source WI is the nearest WI to the source node, satisfying the conditions in Table 6.1. Following these rules, to communicate from 11 to 45, the source WI used is at 13. In addition to the careful selection of source WI, each message is allowed only a single wireless transmission in WiSMART NoC to prevent deadlocks arising from forwarding between different WIs. This condition is ensured by setting a single bit flag in the header of the message, at the source WI prior to the wireless transmission.

For WiSMART unicast communication, the destination WI is the pre-assigned WI that lies physically closest to the destination node and is operating on the same wireless channel as that of the source WI. Thus, the destination WI can reside either in the same square as that of the destination node or in one of the adjacent squares (in cases of systems with super-

### Table 6.1 The rules that must be satisfied by the source WI coordinates to avoid possible deadlocks arising from the re-routing mechanism used for handling WI failures.

<table>
<thead>
<tr>
<th>Given data</th>
<th>Rules for source WI coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_x &lt; D_x$</td>
<td>$S_x \leq W_x &lt; D_x$</td>
</tr>
<tr>
<td>$S_x = D_x$</td>
<td>$W_x = S_x$</td>
</tr>
<tr>
<td>$S_x &gt; D_x$</td>
<td>$D_x &lt; W_x \leq S_x$</td>
</tr>
<tr>
<td>$S_y &lt; D_y$</td>
<td>$S_y \leq W_y &lt; D_y$</td>
</tr>
<tr>
<td>$S_y = D_y$</td>
<td>$W_y = S_y$</td>
</tr>
<tr>
<td>$S_y &gt; D_y$</td>
<td>$D_y &lt; W_y \leq S_y$</td>
</tr>
</tbody>
</table>

(Sx, Sy):Source, (Dx, Dy):Destination, (Wx, Wy): Source-WI
squares). The wirelessly transmitted data are received by the all the WIs operating in the same wireless channel. Upon checking the header, a WI can identify whether it serves as the destination WI for the current communication.

In the WiSMART NoC, similar to the long-range unicasts, the broadcasts are first transmitted to the nearest WIs and transmitted wirelessly. However, unlike unicasts, the broadcast messages are received by the destination WIs (in all regions) that are operating in the same wireless channel as that of the transmitting WI. Finally, the message gets forwarded from each WI to the regional final destination nodes using WiSMART broadcast routing (Section 6.1.5).

6.1.4.2 Transmission of pre-SSRs

In the WiSMART NoC, for each message injected (both unicast and broadcast), first a 2-bit pre-SSR code is identified. In a WiSMART NoC, the unicast wireline data transfers are of two types, a) data transfers from an originating node to a source WI and b) data transfers from an originating node to a final destination node. The data transfers that are used to access the source WIs follow a pre-SSR code of either 10 or 11, depending on the position of the source

![Legend](image)

**Figure 6.4** The pre-SSR values for WIs located in the different positions.
WI (as shown in Figure 6.4). All other wireline data transfers use a pre-SSR code of 01 while 00 indicates a no-request. If the source WI is located in the same row or in the same column as that of the source node, the pre-SSR code used is 10 (indicated by WIs X and Y in Figure 6.4). For output ports with no nearest WIs in the same row, an XY turn is required to reach the source WI (viz., $E_{out}$ of $SR$ in Figure 6.4). In such cases, to avoid the conflicts, data transfers that require an upturn (East $\rightarrow$ North or West $\rightarrow$ North) are indicated through code 10 (accessing WI at $UP$ from $SR$ in Figure 6.4) while those require down turns (East $\rightarrow$ South or West $\rightarrow$ South) are indicated through the pre-SSR code 11 (accessing WI router $DOWN$ from $SR$ in Figure 6.4).

In WiSMART NoC, we employ a look-ahead transmission mechanism for the transfer of pre-SSR signals from a source node to the source WI (thus valued 10 or 11). We use Figure 6.5 to illustrate the transfer of such a pre-SSR signal. In this figure, source node is labeled as $S$, the source WI is labeled as $SWI$ and the $x^{th}$ intermediate router between $S$ and $SWI$ is labeled as $I_x$.

It is important to note that in the WiSMART NoC, along a particular direction, the source router and all intermediate nodes (nodes between source $S$ and turn router $I_3$ in Figure 6.5) follow the same source WI. For example, in Figure 6.5, receiving a value of 11 through the
west pre-SSR wires of any of the three nodes, $S$, $I_1$ and $I_2$ would indicate a request for accessing the same source WI, node $S_{WI}$. Thus, by using the 2-bit pre-SSR data (if it is either 10 or 11) and direction of the node from which the pre-SSR originates, each intermediate node can identify the source WI to which the upcoming data flit is going to be forwarded. Exploiting this information and by using an output port reservation code\(^2\), we can implement look-ahead forwarding mechanism for pre-SSR signals. We further clarify this mechanism through the example shown in Figure 6.5. Here, at cycle $t$, a pre-SSR value of 11 is transmitted from the source node $S$ to all the intermediate nodes that are within the $HPC_{\text{max}}$ hops (nodes $I_1$ and $I_2$). Upon receiving a pre-SSR of value 11 from the east direction, nodes $I_1$ and $I_2$ first identify that the request is to access the WI at router $S_{WI}$. Then, $I_1$ and $I_2$ set the status code of their $W_{\text{out}}$ port representing “bypass $E_{\text{in}} \rightarrow W_{\text{out}}$”. With this status code, the $W_{\text{out}}$ port of routers $I_1$ and $I_2$ are now reserved till the end of data flit transfers, avoiding any local arbitration. In addition, $I_2$ also forwards a look-ahead pre-SSR to node $I_3$, at cycle $t+1$. After receiving the pre-SSR, router $I_3$ sets the status of the $S_{\text{out}}$ as “bypass $E_{\text{in}} \rightarrow S_{\text{out}}$” and forwards the look-ahead

\(^2\) Reservation code has 5 states (3-bits). Each state represents the reservation for a possible bypass connection stemming from the one of the five input ports.
pre-SSR (of value 10) to nodes $I_4$ and $S_{Wh}$ at cycle $t+2$. Finally, $S_{Wh}$ receives the pre-SSR and broadcasts a wireless MAC request at cycle $t+3$, following the distributed MAC protocol. The overall stages in the look-ahead request based routing used to access a WI scheme is shown in Figure 6.6.

It should be noted that the look-ahead request is only applicable when the desired router ports$^3$ are not already reserved and upcoming the SSRs/data flits are not prematurely stopped$^4$. If a desired output port is not available, the look-ahead pre-SSR is avoided and the incoming data flit is buffered and forwarded to local arbitration. In case of prematurely stopped SSRs, reserved output ports are released in the next cycle.

Look-ahead pre-SSRs can also be extended to the unicast wireline data transfers that are not used to access WIs (i.e., data transfers with a pre-SSR of 01). In such cases, since dimension-

<table>
<thead>
<tr>
<th>Field</th>
<th>bit width</th>
</tr>
</thead>
<tbody>
<tr>
<td>No., of Hops in each dimension</td>
<td>$\lceil \log_2 (1 + N) \rceil$</td>
</tr>
<tr>
<td>Is header Flit</td>
<td>1</td>
</tr>
<tr>
<td>Eject flag</td>
<td>1</td>
</tr>
<tr>
<td>Source ID</td>
<td>$\lceil \log_2 (N) \rceil$</td>
</tr>
<tr>
<td>Number of $V_{nets}$</td>
<td>$\lceil \log_2 (VC) \rceil$</td>
</tr>
</tbody>
</table>

$^3$ Desired output ports are present in the turn routers and the routers that are $HPC_{\text{max}}$ apart from the node where the pre-SSR originates.

