Part 1

MULTICORE AND MANY-CORE (MC) SYSTEMS-ON-CHIP

1

A RECONFIGURABLE ON-CHIP INTERCONNECTION NETWORK FOR LARGE MULTICORE SYSTEMS

Mehdi Modarressi and Hamid Sarbazi-Azad

CONTENTS

Large Scale Network-Centric Distributed Systems, First Edition. Edited by Hamid Sarbazi-Azad and Albert Y. Zomaya.

1.1 INTRODUCTION

1.1.1 Multicore and Many-Core Era

With the recent scaling of semiconductor technology, coupled with the ever-increasing demand for high-performance computing in embedded, desktop, and server computer systems, the current general-purpose microprocessors have been moving from single-core to multicore and eventually to many-core processor architectures containing tens to hundreds of identical cores [1]. Major manufacturers already ship 10-core [2], 16-core [3, 4], and 48-core [5] chip multiprocessors, while some special-purpose processors have pushed the limit further to 188 [6], 200 [7], and 336 [8] cores.

Following the same trend, current multicore system-on-chips (SoC) have grown in size and complexity and consist of tens to hundreds of logic blocks of different types communicating with each other at very-high-speed rates.

1.1.2 On-Chip Communication

As the core count scales up, the rate and complexity of intercore communications increase dramatically. Consequently, the efficiency of on-chip communication mechanisms has emerged as a critical determinant of the overall performance in complex multicore system-on-chips (SoCs) and chip multiprocessors (CMPs). In addition to the performance considerations, on-chip interconnects of a conventional SoC and CMP account for a considerable fraction of the consumed power, and this fraction is expected to grow with every new technology point. The advent of deep submicron and nanotechnologies and supply voltage scaling also brings about several signal integrity and reliability issues [9]. As a result, interconnect design poses a whole new set of challenges for SoC and CMP designers.

1.1.3 Conventional Communication Mechanisms

Conventional small-scale SoCs and CMPs use the legacy bus and ad hoc dedicated links to manage on-chip traffic. With dedicated point-to-point links, the intercore data travel

on dedicated wires directly connecting two end-point cores. Thus, they can potentially yield the ideal performance and power results when connecting a few cores. However, when the number of on-chip components increases, this scheme requires a huge amount of wiring to directly connect every component, with less than 10% average wire usage in time [10]. Consequently, the poor scalability due to considerable area overhead is a prohibitive drawback of dedicated links. In addition, the dedicated wires in submicron and nanotechnologies need special attention to manage hard-to-predict power, signal integrity, and performance issues. Furthermore, due to their ad hoc nature, dedicated links are not reusable. These issues bring the design effort to the forefront as the second drawback of the dedicated wires.

Bus architectures are the most common and cost-effective on-chip communication solution for traditional multicore SoCs and CMPs with a modest number of processors. However, bus-based communication schemes, even those utilizing hierarchies of buses, can support a few concurrent communications. Connecting more components to a shared bus would also lead to large bus lengths that in turn result in considerable energy overhead and unmanageable clock skew. Therefore, when the number of devices that need to communicate is high, bus-based systems show poor power and performance scalability [9]. Such scalability problems continue to increase, as technology advances allow more cores to be integrated on a single chip. The scalability and bandwidth challenges of the bus have led to a shift in the board-level interchip communication paradigm and the widely used PCI bus is replaced by the switch-based PCI Express network-on-board.

The on-chip communication has traveled the same path in the past decades: the problems of the bus and dedicated links and the efficiency of packet-based interconnection networks in parallel machines motivated researchers to propose switch-based network-on-chips (NoCs) to connect the cores in a high-performance, flexible, scalable, and reusable manner [10–12].

1.1.4 Network-on-Chip

Networks on chip have now expanded from an interesting area of research to a viable industrial solution for multicore processors ranging from high-end server processors [5] to embedded SoCs [13]. The building blocks of on-chip networks are the routers at every node that are interconnected by short local on-chip wires. Routers multiplex multiple communication flows (in the form of data packets) over the links and manage the traffic in a distributed fashion. Relying on a modular and scalable infrastructure, NoCs can potentially deliver high-bandwidth, low-latency, and low-power communication. From the communication perspective, this allows integration of many components on a single chip.

The benefits of NoCs in providing scalable and high-bandwidth communication are substantial. However, the need for complex and multistage pipelined routers presents several challenges in reaching the potential latency and throughput of NoCs, due to their tight area and power budgets. Authors in [1] show that the bandwidth demands of future server and embedded applications is expected to grow greatly and project that in future CMPs and multicore SoCs, the power consumption of the NoCs implemented by the current methodologies will be about 10 times greater than the power budget that can be devoted to them. Therefore, much research has focused on improving NoC efficiency

to bridge the existing gap between the current and the ideal NoC power/performance metrics.