$^4$ Owing to contention, the SSR and data flits may not travel same number of hops as the pre-SSR and can be stopped and buffered in an intermediate node.
ordered XY routing is followed, we need to know the number of hops required in a dimension before forwarding look-ahead pre-SSR. This hop count information is only carried by the SSR wires and hence, the look-ahead pre-SSR needs to be delayed till the arrival of the SSR data. As shown in Table 6.2, we use \([\log_2(1 + N)]\) SSR wires to indicate the number of hops needed in each dimension and \(\log_2(N)\) wires to indicate the position of the node (along the X or Y dimension) where the SMART hop originates. Hence, upon receiving SSR, each intermediate and turn router can easily identify the number of remaining hops needed in a dimension and forward a pre-SSR.

![Figure 6.7 Transmission of unicast pre-SSRs in the WiSMART NoC.](image-url)
Figure 6.7 illustrates all the steps associated with the WiSMART NoC unicast pre-SSR transmission.

6.1.5 Collective Communication in WiSMART NoC

In this sub-section, first we discuss the wireline SMART NoC broadcast mechanism and its limitations. Next, we explain the architecture of the collective communication aware WiSMART NoC and also comment on the efficiency WiSMART collective communication over the wireline SMART approach.

6.1.5.1 Discussion on Wireline SMART Broadcast distribution

A four-stage private virtual tree (PVT) based broadcast distribution mechanism for wireline SMART NoCs is proposed in [67]. The PVT mechanism is also shown in Figure 6.8(a). During stage 0 of this PVT based broadcast distribution mechanism, a broadcast message is first transferred from the source node (as an example, node 26 is a source node in Figure 6.8(a)) to one of the four corner nodes of the NoC (nodes 0, 7, 56 and 63 are corner nodes in Figure 6.8(a)). In stage 1, the message is passed on from the corner node to all the other nodes in one of the NoC edges (for example, from corner node 0 to all the nodes located in the top edge of the NoC in Figure 6.8(a)). In this stage 1, aside from passing the message to the immediate neighbor on the right, each edge node also retain a copy of the original message (copied to the $S_{out}$ port buffers in Figure 6.8(a)). Then in stage 2, the broadcast message is passed from each of the edge node to all the other nodes located in the same column or row. Finally, during stage 3, all the NoC nodes absorb the received broadcast message. By using the four different corner nodes of a NoC as originating nodes, one can enable a maximum of
four simultaneous broadcast distributions using the PVT mechanism, without any congestion [67].
For a system with \( N \times N \) nodes, by employing a \( HPC_{\text{max}} \) value of \( N \) in a wireline RSM\textsc{art} mesh NoC, one can distribute the broadcast message from the corner node to all the final destination nodes in 2 cycles using the PVT based broadcast distribution mechanism. However, for systems with high NoC clock frequency and \( HPC_{\text{max}} \) value much smaller than \( N \), the PVT broadcast distribution can be inefficient. In such systems, the PVT broadcast mechanism can require multiple clock cycles to complete both stage 1 and stage 2. In addition, the overhead for transferring the broadcast message from the source node to one of the four corner nodes can also be high when \( HPC_{\text{max}} \) value is much smaller than \( N \). On overall, the maximum number of cycles required to perform the broadcast distribution stages 0-2 in a zero-load wireline RSM\textsc{art}/SM\textsc{art} mesh NoC is given by Equation (6.2).

\[
C_{\text{SM\textsc{art}}} = 2 \times \left\lceil \frac{N}{2} \right\rceil^{HPC_{\text{max}}} + 2 \times \left\lceil \frac{N-1}{HPC_{\text{max}}} \right\rceil
\]  

(6.2)

As we can understand from Equation (6.2), both increasing NoC clock frequency (thus decreasing \( HPC_{\text{max}} \)) and system size (\( N \)) will increase the number of cycles required to perform the broadcast distribution, affecting the scalability of the system.

In a SM\textsc{art} NoC, to establish a congestion free broadcast distribution during stages 1 and 2, it must be ensured that there is no contention from (i) other broadcasts under distribution stages 1 and 2, (ii) other broadcasts under source node to corner node transfer, (iii) unicast messages, and (iv) acknowledgement messages. As mentioned above, case (i) congestions are avoided through the selection of the uncongested paths for simultaneous broadcasts that originate from the four different corner nodes in a NoC [67]. In order to avoid the contentions arising from other three cases (ii-iv), one can follow a fixed broadcast distribution interval.
(BI), similar to [67]. In this fixed BI scheme, the wireline links are reserved and available only for stage 1 and stage 2 broadcast distributions at specific cycles. For example, given a BI of 6 and a \( HPC_{\text{max}} \) value of \( N \) in Figure 6.8(a), cycles 0, 6, 12... are reserved for the stage 1 broadcast while cycles 1, 7, 13... are reserved for stage 2 broadcast (accomplished using stage 1 and 2 counters for each router port). During the specified cycles, each router port follows a predefined bypass connectivity that is required perform stage 1 or stage 2 broadcast distribution. Though the fixed broadcast interval mechanism is simple to implement, it can be inefficient under both high-injection and low-injection loads. For systems with heavy injection loads, different broadcasts using the same corner nodes experience additional queueing latency caused by waiting for next available broadcast slot. In addition, under low load scenarios, the wireline reservations unnecessarily wastes the NoC resource that can be used for unicast transmissions.

To overcome the limitations associated with the PVT based broadcast distribution mechanism, here we propose a wireless-assisted region-based WiSMART broadcast distribution.

### 6.1.5.2 WiSMART Broadcast Distribution

Similar to the wireline SMART NoC, WiSMART follows a four-stage broadcast distribution. However, unlike the wireline SMART mechanism that uses corner and edge nodes as starting and intermediate nodes in the broadcast distribution, the WiSMART NoC uses WIs and a set of designated regional nodes as starting and intermediate nodes.

In stage 0 of WiSMART broadcast distribution, the broadcast message is forwarded from the source node to a set of destination WIs. As shown in Figure 6.8(b), the broadcast message is
first forwarded from the source node to the nearest WI which transmits it wirelessly. This message is then received by all the WIs operating in the same wireless channel, ensuring the transfer of the broadcast message from the source node to all the necessary destination WIs.

In Figure 6.8(b) we consider a specific example where broadcast is first sent from source node 27 to WI 18 and then broadcast using wireless channel 4 to WIs 22, 50 and 54. The same method will be repeated for other broadcast messages using different WIs and channels depending on the location of the source node.

The destination WIs then proceed to initiate broadcast distribution stage 1. During this stage, the message is forwarded from each destination WI to a set of predefined intermediate nodes. As seen in Figure 6.8(c), the set of intermediate nodes is distinct for each WI and all the intermediate nodes lie within $HPC_{\text{max}}$ hops from the WI, either along the same row or along the same column. The message is then replicated during stage 2 and forwarded from the intermediate nodes to the final destinations using regional wireline links (Figure 6.8(c)).

Similar to the wireline SMART mechanism, in stage 3 of the WiSMART broadcast, the message is consumed by all the destinations.

As explained in earlier chapters, for cache coherence induced broadcast messages, each destination node generates an acknowledgement (ACK) message that must be sent back to the original broadcast source node. WiSMART NoC employs an ACK aggregation mechanism which aggregates the acknowledgements from the final destination nodes at multiple intermediate nodes and destination WIs, before forwarding the aggregated ACK message to the original source node. The same set of intermediate nodes and destination WIs that initiated the original broadcast distribution are also used to aggregate ACK.
The WiSMART acknowledgement aggregation mechanism uses a set of multi-drop pre-SSR wires and ACK tables, similar to the wireline SMART NoC [67]. ACK tables in each intermediate node keeps a count on the number of ACKs it has to aggregate from each direction. This count gets decremented by each ACK/NACK which is received from the destination nodes through the set of multi-drop wires used to forward pre-SSR. The ACK multi-drop field is of 3 bits wide, one bit for the ACK value (value: 1 for ACK/aggregated ACK, 0 for NACK) and two bits for the ACK ID. The ID value for the ACK is determined by the wireless channel used for the original broadcast message. With the 2-bit ACK ID, four ACKs can be aggregated simultaneously in WiSMART. For each entry in the table, when the remaining ACK count is zero, the intermediate node forwards the aggregated ACK value to the destination WI (which also keeps an ACK table indicating remaining ACK value). Finally, all the destination WIs transmit the aggregated ACK (or a NACK), using an orthogonal on-off keying (OOOK) mechanism [90].

During the cycles free from wireless data transmission, the WIs in WiSMART are used to transfer MAC requests and aggregated ACKs. In such instances, each WI first identifies a 2-bit pattern (00-no request/ACK transmission, 01-MAC request, 1X-ACK transmission where X is the aggregated ACK value). Next, these 2-bits are coded using simple prefixed OOOK technique (results in an 8-bit coded flit for a WiSMART with four WIs in each channel). The OOK coding allows all the WIs in the same channel to transmit the coded request and ACK values simultaneously. The request and ACK values are then received and processed by all the WIs operating in the same channel using a distributed decision-making algorithm (requires 80pS runtime in a system with 4 WIs in each channel).
For MAC requests, all WIs follow a simple fairness based common node selection algorithm explained in Section 3.2.3. In NoCs with ACK transmission, after receiving the aggregated ACK, the broadcast source sends a separate notification message to the destination nodes to free the ACK table entry [67]. In WiSMART, through distributed processing of wireless ACKs, each WI can identify that all ACKs/NACKs associated with a broadcast is aggregated. Hence, unlike the conventional ACK aggregation NoCs, the WIs can generate the notification message simultaneously in all the regions. This notification message can be piggy-backed on the forthcoming broadcast message distribution avoiding additional transmissions [67].

In addition to the region based broadcast distribution and ACK aggregation, WiSMART uses a flexible broadcast interval scheme to avoid the limitations of the fixed BI scheme used in the wireline SMART NoCs. To enable this scheme and also to accommodate ACK aggregation, we use four two bit pre-SSR wires in collective communication enabled WiSMART NoCs. The pre-SSR values and their functions are listed in Table 6.3. As shown, pre-SSRs with initial two bits being ‘00’ are similar to the unicast pre-SSR explained in Section 6.1.4.2. The pre-SSRs carrying ACK values start with 1 in the MSB position while those carrying broadcast requests start with 01. Thus, just by checking the first two bits, the pre-SSR analyzer can easily recognize the type of the pre-SSR and make quick decisions.

6.1.5.3 WiSMART Collective Communication: An Example Scenario

To further clarify the WiSMART collective communication mechanism, we consider an example using Figure 6.8(b). Here, a broadcast message is transferred from a source node
27 to its nearest WI (node 18). This forwarding follows the same router stages as that in Figure 6.6 and uses a pre-SSR value of 0100. After receiving a pre-SSR of 0100, the source WI 18 transmits a broadcast transmission request using the OOK based distributed MAC protocol. Following a successful request, WI node 18 acquires the wireless channel Ch4 and transmits the data flit. The data flit is then received by all the WIs operating on the same wireless channel (nodes 22, 50, 54).

After receiving a broadcast message, each destination WI (nodes 18, 22, 50 and 54) identifies the number of hops in each direction (say $H_x$ hops in direction $x$) up to which the broadcast message has to be forwarded in stage 1. Computation of $H_x$ is accomplished using a set of simple predetermined functions, explained later in Section 6.1.5.3. Next, each destination WI (nodes 22, 50, 54) forwards a pre-SSR of value 0101 to all the intermediate nodes within $H_{PC_{max}}$ hops, along the pre-designated output ports (Ex: from destination WI node 50 to intermediate nodes 48, 49 and 51 in Figure 6.8(c)). The pre-SSR transmission is followed first by an SSR and then by the data flits (similar to Figure 6.6). As given in Table 6.2, the SSR data in WiSMART consists both the number of hops requested in direction (equivalent to $H_x$)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>No request</td>
</tr>
<tr>
<td>0001</td>
<td>Regional Unicast</td>
</tr>
<tr>
<td>0010 (or) 0011</td>
<td>Wireless enabled Unicast</td>
</tr>
<tr>
<td>0100</td>
<td>Broadcast Stage-0</td>
</tr>
<tr>
<td>0101</td>
<td>Broadcast stage-1</td>
</tr>
<tr>
<td>0110</td>
<td>Broadcast Stage-2</td>
</tr>
<tr>
<td>1xyz</td>
<td>ACK transmission, x gives ACK value, y and z give ACK ID</td>
</tr>
</tbody>
</table>
and the position \((P_w)\) of the destination WI which initiated the pre-SSR request. The \(P_w\) is either the X or Y dimension coordinate, depending on SSR direction. Using SSR data, any node \(i\) satisfying the following condition can identify itself as an intermediate node for stage 2.

\[
|H_x - P_w| \leq |H_x - P_i| \quad (6.3)
\]

In above equation, \(P_i\) represents the position of node \(i\).

After receiving the SSR from the destination WIs, the intermediate nodes identify themselves using Equation (6.3) and initiate a look-ahead pre-SSR transmission of 0110 to all the final destination nodes (Ex. from intermediate node 49 to final destination nodes 33, 41 and 57 in Figure 6.8(d)). In addition, each intermediate node also reserves appropriate output ports during the following cycles for SSR and broadcast message transmissions. Any broadcast reservation overrides all the unicast reservations. Thus, when affected by broadcast reservations the incoming unicast message will be pre-maturely stopped at an intermediate node and undergoes local arbitration again.

As stated earlier, for cache coherence induced broadcasts, each destination node generates an ACK/NACK message. These ACK messages are aggregated first at the intermediate nodes and then at WIs before forwarded back to the original broadcast source. For the example shown in Figure 6.8(b), after aggregating regional ACKS, WIs 22, 50, 54 transmit the ACK (or a NACK) using the wireless channel 4. These wireless ACKs are aggregated by node 18 and gets forwarded to node 27 to complete the collective communication.