Application-specific optimization is one of the most effective methods to increase the efficiency of the NoC [1]. This class of optimization methods tries to customize the architecture and characteristics of an NoC for a target application. These methods can work at either design time, if the application and its traffic characteristics are known in advance (which is the case for most embedded applications running on multicore SoCs), or at run time for the NoCs used in general-purpose CMPs.

There has been substantial research on application-specific optimization of NoCs, varying from simple methods that update routing tables for each application to sophisticated methods of router microarchitecture and topology reconfiguration [14].

1.1.5 NoC Topology Customization

The performance of a NoC is extremely sensitive to its topology, which determines the placement and connectivity of the network nodes. Proper topology, consequently, is an important target for many NoC customization methods. An equally important problem in specialized multicore SoCs is core (or processing node) to NoC node mapping, which determines on which NoC node each processing core should be physically placed. Mapping algorithms generally try to place the processing cores communicating more frequently near each other; note that when the number of intermediate routers between two communicating cores is reduced, the power consumption and latency of the communication between them decreases proportionally.

Topology and mapping deal with the physical placement of network nodes and links. As a result, the mapping and topology cannot be modified once the chip is fabricated and will remain unchanged during system lifetime. Due to this physical constraint, most current design flows for application-specific multicore SoCs are only effective in providing design time mapping and topology optimization for a single application [15–18]. In other words, they generate and synthesize an optimized topology and mapping based on the traffic pattern of a single application.

This makes problems for today's multicore SoCs that run several different applications (often unknown at design time). Since the intercore communication characteristics can be very different across different applications, a topology that is designed based on the traffic pattern of one application does not necessarily meet the design constraints of other applications. Even the traffic generated by a single application may vary significantly in different phases of its operation. For example, the IEEE 802.11n standard (WiFi) supports 144 communications modes, each with different communication demands among cores [19]. In [20], more than 1500 different NoC configurations (topology, buffer size, and so on) are investigated and it has been shown that no single NoC can be found to provide optimal performance across a range of applications.

1.1.6 NoCs and Topology Reconfigurations

In this chapter, we introduce a NoC with reconfigurable topology, which can dynamically match the communication pattern of the currently running application.

The reconfiguration of the proposed architecture is achieved by inserting several simple switches in the network, allowing the network to dynamically change the internode connections and implement the topology that best matches the communication pattern of the running application. In other words, we try to reduce the hop count (or number of routers) between the source and destination nodes of high-volume (or heavy) communication flows by bypassing the intermediate routers. This can lead to considerable performance improvement since the latency (power) of the router pipeline stages makes a significant contribution to the total NoC latency (power). For example, in Intel's 80-core TeraFlops, more than 80% of the on-chip communication power is consumed by routers [4].

We then explore different reconfigurable NoC architectures by altering the placement of routers and switches. The most interesting structure is achieved by grouping the processors into some mesh clusters and using the reconfigurable switches to connect the clusters in a hierarchical fashion. This hierarchy consists of several clusters with fixed topology at the first level and a reconfigurable topology at the second level.

The topologies proposed for on-chip networks vary from regular tiled-based [21–23] to fully customized [15, 16] structures. Because fully customized NoCs are designed and optimized for some specific applications, they give the best performance and power results for those applications. However, distorting the regular structure of standard topologies leads to a nonreusable ad hoc topology with several implementation issues such as uneven wire lengths and routers with varying number of ports. On the other hand, regular NoC architectures provide standard structured interconnects that ensure well-controlled electrical parameters. In these topologies, designers can solve usual physical design issues like crosstalk tolerance, timing closure, and wire routing for a specific regular topology and reuse it in several designs. This reusability effectively reduces the design time.

Our reconfigurable NoC stands between these two extreme points of NoC design schemes and benefits from both worlds. While this NoC architecture is designed and optimized like a regular NoC, it can be dynamically reconfigured to a topology that best matches the traffic pattern of the currently running application. In other words, this architecture realizes application-specific topologies over structured and regular components.

1.1.7 Reconfigurations Policy

The NoC architecture introduced in this chapter and its design flow can be employed in both homogenous CMPs and heterogeneous multicore SoCs. Here, we focus on multicore SoCs, where the set of target applications is often known a priori. Each application is described as a set of concurrent tasks that have been already assigned and scheduled onto a set of selected IP cores. In such systems, mapping involves physical placement of the cores into the network nodes at design time. The design flow associated with this reconfigurable architecture can then find a customized topology for each application (based on the NoC mapping and application traffic demand) and load it onto the network when the application starts execution. The dynamic run-time reconfiguration policy for supporting homogenous CMPs with unpredictable traffic can be found elsewhere [24].