The WiSMART broadcast distribution has a few salient features that can be identified from the following explanation and using Figure 6.8. First, WiSMART accelerates the wireline
broadcast distribution by using multiple parallel regions (each regional distribution originates from a different destination WI). Second, the transmission from a source node to a source region WI in WISMART NoC requires lesser hops than the transmission from the source node to the corner node in a RSMART mesh NoC. For example, in Figure 6.8(b), the nearest source WI for any source router is within $H_{PC_{max}}$ hops. The maximum number of cycles required to perform broadcast distribution stages 0-2 in a zero-load WiSMART NoC of size $N \times N$ nodes is given by Equation (6.4). In this equation $C_{MAC}$ represents the MAC latency experienced by the message.

$$C_{WiSMART} = C_{MAC} + 2 \times \left\lceil \frac{N}{2+R} \right\rceil - 1 + 2 \times \left\lceil \frac{N}{R} \right\rceil - 1$$

(6.4)

By comparing Equations, (6.2) and (6.4), one can conclude that the WiSMART NoC reduces the wireline broadcast distribution latency by a factor of $R$ when compared to the RSMART NoC. Thus, in general, increasing the number of regions ($R$) increases the WiSMART NoC efficiency. However, as noted in Section 6.1, there is an upper bound of the value of $R$, determined by the selected MAC protocol and allowed WI area overhead.

The third and the final salient feature in the WiSMART broadcast distribution is the use of uncongested wireline paths. As seen from Figure 6.8(c-d), for stages 1 and 2, the broadcast messages originating from different wireless channels use distinct sets of wireline links. Such path selection avoids the wireline contentions among the different broadcasts that are distributed simultaneously and eliminates congestion. The number of simultaneous broadcast distributions that can happen in a WiSMART NoC is limited by the number of wireline ports in a router. As shown in Figure 6.8(c-d), up to four broadcasts (each occupying
a different wireless channel) can be distributed simultaneously in the WiSMART NoC without any contention.

6.1.5.4 Design of WiSMART NoC for Collective Communication

To design a collective communication enabled WiSMART NoC, we need to identify: i) the optimal location of the WIs; ii) the uncongested broadcast distribution paths and iii) the number of hops up to which the message has to be forwarded from each output port, during stages 1 and 2.

We employ a simulated annealing (SA) based mechanism to identify both the position of the wireless interfaces and the uncongested wireline distribution paths in the WiSMART NoC. The proposed mechanism is shown in Figure 6.9. The SA based mechanism requires the backbone wireline topology, the number of wireless channels used for collective

![Figure 6.9 Collective Communication enabled WiSMART NoC Creation.](image-url)
communication ($N_{ch}$) and the maximum number of WIs allowed in a channel ($M_{ch}$), as inputs. In this mechanism, a new WiSMART NoC candidate is created for each optimization cycle using SA-based perturbations. For each iteration, the set of wireless channels that use stage 1 intermediate nodes in the same row (X-first) and the wireless channels that use stage 1 intermediate nodes along the same column (Y-first) are identified first. Next, the locations for the W are identified. In this step, we ensure that the vertical and horizontal spacing between the same-channel WIs are uniform. Following the placement of WIs, the stage 1 and stage 2 broadcast paths and the maximum hop count to the nearest WI ($H_{WI}$) are identified to evaluate the quality of the current WiSMART design.

In the WiSMART broadcast distribution, it is necessary to identify the nodes to which the message has to be forwarded from the destination WIs and intermediate nodes during stages 1 and 2. For this purpose, at each output port $X$, for each wireless channel $i$, we use a simple function $H^{i}_{ch}(X)$ to identify the number of hops up to which the message has to be forwarded. A $H^{i}_{ch}(X)$ function is constituted using the ID of the wireless channel carrying current broadcast (found using the value and the direction of pre-SSR, explained in III.B.2) and column and row IDs (given by $C_{ID}$, $R_{ID}$) of the intermediate node itself (in a 64-core NoC, $C_{ID}$ and $R_{ID}$ are 6 bits each).

As mentioned above, we ensure regularly spaced WI locations in the construction of the WiSMART NoC. Hence the set of $H^{i}_{ch}(X)$ functions remain the same for all routers. For example, in Figure 6.8(c-d), after receiving a broadcast message that uses channel 2, all intermediate nodes identify the number of hops ($H^{2}_{ch}$) up to which the message has to be forwarded along each output port ($E$, $W$, $S$, $N$) using the following functions:
\[ H^2_{ch}(E) = 0; H^2_{ch}(W) = 0; H^2_{ch}(S) = 2; \]

\[ H^2_{ch}(N) = \begin{cases} 0, & \text{if } R_{ID}[5] = 1 \\ 1, & \text{otherwise} \end{cases} \quad (6.5) \]

As another example, we can consider the intermediate routers designated for channel 3 in Figure 6.8(c-d).

\[ H^3_{ch}(E) = 1; H^3_{ch}(W) = 2; H^3_{ch}(S) = 0, H^3_{ch}(N) = 0; \quad (6.6) \]

We can note that the use of the \( H_{ch}(X) \) functions and the pre-SSR values (in Table 6.3) restrict the set of nodes to which a broadcast message can be forwarded from the output port of an intermediate node. Aside from eliminating congestion, this restriction also ensures that the WiSMART wireline broadcast distribution is free of deadlocks. To handle the WI failures in a collective-communication-aware NoC, the deadlock-avoidance rules in Table 6.1 cannot be applied since the number of destinations in a broadcast message is greater than one. Hence, we use the dynamic quick reconfiguration (DQR) technique \cite{111} to prevent the occurrence of deadlocks caused by WI failures. In this situation, the source WI becomes the new source for the broadcast packet, which is then distributed using the shared virtual tree wireline paths \cite{67}.

### 6.1.6 Handling pre-SSR Contention in WiSMART NoC

To handle the contentions arising from different pre-SSRs requesting the same output port, the following priority is used:

*Broadcast Reservation > Unicast Reservation > Wireless pre-SSRs (10/11)*

*Wireline-only pre-SSR (01)*
Within the wireless and wireline-only pre-SSRs, we use \( \text{"prio=local"} \) mechanism [66] to handle contentions in which the requests from the physically closer nodes are preferred over the relatively far nodes. Finally, if pre-SSRs coming from multiple directions request the same WI port in a router, we use the following priority rule to access the WI port:

\[
\text{Reservation Port} > C_{in} > E_{in} > W_{in} > N_{in} > S_{in}
\]

Next, we incorporate the discussed routing in the WiSMART architecture and undertake a detailed performance evaluation.

6.2 Performance Evaluation of WiSMART NoC

In this section, we evaluate the performance of the WiSMART NoC by comparing it with other state-of-the-art NoC architectures. Considering the broad applicability of the WiSMART NoC, we employ the set of ten applications introduced in Section 3.4 for this performance evaluation.

We consider a 16×16mm die with 64, 256, 576 and 1024 cores, using both Directory and Hammer coherence protocols. For full system evaluations, we use GEM5 full system simulator with a memory system comprised of private 64KB L1 instruction and data caches, one shared 16MB L2 cache (256KB distributed L2 per core) and running sixty-four x86 cores. The interconnects considered here are modeled through the GARNET and RUBY modules within Gem5.

6.2.1 Scalability of the WiSMART NoC

In this section, we analyze the scalability of the WiSMART NoC with increasing operating frequencies and system sizes. Figure 6.10 compares the zero-load latency for uniform
random (UR) unicast and broadcast traffic in a 256-core system incorporating 1D RSMART and WiSMART NoCs, operating at different clock frequencies. The WiSMART NoC consists of four 64-core super-squares. For increasing clock frequencies, the $HPC_{\text{max}}$ values of both the NoCs decrease linearly, starting from $HPC_{\text{max}}=8$ for a clock frequency of 1.375GHz.

For lower $HPC_{\text{max}}$ values (i.e., high frequencies), the data exchanges in a RSMART mesh NoC involve high number of SMART hops (Figure 6.2(a)) and hence, exhibit high network latencies. On the other hand, the WiSMART NoC provides single cycle communication links between physically distant nodes, even with high clock frequencies. Thus, for increasing clock frequencies, the WiSMART NoC achieves increasingly higher gains, when compared to the RSMART mesh NoC. More specifically, WiSMART NoC provides 35% and 45% reduction in unicast communication latency when compared to the 1D RSMART mesh NoC with 2.75 and 3.667 GHz clock frequencies, respectively. For the broadcast distribution, WiSMART NoC uses four smaller regions where the broadcast distributions are happening in parallel. Thus, following Equations (6.2) and (6.4), WiSMART can achieve up to 4× reductions in wireline

---

**Figure 6.10** Zero load latency of 256-core SMART NoCs for different clock frequencies with UR unicast and broadcast traffic. The shown frequency values (1.375GHz to 5.5 GHz), are simply the maximum possible frequency values for $HPC_{\text{max}}$ varying from 8 to 2. As mentioned in Section 6.1, the true operating frequency of the NoC depends on the allowed power budget.
transfer latencies associated with the broadcast, when compared to RSMART mesh NoC. On Overall, with 2.75 and 3.667 GHz clock frequencies, the WiSMART NoC achieves 53% and 62% lower broadcast distribution latency compared to the RMASRT mesh. It should be noted that the gains demonstrated in Figure 6.10 are the maximum possible gains achieved by employing any low-latency interconnect technique (on-chip wireless or surface-wave or optical links) that achieves single-hop long-range communication along with the SMART NoC. The true gain values will ultimately depend on the achieved data-rate.

The WiSMART NoC latency savings explained above is also observed for different system sizes. This statement is corroborated through Figure 6.11, which compares the zero-load broadcast distribution and unicast communication latency under uniform random traffic for four different system sizes incorporating 1D RSMART and WiSMART NoCs. Figure 6.11 also shows the variation in the NoC performances with varying SMART hop latencies, considering both 3 and 4 cycles per SMART hop. It is important to note that with different NoC frequencies, the number of clock cycles required to implement all the four stages of a SMART
hop may vary (higher the frequency, higher the number of cycles required to implement the router stages in a SMART hop). As seen from Figure 6.11, the gain achieved by using the WiSMART increases with increasing router-stage latency. This further demonstrates the usefulness of the WiSMART mechanism in designing NoCs that operate with high clock frequencies.

### 6.2.2 Gains induced by Look-Ahead pre-SSRs and WIs

In Figure 6.12, we compare the zero-load latency (UR traffic) of a 256-core 1D RSMART, 1D RSMART incorporating look-ahead pre-SSR mechanism (1D RSMART-LA) and WiSMART NoCs. As seen, the look-ahead pre-SSR mechanism notably accelerates the data transfers in 1D RSMART NoCs designed with small $HPC_{max}$, resulting 35-20% latency savings for $HPC_{max}$ ranging from 2-5. For higher $HPC_{max}$, the physical distance over which a message can be transferred in each SMART hop also increases. Hence, with growing $HPC_{max}$ values, the number of data transfers that require look-ahead pre-SSR reduces, leading to diminishing returns.

![Figure 6.12 Comparing the zero-load latency of 256-core 1D RSMART, 1D RSMART with look-ahead pre-SSR and WiSMART NoCs.](image-url)
By comparing the latency values of the three different NoCs in Figure 6.12, we can understand that the main reason for the WiSMART NoC performance gains under unicast and broadcast vary depending on the nature of the communication. For unicast messages, both the incorporation of the WIs and the look-ahead pre-SSR mechanism provides significant gains while in case of broadcast messages, the WiSMART NoC gains are mainly derived from the wireless assisted region based data transfers. More precisely, with $HPC_{\text{max}} = 4$, WiSMART NoC achieves 56% gain in broadcast latency when compared to the 1D RDMART NoC while the 1D RSMART-LA achieves only a gain of 9%. This is because, 1D RSMART-LA NoC only accelerates source to corner nodes data transfers from in PVT broadcast mechanism. The WiSMART NoC on the other hand, accelerates all the first three stages in broadcast distribution using the wireless-assisted region-based communication.

### 6.2.3 Evaluation of WiSMART NoC with Real Application Traffic Patterns

In this section, we compare the flit latencies of WiSMART NoC with two wireline-only and two wireless enabled state-of-the-art NoCs, for running the above-mentioned set of 10 real applications. Specifically, we consider a) the RSMART mesh NoC, b) the low hop-count high radix NoC with flattened butterfly topology c) a small-world based multicast-aware wireless NoC (WiNoC) discussed in Chapter 5 and d) Wireless mesh (WiMesh).

The WiMesh NoC is constructed by placing wireless interfaces on the traditional wireline mesh architecture. WiMesh is similar to WiNoC except for the regular wireline mesh connectivity (illustrated in Figure 6.13). We follow the same region based wireless node placement that we explained in Section 5.1.1 for this WiMesh. In WiMesh, all the intra-region traffic is handled by the traditional XY routing. For inter-region multicast data exchange, the
packets are first forwarded to the nearest WIs using XY routing and are broadcast using the wireless channel. WIs residing in the non-source regions receive this broadcast message and distribute it to the intended regional destinations using the wireline links using XY-Tree multicast routing.

The HR-FB NoC considered here operates at 3GHz and uses 4-stage routers and pipelined long-range wires (up to 3 segments) [46]. This HR-FB NoC uses dimension ordered XY routing to transfer unicasts and the Tree multicast mechanism [14] to distribute the broadcasts. To provide a fair comparison with this HR-FB NoC, we use \( LPC_{\text{max}}=4\text{mm} \) (hence a clock frequency of 2.75GHz) for the WiSMART NoC (WiMesh and WiNoC also operate with 2.75GHz clock frequency). We consider 2 different 1D RSMART mesh NoCs with \( LPC_{\text{max}} \) values of \( 8\text{mm} \) (1.375GHz clock) and \( 4\text{mm} \) (2.75GHz clock), denoted as RSMART8 and RSMART4, respectively.
In Figure 6.14(a) and Figure 6.14(b) we compare the flit latencies of the different NoCs with Directory and Hammer cache coherence protocols, respectively. As observed from Figure 6.14, RSMART8 exhibits the highest network latency, mainly due to its low NoC clock frequency. When compared to RSMART8, the RSMART4 mesh NoC achieved about 33% reduction in the flit latency for both the collective communication heavy Hammer and the unicast heavy Directory protocols. This once again demonstrates that using a NoC with a high operating frequency and a small $HPC_{\text{max}}$ is more beneficial than using a high $HPC_{\text{max}}$ NoC that is operating with lower clock frequency.

![Figure 6.14 Average flit latencies for the RSMART, WISMART and High Radix NoC s.](image)

(a) Flit latencies for Directory Coherence Protocol

(b) Flit latencies for Hammer Coherence Protocol
In WiSMART and RSMART NoCs, the data flits bypass the router stages at intermediate and destination nodes. However, in HR-FB, the data flit traverses all stages in a router (on average 2.6 router traversals). Also, HR-FB uses latency-expensive routers and links. Using latency-efficient wireless links, WiSMART NoC outperforms both the RSMART4 (on average 3.17 SMART hops) and HR-FB NoCs.