The reconfigurable topology of this chapter can be coupled with some CMP management schemes to boost their performance. The key to most of these techniques is to

allocate data accessed by threads to cache banks or memories closer to the processing core that is executing the thread. For example, R-NUCA, a state-of-the-art block placement and cache-management policy for CMPs, suggests decomposition of a large-scale CMP into virtual clusters [25], each with its own subset of the cache shared by the cores within the cluster. Since most data transfers occur among the cores within a cluster, the proposed reconfigurable topology, in particular in the extended hierarchical form, can be customized to support power- and performance-efficient communication among cluster cores.

Similarly, the specialization methods that trade the silicon real estate for energy efficiency and performance can benefit from the reconfigurable topology. Recent works [26, 27] have shown that, as transistor counts grow with each process generation, increasing core counts would not lead to performance improvements, because chips are physically constrained in power and off-chip bandwidth. Therefore, a fraction of on-chip transistors, referred to as *dark silicon*, must be power-off or underutilized to stay within power and bandwidth budgets. Specialization is the most promising solution to the dark silicon problem [26, 28]. In this approach, the unused dark silicon is exploited to implement a large number of specialized accelerator cores and power up only the subset of these cores that most closely match the requirements of the executing workload at a given time. Specialized cores are designed by characterizing the target workloads and identifying the functional units that execute different parts of the codes in a performance- and power-efficient manner. The specialized cores include ASIC accelerators, GPUs, DSPs, and FPGAs. The reconfigurable topology introduced in this section is one of the best options for implementing the network-on-chip in such manycore CMPs. It can dynamically set up a customized topology among the active cores at any time to efficiently manage the intercore traffic of the current workload.

1.2 TOPOLOGY AND RECONFIGURATION

A large body of research has focused on different optimization methods for on-chip networks to reduce the power consumption and message latency. Some of these methods aim to reduce the power consumption and latency within a network hop by optimizing the microarchitecture of the router and switching mechanism [29–31]. To reduce the message latency, most of these (often general-purpose) optimization methods try to cut down the router critical path delay by parallelizing multiple pipeline stages [29, 32, 33]. Similarly, the power reduction is mostly achieved by reducing the router activity and the total capacitance switched per cycle [34, 35]. These methods are all orthogonal to our reconfigurable topology, which aims to reduce the hop count rather than per hop energy and latency. Mapping, routing, and topology selection mechanisms, on the other hand, try to decrease the length of the path taken by the packets.

A popular choice for modern NoCs is regular topologies such as mesh, which is laid out on a two-dimensional plane [22, 23, 36]. Despite the advantages of meshes for on-chip implementation, some packets may suffer from long latencies due to lack of short paths between remotely located nodes. Owing to the fact that the communication traffic characteristics of multicore SoCs used in embedded systems are nonuniform and

can usually be obtained statically, an application-specific NoC with a custom topology that satisfies the design objectives and constraints of the target application is more appropriate [15–17, 37]. The problems of topology selection and core-to-NoC mapping have been explored in the past [17, 38, 39], but almost all of them are only effective in optimizing the topology based on a set of communication constraints obtained from a single application. Some solutions, however, have considered customizing NoC topology for multiple applications by using reconfigurability [20, 40–44].

In [41], topology reconfiguration is achieved by wrapping the NoC routers by some logic called *topology switch*. In this architecture, topology can be dynamically reconfigured by bypassing some routers through such switches. The authors report 56% reduction in power consumption and 10% area overhead for a multimedia application [41]. Authors in [20] introduce a polymorphic NoC with a configurable set of buffers, crossbars, and links on which an arbitrary network can be constructed. The network can be configured to offer the same performance as a fixed function network while incurring 40% area overhead, on average. However, authors have not analyzed the power consumption of their NoC architecture. In [44], a NoC is designed based on the worst-case delay and throughput constraints of a set of input applications. By applying DVS and DFS at run time, the power consumption of the NoC is optimized while the performance constraints of the currently running application are met.

Virtual point-to-point connections (VIPs) [45], express virtual channels (EVCs) [31], and shortcut paths (SCPs) [46] try to reduce the average packet hop count over a regular mesh topology by virtually bypassing some intermediate nodes along packet paths. A semiregular topology is presented in [18] by inserting some physical long links between distant nodes in a mesh, based on the traffic pattern of the target application. Long-range links target the nodes with high traffic volumes and are constructed statically at design time. The authors reported significant improvements over a conventional mesh. In addition to lack of support for multiple applications, this NoC cannot fully exploit the reusability and predictability of regular topologies; physical design and optimization procedures must be repeated for each NoC as the long links are specific to that specific NoC.