Comparing WiSMART, WiMesh and WiNoC, one can understand the advantages of using the WiSMART control mechanism. Unlike WiSMART, WiMesh and WiNoC lack SSR network and hence cannot establish bypass paths to avoid intermediate nodes. In fact, WiMesh requires more than seven hops for distributing broadcasts and each hop involves three router stages. Hence, despite having identical data transfer network and WI placements, the WiMesh NoC exhibits more than 50% higher flit latency than the WiSMART NoC. The small-world based WiNoC incorporates additional wireline shortcuts and performs better than the WiMesh. However, in WiNoC there are only a limited number of shortcuts which are determined at design time. On the other hand, WiSMART NoC can establish traffic-dependent dynamic bypass paths leading to efficient data transfer times. Considering all the applications with both Hammer and Directory protocols, WiSMART achieves 60%, 52%, 47.5%, 34% and 41.5% savings in average flit latency compared to RSMART8, WiMesh, HR-FB, WiNoC and RSMART4 respectively (Figure 6.14).

As shown in Figure 6.15(a), among the considered applications, CNL, RAD, FFT, LU, GP and KM exhibit high fractions of long range traffic injections under the Directory coherence protocol. The Hence, these applications benefit more from the use of WiSMART NoC than other applications (Figure 6.14(a)). LR and PCA exhibit mainly short range traffic and hence
achieve relatively low latency improvements by employing the WiSMART NoC (26% gain when compared to the RSMART4). Moreover, the WIs in the WiSMART NoC also help in efficiently distributing the hotspot-heavy skewed traffic patterns exhibited in KM, WTR and CNL. In these applications, a few hot-spot cores inject high volumes of traffic, causing network congestion, as shown in Figure 6.15(b). Unlike the wireline-only NoCs that use wireline links for all data transfers, the WiSMART NoC routes a certain fraction of traffic (long-range traffic) using wireless links. This leads to the alleviation of wireline bandwidth bottlenecks and improved performances.

Finally, the effectiveness of the wireless-enabled region-based collective communication used in WiSMART NoC can be understood from Figure 6.14(b). By quickly distributing
broadcasts through its multiple parallel regions, WiSMART achieves about 45% latency gain with the broadcast-heavy Hammer WiSMART NoC when compared to RSMART4 NoC.

### 6.2.4 Full System Energy Delay Product

In this section, among the six on-chip networks considered earlier, we evaluate the full system energy delay product (EDP) of the best performing wireless-enabled and wireline-only NoCs (viz., WiSMART and RSMART4). For this EDP computation, we use the application runtimes and the total energy consumed by the computing cores and the network components. The core-level statistics generated by the GEM5 simulations are incorporated into McPAT (Multicore Power, Area, and Timing) to determine the core power. The energy consumption of the network components is computed following [66].

Comparing the power consumption profile of the two NoCs, some of the WiSMART NoC routers are found to be dissipating more power than 1D RSMART NoC routers, mainly due to the presence of WIs. However, the share of the power consumed by a WI (which is in order of mW) in the total system power dissipation (in order of tens of Watts) is usually negligible. Moreover, by reducing the message transfer latencies, the WiSMART NoC lowers the execution times of the applications and ultimately leads to lower full system energy consumptions. This statement is corroborated through Figure 6.16, which compares the full system EDPs of the RSMART4 and WiSMART NoC-based multicore systems for the ten applications. Among the considered applications, GP and CNL exhibit the two highest flit-intensity values (average number of flits per instruction) while FFT and LR have the two lowest flit-intensity values for both Directory and Hammer coherence protocols. Hence, CNL and GP benefit most from the better NoC architecture and achieve high EDP gains (Figure
With the Directory coherence protocol, GP and CNL achieve 27% and 22% gain in full system EDP with the WiSMART NoC when compared to the RSMART NoC respectively. For Directory coherence protocol, despite achieving high latency gains with WiSMART (shown in Figure 6.14), FFT and LR achieve only 6% and 7% full system EDP savings due to their low flit-intensity. For the all the considered applications, the WiSMART NoC achieves an average of 16% of EDP savings when compared to the RSMART NoC, under Directory cache coherence protocol.
In general, the latency gains achieved by the WiSMART NoC is higher with the collective communication messages than for the unicasts (as seen from Figure 6.12(a) and (b)). All broadcast distributions and ACK aggregations require long-range data transfers and are highly accelerated using multiple parallel regions in the WiSMART NoC. Since, Hammer induces more collective communication traffic than Directory, it also exhibits higher full-system EDP gains (Figure 6.16). Depending on the on-chip traffic intensity of the applications considered, WiSMART NoC achieves 12-43% full system EDP gains when compared to the RSMART NoC incorporating the Hammer cache coherence protocol (Figure 6.16(b)).

6.2.5 Comparison of Router Overheads

As explained in [68], the router overheads associated with a SMART NoC is mainly dependent on the number of control network connections and the $HPC_{\text{max}}$. Hence, the worst-case control overheads for the wireline unicast SMART and RSMART NoCs were analyzed in [68]. In this work, first we improve on this analysis by adding the connection overheads associated with the collective communication and WiSMART NoC. Moreover, we use the actual number of connections instead of using worst case values for the overhead factor computation. Figure 6.17(a) compares the connection overheads of a 1024 core SMART and RSMART NoCs with both 2D and 1D bypass request networks. As it can be seen from this figure, both the 2D bypass request mechanism and the original SMART NoC implementation (using multi-drop wires for all control signals) lead to high connection overheads. Thus, among all the control network options, it is more desirable to select to 1D RSMART control network as the router-bypass setup network for our WiSMART NoC. The total number of overhead connections in a collective communication aware WiSMART NoC ($OV_{\text{WiSMART}}$) is given by the sum of all the
multi-drop pre-SSR connections, SSR mesh network connections and all the connection between the SSR router and the data flit router. Hence, the $OV_{WiSMART}$ value is given by Equation (6.7) where $W$ represents the bit-width of the wire and the parameter $N$ is the sum of all the overhead connections computed from the given NoC topology.

$$OV_{WiSMART} = W_{pre-SSR} \times N_{pre-SSR} + W_{SSR} \times N_{SSR} + W_{SSR}$$

$$+ N_{SSR-data_router}$$

(6.7)

WiSMART NoC uses a $W_{pre-SSR}$ value of 4 (Table 6.3). The SSR wire width can be calculated using Table 6.2. As shown in Table 6.2, WiSMART NoC uses $\lceil \log_2(1 + N) \rceil$ SSR wires to indicate the number of hops and $\log_2(N)$ wires to indicate the pre-SSR source node location, necessary to enable look-ahead pre-SSRs. Figure 6.17(b) shows a zoomed version of Figure 6.17(a) comparing the overheads of 1D SMART, 1D RSMART and WiSMART NoCs. As it can be seen from this figure, for low $HPC_{max}$ values, WiSMART NoC has a slightly higher overhead than the 1D RSMART mesh due to the variation in the SSR wire width. For example, for a 1024 core NoC with a $HPC_{max}$ value of 8, WiSMART NoC requires 11% more control overheads than the RSAMRT NoC. Similar to the pre-SSR of WiSMART NoC, the baseline 1D RSMART NoC also requires four wires per multi-drop connection (1 bit pre-SSR and 3 bits for ACK aggregation network). However, unlike WiSAMRT, the RSMART mesh NoC only uses $\lceil \log_2(1 + HPC_{max}) \rceil$ SSR wires to indicate the number of hops and $\log_2(HPC_{max})$ to indicate the SSR-source locations [68] and hence require smaller SSR widths than WiSMART NoC.

The true silicon area overheads associated with a RSMART router depends on the actual wiring [68]. However, compared to a router in a traditional NoC without any control network,
the 1D RSMART control network induced silicon area overheads in a 1024-core WiSMART NoC (with four regions and a $HPC_{max}$ of 8) can be kept within 10% [68].

In addition to the control network, the WiSMART NoC also adds area overhead caused by the WIs. For the WI design considered [57], the transmitter occupies 200µm×460µm and the receiver occupies 270µm×15µm area. Thus, considering a 2mm×2mm NoC tile, addition of a WI causes an area overhead of 5.78%. Despite this added cost, as shown throughout the paper, the WiSMART NoC clearly outperforms the RSMART NoC in terms of network latency.
(about 40% gain with 2.75GHz clock). Hence, WiSMART NoC is more beneficial than RSMART NoC for high-performance manycore platforms despite the small addition in the overheads.

### 6.3 Summary

There are a number of applications that exhibit highly challenging on-chip traffic patterns in manycore platforms. For example, AMD’s Hammer based cache coherence mechanism generates high volumes of collective communication while graph analytics applications inject heavy long-range traffic that require multiple hops for distribution. In order to efficiently execute such applications with challenging communication requirements, it is necessary to integrate a low latency on-chip data transfer network. Towards this, we propose a wireless communication enabled SMART NoC (WiSMART) that is incorporated with a novel look-ahead request based routing mechanism. The WiSMART NoC also integrates a region based congestion-free collective communication mechanism to address the data transfer requirements of the modern cache coherence protocols.
7 Future Research Directions

As elaborated throughout this thesis, performing Big Data processing on manycore platforms necessitates specialized NoC architectures capable of handling complex memory access patterns. Towards this, this thesis proposed a solution targeting the acceleration of on-chip memory access through low-latency NoC architectures employing on-chip wireless links. In addition to using specialized WiNoC architectures, the efficiency of Big Data computing on manycore platforms can be further enhanced by improving both software and hardware architectures. Here we briefly introduce a few such techniques that can explored in future.

7.1 Hardware-Software Co-design Addressing Memory Access Complexities

To scale an advanced Big Data operation such as graph community detection to large networks, it is necessary to design architectures that address the complex on-chip memory access patterns associated with it. More precisely, given that irregular data movement is the primary limiting factor in the performance of graph and MapReduce applications, we posit that reducing the cost and/or the volume of data movement will be critical in scaling up graph and MapReduce applications. Furthermore, in scaling up graph and MapReduce applications, the cost of energy consumption cannot be ignored, and exploring ways to
reduce data movement and making it more effective will impact the energy cost of such applications.

As a preliminary work, we explored the effects of employing approximate updates in Grappolo. More specifically, we focus on evaluating the full-system benefits attained through the new software implementation. Next, we present a tutorial on the software implementation considered, followed by the full system evaluations.

### 7.1.1 Graph Community Detection with Approximate Updates

The community detection application implemented here uses full-synchronization technique along with approximate updates. First, we explain the operations associated with the community detection application employing full-synchronization technique (called as Grappolo FS version).

Given a graph $G(V,E)$, with $n$ vertices and $m$ edges, Grappolo iteratively tries to improve the overall modularity by running multiple “phases”, and multiple iterations within each phase, until modularity gain becomes negligible. The steps of a phase are:

i) Initially, each vertex is assigned to a community of its own.

ii) Within each iteration the vertices are scanned in parallel and for each vertex, a decision is made to determine whether or not to migrate it to one of its neighboring communities (as defined by the communities of its neighbors). For executing this step, the latest community information from all neighboring vertices are pulled by a given vertex. Locking is used to prevent concurrent and potentially conflicting updates from happening at the neighbors. In this model, locking needs to happen at two data
structures – one at the level of a neighboring vertex and another at the level of the community that holds that vertex. This software version is labelled “fully synchronized update” version.

iii) The algorithm proceeds to subsequent iterations until the gain achieved in modularity becomes negligible. Reaching convergence by this criterion marks the end of current phase and the algorithm constructs a compacted graph $G'(V',E')$ by collapsing every community detected in $G$ into a single meta-vertex in $G'$, and creating edges with weights corresponding to intra- and inter-community links in $G$.

iv) Subsequently, the algorithm initiates the next phase on the newly compacted graph $G'$, until no more appreciable modularity gain is achieved (i.e., convergence).

One of the major contributors to the computational cost of the above algorithm is its local neighborhood querying in step (ii). The fully synchronized update model ensures that the most recent community information of a vertex is made available to any of its vertex neighbors for its migration decision. However, the cost of such full synchronization manifests itself in a pattern of large volume and potentially irregular data traffic. Therefore, this fully synchronized version is expected to yield a fast convergence (i.e., less number of iterations due to latest information availability) but at the possible expense of more time cost per iteration.

7.1.1.1 Early Termination of Vertices

As explained above, within each iteration of Grappolo algorithm using the fully synchronized update model, the computation time is dominated by the time taken to decide on a vertex migration. Here we use a more efficient implementation of Grappolo that avoids precise
updates. More specifically, unlike classical heuristics that mainly focus on reducing the runtime, our proposed scheme focuses on reducing the intensity of data lookups, thereby generating a potential to reduce both time and energy caused by the data movements. The main idea hinges on the key observation of a diminishing returns property in quality as the iterations progress [38]. As shown by the results on real-world graphs (Figure 7.1), the gain in modularity plateaus after a few initial iterations, because of significantly fewer community updates for vertices during the later iterations. We take advantage of this observation and include a counter, \( n_{\text{elapsed}} \), at every vertex to denote the number of iterations that have

![Figure 7.1](https://example.com/figure71.png)

*Figure 7.1 Effect of early termination on the convergence rate of modularity, and the number of vertices processed per iteration. Iterations continue until the attained modularity saturates up to the third decimal place.*
elapsed since its last successful migration (i.e., change in community). If the counter $n_{Elapsed}$ exceeds a certain threshold $\tau$, then we “terminate” that vertex—i.e., we (optimistically) stop considering that vertex during any subsequent iteration of that phase implying that the vertex is locked into that community. This heuristic has two performance advantages: a) it enables a faster convergence of modularity by reducing both the number of vertices that need to be processed at every iteration and the total number of iterations required, and b) it also reduces the volume of memory lookups generated from each vertex because terminated vertices will stop seeking information from their neighborhoods. However, this also runs the risk of possibly degrading the final quality achieved (measured by modularity).