1.3 THE PROPOSED NOC ARCHITECTURE

1.3.1 Baseline Reconfigurable NoC

The proposed reconfiguration mechanism relies on an array of simple switches to dynamically change the internode connections. Figure 1.1 shows how the proposed reconfiguration is added to a conventional 2D mesh network. In this architecture, the network nodes, which are composed of a processing element and a router (represented by squares in Fig. 1.1) are not connected directly to each other, but through simple switch boxes, called *configuration switches* (represented by circles in Fig. 1.1). This structure is inspired by the reconfigurable processor arrays in [47].

Figure 1.1 also shows the internal structure of a configuration switch. It consists of some simple switching fabric that can establish connections between incoming and

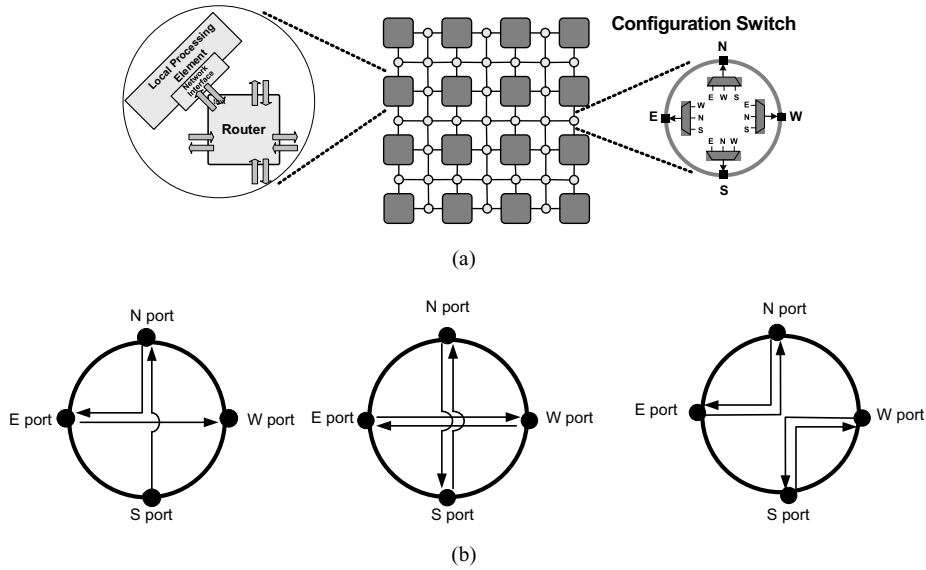


Figure 1.1 The reconfigurable NoC architecture (a) and three possible switch configurations (b).

outgoing links. Actually, the internal connections can be implemented by either a crossbar or four multiplexers, each at an output port of the switch. Figure 1.1 displays three possible switch configurations.

Compared to a router, switches have no buffers, no arbitration and routing logic, and smaller switching fabric (4×4 crossbars with negligible activity on the *select line* of cross-points, compared to a 5×5 crossbar of mesh routers). Furthermore, since a connection coming through an input port does not loop back, a smaller number of cross-points can be used. Some details about optimizing the switch structure and several implementation issues can be found in [48].

An important consideration in the proposed topology is that long links that may be generated by merging a number of channel segments by chaining the configuration switches. Such long links may decrease the NoC clock frequency, if the link delay exceeds one clock period. This problem can be solved by segmenting the long links into fixed-length links connected by a 1-flit register buffer and sending flits over them in a pipelined fashion. Since the connection between two adjacent nodes (on which flits travel in a single NoC cycle) consists of two channel segments, the registers should be placed in some configuration switches in such a way that each flit is buffered after passing through two channel segments. Figure 1.2 shows the switches in which the flits are latched. For example, the bold wire segments in the figure form a long link between nodes X and Y. Here, each flit passes over this link in five cycles, in a pipelined manner.

This NoC can be configured to implement arbitrary topologies, including some standard topologies, if the configuration switches are set properly. For example, Fig. 1.3.a

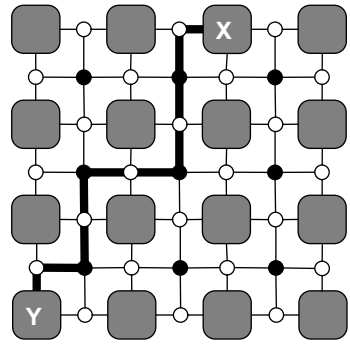


Figure 1.2 Flits traverse the connection between nodes X and Y (the black solid line) in five cycles in a pipelined fashion. Flits are buffered at the black switches.

displays an 8×8 network configured as a 2D mesh. Figure 1.3.b shows the case where a workload is running on a selected set of specialized cores (with black color) and the other cores are left dark (inactive). In this figure, the active nodes are connected by a mesh topology implemented on the reconfigurable NoC.