For our evaluating the sue of approximate updates, we employ $\tau=3$. First, we compare the convergence rates (measured in the number of iterations) of the fully synchronized (“FS”) and the fully synchronized with early termination (“FS+ET”) versions of Grappolo, for the ASTRO and COND input graphs in Figure 7.1(a) and Figure 7.1(b), respectively. As it can be seen, the number of iterations reduces from 28 iterations in FS to 20 iterations in FS+ET implementation for ASTRO. COND also follows a similar trend (Figure 7.1(b)). The faster

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Vertex Count</th>
<th>Edge Count</th>
<th>Execution Time (sec)</th>
<th>Modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FS</td>
<td>FS+ET</td>
<td>Gain</td>
<td>FS</td>
</tr>
<tr>
<td>ASJ</td>
<td>22,963</td>
<td>48,436</td>
<td>0.7608</td>
<td>0.3437</td>
</tr>
<tr>
<td>ASTRO</td>
<td>16,706</td>
<td>121,251</td>
<td>0.2998</td>
<td>0.1854</td>
</tr>
<tr>
<td>COND</td>
<td>31,163</td>
<td>120,029</td>
<td>0.3174</td>
<td>0.11961</td>
</tr>
<tr>
<td>HEP</td>
<td>8,361</td>
<td>15,751</td>
<td>0.0535</td>
<td>0.0343</td>
</tr>
<tr>
<td>PGP</td>
<td>10,680</td>
<td>24,316</td>
<td>0.0829</td>
<td>0.0623</td>
</tr>
</tbody>
</table>

Table 7.1 Evaluation of the approximate update scheme (via ET) on community detection’s runtime and modularity. All runs were performed on a 32-core Intel multicore platform.
convergence in the FS+ET version is caused by the early-termination of non-migrating vertices, which results in less changes to accumulate as iterations progress and subsequently reach peak modularity in reduced number of iterations. Aside from the lowered number of iterations, it should also be noted that with the early termination, the execution time per iteration significantly reduces as the algorithm progresses. As the iterations grow, more vertices settle in their communities, allowing a large number of vertices to be terminated within each iteration. This statement is corroborated by Figure 7.1(a) and Figure 7.1(b) where the active number of vertices reduces with the increase in number of iterations. The expected reduction in overall runtime is confirmed in the results of Table 7.1. Notably, this table shows that the runtime savings are achieved without compromising on the output quality (modularity). Table 7.2 demonstrates the reductions achieved by the use of early termination in the number of edges traversed and the number of community lookups, both of which are also the dominant contributors toward memory traffic.

<table>
<thead>
<tr>
<th>Graphs</th>
<th>NET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FS</td>
</tr>
<tr>
<td>ASJ</td>
<td>871,848</td>
</tr>
<tr>
<td>ASTRO</td>
<td>2,667,522</td>
</tr>
<tr>
<td>COND</td>
<td>3,600,870</td>
</tr>
<tr>
<td>HEP</td>
<td>283,518</td>
</tr>
<tr>
<td>PGP</td>
<td>437,688</td>
</tr>
</tbody>
</table>

**Table 7.2 Evaluation of the reduction in number of edges traversals (NET) and number of community lookups (NCL), induced by ET.**

<table>
<thead>
<tr>
<th></th>
<th>NCL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FS</td>
</tr>
<tr>
<td>ASJ</td>
<td>603,953</td>
</tr>
<tr>
<td>ASTRO</td>
<td>850,865</td>
</tr>
<tr>
<td>COND</td>
<td>1,601,311</td>
</tr>
<tr>
<td>HEP</td>
<td>170,780</td>
</tr>
<tr>
<td>PGP</td>
<td>224,060</td>
</tr>
</tbody>
</table>
7.1.2 Full System Evaluations

To evaluate the effectiveness approximate updates, we compare execution times and manycore system energy dissipation for FS and FS+ET versions of Grappolo. To ensure a detailed evaluation, we consider WiNoC (explained in Chapter 5) and mesh interconnection architectures operating with both Hammer and Directory coherence protocols. The experimental setup is similar to the one explained in Section 5.x.

7.1.2.1 Application Runtimes

Figure 7.2 compares the execution times of the mesh and WiNoC interconnected manycore systems running FS and FS+ET Grappolo versions with both Directory and Hammer protocols. All the values in Figure 7.2 are normalized with respect to the execution times observed in the baseline system with mesh running FS version. As it can be observed from this figure, regardless of the NoC platform and cache coherence protocol employed, the use of early termination technique greatly benefits the execution of Grappolo. When compared to the FS version, the software implementation using approximate updates (i.e., FS+ET) achieves on an average of 40.7% and 39.1% execution time savings with the Hammer and Directory cache coherence, respectively.
7.1.2.2 Full System Energy

Figure 7.3 compares the energy consumptions of the manycore platforms executing the FS and FS+ET versions of *Grappolo*, employing Mesh and WiNoC architectures. All the values in this figure are normalized with respect to the energy consumption of the baseline architecture, the mesh NoC running the FS version.

Both the cache-miss rates and the subsequent NoC traffic caused by the cache-misses can be significantly reduced by the adopting early-termination. Compared to the baseline implementation (FS), the FS+ET implementation achieves from 29% (with ASJ graph) to 51% (with COND) lower cache miss induced NoC traffic, owing to its reduced number of shared memory accesses and lower run times. This reduced NoC traffic and lowered runtimes helps in enhancing the energy consumption profile of the manycore platform running FS+ET *Grappolo*. Even with mesh NoC, FS+ET achieves around 41% savings in full system energy consumptions, compared to FS version of *Grappolo*.

Compared to the baseline mesh NoC running FS version, the WiNoC interconnected manycore platform running FS+ET version of *Grappolo* achieves about 52% and 53.9%
savings in execution time and full system energy consumptions. These results demonstrate that software and hardware level innovations can lead to a significant acceleration of the community detection operation, while also making the proposed manycore platform more energy-efficient than the baseline architecture.

7.1.3 Opportunities

The approximate computing approach can be employed to accelerate a broad class of graph operations that have an iterative structure and are suited for approximation. Similar to accelerating on-chip memory access using a low-latency NoC, the off-chip data transfer latency can be lowered by employing efficient main-memory to processing core interconnection architectures.

7.2 Heterogeneous Manycore Platforms for Big Data Processing

Big Data computing involves high data parallelism and hence, is highly suitable to run on a manycore platform integrated with hardware accelerators such as Graphics Processing Units (GPUs). Similar to the homogenous manycore platforms considered in this work, the memory access latency in a heterogeneous platform can be minimized by employing efficient NoC architectures. However, the on-chip communication requirements of a heterogeneous manycore platform can vary significantly from that of the homogenous manycore platforms [112]. Moreover, as explained in Chapter 3, the on-chip communication requirements will also depend on the Big Data application under consideration and the cache coherence mechanism employed. To address these requirements, specialized on-chip interconnection networks for manycore platforms running Big Data processing are required.
7.3 Integrating SMART Control Network in Emerging Interconnects

The WiSMART network-creation and the routing mechanisms explained in Chapter 6 are agonistic of the technology that is used to create the long-range on-chip communication links. Hence, the WiSMART NoC can also employ the surface wave (SW) interconnects instead of wireless interfaces. Moreover, the WiSMART collective communication approach can be adopted in other emerging NoCs such as 3D and photonic NoCs. However, such adoptions have both physical implementation and routing challenges of their own and hence, require thorough investigation. For example, a 3D SMART NoC would require multi-drop TSVs integrated with clockless repeaters and uncongested broadcast trees spanning along three dimensions. We plan to further explore the 3D-SMART and SW-SMART augmentations as parts of our future works.
References