Although we used the mesh topology as the base of our reconfigurable NoC architecture, the proposed reconfiguration mechanism can be applied to other well-known topologies, such as torus, hypercube, and the general k-ary n-cube. It can also be used in some modern topologies, such as concentrated mesh (or X-mesh) networks, to further improve their performance when dealing with multiple applications.

The proposed NoC is not restricted to specific switching and routing schemes. The NoC routers in this study adopt a wormhole switching mechanism; note that wormhole

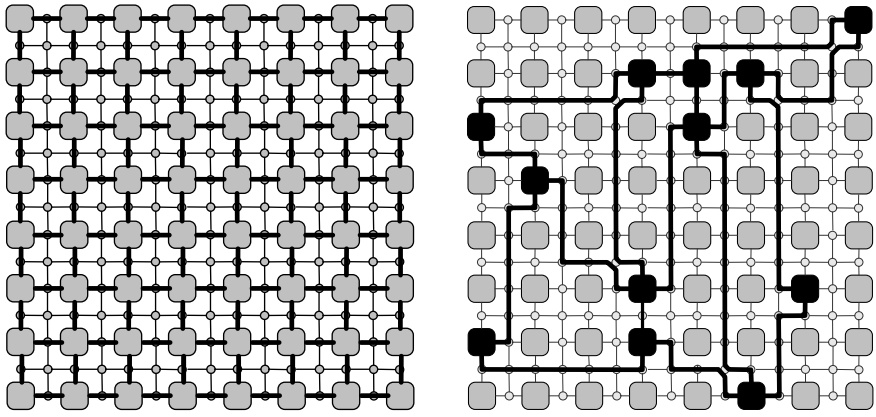


Figure 1.3 The implementation of a mesh on the reconfigurable topology among all network nodes (a) and a selected set of cores, where the black nodes are active and the gray cores are inactive (dark) (b).

switching best suits the limited buffering resources and low-latency communication requirements of on-chip networks. Like most application-specific optimized NoCs, this NoC applies a table-based routing scheme. This allows the NoC to support any static routing algorithm and is a suitable choice due to the irregular nature of application-specific topologies. It also allows the designer to exploit our understanding of the application traffic characteristics and avoid network congestion and load unbalance by appropriately allocating paths to traffic flows.

The NoC reconfiguration process can be initiated by a configuration manager, whenever a switching takes place between two applications. Switching between network configurations is done in parallel with application switching. In most SoC designs, the application switching time is of the order of few milliseconds, which is the time needed to load the data and code of a new application into the SoC, sending control signals to different parts of the SoC and shutting down the old application [44]. Because the configuration data of the proposed NoC architecture are small and can be stored in configuration switches and routers, the NoC configuration switching time is far smaller than the time needed to switch between applications and does not impose any additional delay to the application switching procedure. It has been shown in [44] that, even if the configuration data are stored in an off-chip memory, it can be loaded and distributed around the NoC in few microseconds, which is still shorter than the application switching time. The energy dissipated for configuration switching can also be ignored due to infrequent switching and the small amount of data transition during each switching event.

In general, dynamic hardware reconfiguration can only be implemented on reconfigurable devices; hence, most of the reconfigurable architectures are implemented using FPGAs. However, since the reconfigurable part of the proposed NoC is limited to some simple configuration switches, this NoC can be implemented on both FPGA and ASIC platforms.

The proposed reconfigurable network pays for the flexibility with additional area overhead. Figure 1.4 shows the effects of different architectural parameters, including buffer depth, network size, and link width, on the area overhead of the proposed

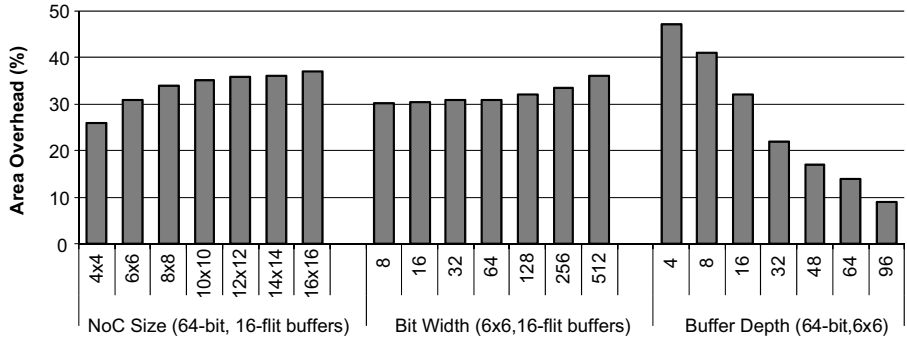


Figure 1.4 The area overhead of the reconfigurable NoC for different buffer depths (flits), network sizes (number of nodes), and link widths (bits).

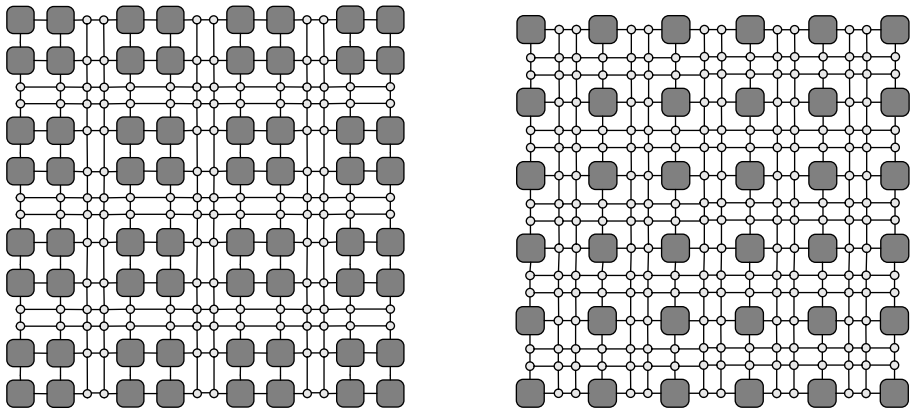
reconfigurable NoC over a conventional NoC. The results are obtained by a detailed area model for NoC components [48]. The figure reports 10–45% area overhead over a conventional NoC for some common NoC configurations. Please note that Fig. 1.4 reports the area overhead over a conventional network, not the entire chip (cores+network).

1.3.2 Generalized Reconfigurable NoC

We can explore different reconfigurable structures by altering the placement of routers and switches. Figure 1.5 shows two interesting reconfigurable structures. The structure in Fig. 1.5.a is achieved by increasing the number of configuration switches between two adjacent routers, or NoC *corridor width*, to two switches. Reconfigurable NoCs with wider corridor widths best match the applications with a large number of intercore communication flows.

The other alternative placement is depicted in Fig. 1.5.b. In this structure, the nodes are grouped into four-node clusters. The connection among the nodes within a cluster is fixed, but the clusters are interconnected via configuration switches. The idea behind this structure is to benefit from the interesting characteristics of the mesh topology while avoiding its drawbacks. From the traffic management perspective, a mesh NoC is efficient in handling local traffic patterns where each node communicates with its adjacent nodes, but it suffers from the lack of short paths between remotely located nodes.

The proposed architecture can efficiently support local traffic by mapping the tasks generating high intertask traffic flows into the same cluster. Unlike the baseline reconfigurable NoC in Fig. 1.1, in this new architecture, the local traffic does not pass through configuration switches, hence the power consumed in theses switches during data communication between adjacent nodes is saved. The problem associated with



(a) (b)

Figure 1.5 Reconfigurable NoC with the corridor width of 2 (a) and a cluster-based reconfigurable NoC with cluster size of 4 (b).

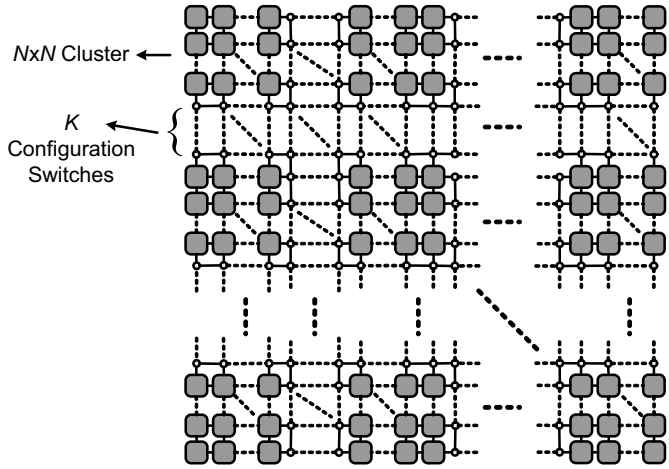


Figure 1.6 The generalized cluster-based reconfigurable structure.

communication between far nodes can be also mitigated by configuring the intercluster connections in order to make a direct connection, to reduce the hop count, between the endpoint nodes of long traffic flows.

These reconfigurable structures can be extended to a more generalized reconfigurable NoC structure depicted in Fig. 1.6. This architecture offers a hierarchical clustered communication infrastructure with two parameters: cluster size and corridor width. This structure can be considered as a hierarchical topology in which several mesh subnetworks, which provide the first-level local communication structure, are connected via a higher-level global network implemented by reconfigurable switches. Like other hierarchical NoCs [49–51], this two-tier topology facilitates local and global communication between cores. Consequently, this structure is an ideal candidate for the next generation many-core CMPs and MPSoCs where hundreds and thousands of cores are integrated into a single chip.

As mentioned before, the reconfigurability is the advantage of our architecture over the existing hierarchical NoC architectures. The intracluster topology is not limited to mesh, and any topology can be used for this purpose. In particular, when the cluster size is small (Fig. 1.5), using a bus or crossbar may be a more practical choice to support the local communication.

1.4 ENERGY AND PERFORMANCE-AWARE MAPPING

1.4.1 The Design Procedure for the Baseline Reconfigurable NoC

In this section, we address the mapping and routing problems in the baseline reconfigurable NoC. It is assumed that each input application is spatially partitioned into several tasks, each of which is nonmigratory and assigned to a processing core. The intercore

communication pattern remains relatively static as each core performs a fixed task. Each input application is described as a Communication Task Graph (CTG). The CTG is a directed graph $G(V, E)$, where each $v_i \in V$ represents a task, and a directed edge $e_{i,j} \in E$ represents the communication flow from v_i to v_j. The communication volume (bits per second) corresponding to each edge $e_{i,j}$ is also provided and is denoted by $t(e_{i,j})$.

Simply stated, for a given set of input applications whose tasks are assigned to a specific set of processing cores, our objective is to (1) map the cores into different nodes of a reconfigurable NoC, (2) find a customized topology for each application based on the mapping in previous step and the application traffic characteristics, and (3) find a route for the traffic flows of each application based on the topology found for the application.

We develop a two-step algorithm for this problem, where core-to-network mapping is done at the first step and then topology and route generation are done concurrently at the second step. The idea behind splitting the procedure into two steps is that, in our proposal, the reconfigurability only changes the connectivity among the cores and not the physical placement of them; so, our system is comprised of some nodes with fixed placement and a reconfigurable set of links. Consequently, mapping and topology selection should be done per MPSoC and per application, respectively. This suggests that our procedure must be carried out in two subsequent steps: the first step handles physical placement of cores by considering all applications and the second step works on each individual application and forms a proper topology for it.

1.4.1.1 Core-to-Network Mapping. In the first step, our objective is to figure out how to physically map the cores required by input applications onto different tiles of a mesh network such that the distances between the communicating cores are minimized. We assign a weight to each task graph based on its criticality, for example, the percentage of time that the corresponding application is run on the NoC. Assigning weights enables the designer to bias the mapping for major or critical applications.

This step is performed by constructing a synthetic average task graph (the average task graph) from the task graphs of the given set of input applications. This average task graph includes all the nodes of all task graphs of the input applications. For the edge between every pair of nodes, the average weight of the volumes relating to the corresponding edges across all task graphs are calculated and used in the average task graph. If an edge does not exist in a task graph, its volume is considered to be zero. More formally, the weight (communication volume) of each edge is calculated as

$$t_{avr}(e_{x,y}) = \left(\sum_{\forall applications} t_i(e_{x,y}) \times W_i \right) \Big/ n$$

where W_i represents the weight of the i^{th} task graph and $t_i(e_{x,y})$ and $t_{avr}(e_{x,y})$ denote the volume of $e_{x,y}$ (which defines the edge between nodes v_x and v_y) in the i^{th} task graph and the average task graph, respectively, and n is the number of input task graphs.

The mapping problem can be formulated as follows. Given a synthetic average CTG constructed from the task graphs of the input applications and a reconfigurable NoC (satisfying $size(CTG) < size(NoC)$, find a mapping M from CTG to the NoC

nodes as $Min\left\{\sum_{\forall e_{i,j}} t(e_{i,j}) \times dist(M(v_i), M(v_j))\right\}$ such that for every node $v_i \neq v_j$ of the CTG, we have $M(v_i) \neq M(v_j)$. The constraint states that each core should be mapped to exactly one NoC node, and no node can host more than one core. $dist(a, b)$ shows the Manhattan distance between nodes a and b in the network, and $M(v_i)$ is the network node to which CTG node v_i is mapped. $size(CTG)$ and $size(NoC)$ denote the number of CTG and NoC nodes, respectively. Again, $t(e_{i,j})$ denotes the volume of $e_{i,j}$, which represents the edge between vertex v_i and vertex v_j.

Central to network mapping is an NP-hard problem [52], therefore rather than searching for an optimal solution, it has been solved heuristically in prior works [22, 37, 53]. As the focus of this chapter is the topology reconfiguration, we perform mapping for the average graph using NMAP, a well-known and popular heuristic method presented in [22]. NMAP uses a heuristic algorithm for power-aware mapping of task graph nodes on a mesh-based network and selects a route for each task graph edge. We only use the mapping algorithm of NMAP and then, in the next step, propose a topology and route selection algorithm based on the reconfigurable network links.

In NMAP, all cores are initially unmapped. Then, the core mapping is accomplished as follows:

Step 1. Map the core with the maximum communication demand onto one of the mesh nodes with maximum number of neighbors.

Step 2. Select the core that communicates the most with already mapped cores and examine all unallocated mesh nodes for placement. Select the node that minimizes the communication cost between the current core and already mapped cores. The communication cost of mapping vertex v_i of the CTG into node x of the NoC is given by $\sum_{\forall j | e_{i,j} \in CTG}(t(e_{i,j}) \times dist(x, M(v_j))$, where $dist(x, M(v_j))$ is the Manhattan distance between x and the node to which CTG vertex v_j is mapped.

The process is repeated until all cores are mapped. We refer interested readers to [22] for more details on the NMAP.

1.4.1.2 Topology and Route Generation. Once the mapping is obtained from the average task graph, a suitable topology is constructed for each individual application, aiming to reduce the NoC average power consumption and message latency when the application is being processed. To achieve this goal, we implement a topology for each application where the number of hops between the source and destination nodes of heavy communication flows is as small as possible. The main idea is to choose the heaviest communication flow that is not yet assigned a route and find a path with minimum possible hop counts for it. Finding this route may involve configuring the switches that are not yet configured in order to skip over some intermediate routers and make a shorter connection between the end nodes. As a result, route selection and topology construction are done in parallel, within the same procedure. The algorithm can configure the unconfigured internal connections of the configuration switches but not the connections that

have been configured at previous iterations of the algorithm for the edges with higher volumes.

Initially, in the topology selection algorithm, all edges of an application task graph are stored in a decreasing order (based on their communication volumes) and the internal connections of all configuration switches are unconfigured. Then, for each edge in the order, a branch-and-bound algorithm chooses the path with lowest cost between its source and destination nodes. We calculate the cost of a path based on the routers and configuration switches it includes. We assign a cost of 1 to a link ending to a configuration switch and a cost of 4 to a link ending to a router. This cost assignment scheme reflects the power/latency ratio of the switches and routers and encourages the algorithm to find a path through the configuration switches, hence creating a long, pipelined link between the flow endpoint nodes. The algorithm searches for the optimal path by alternating the following branch and bound steps:

Branch: Starting from the source router of the selected edge, the algorithm makes a new branch by adding a router/configuration switch adjacent to the current node of the partial path. Current node is defined as the last node added to a partial path through which the path is extended. The added node must be located within the shortest path area, that is, between the source and destination nodes of the edge. The shortest path area is defined by the nodes and configuration switches located along one of the shortest paths between the source and destination nodes, as well as their adjacent configuration switches. If the current node is a router, the path is extended by including its neighboring configuration switches along the shortest path toward the destination node. If the current node is a configuration switch, the path is extended by adding the neighboring routers or configuration switches along the shortest path. However, if a switch is already configured in previous steps of the algorithm for the flows with higher traffic rates (e.g., by connecting its E input port to S output port), the algorithm cannot consider some other paths that involve establishing conflicting turns on the switch (e.g., connecting E input port to W input port).

Bound: A partial path is bounded (discarded) in some conditions. First, it is bounded if, by adding the new node, the predefined bandwidth constraints of the newly added link are violated. More formally, the bandwidth constraint of each NoC link l_k must be satisfied as

$$\forall l_k,\, BW(l_k) \geq \sum_{\forall e_{i,j} \in E} X^k(i,\, j)$$

where $BW(lk)$ is the bandwidth of link l_k and $X^K(i,\, j)$ is obtained by

$$X^k(i,\, j) = \begin{cases} t(e_{i,j}) & ,\ if\ l_k \in path(e_{i,j}) \\ 0 & ,\ otherwise \end{cases}$$

where $path(e_{i,j})$ represents the set of links on which task graph edge $e_{i,j}$ (with volume $t(e_{i,j})$) is mapped.

In addition, if the cost of a partial path reaching a node is larger than the minimum cost of the partial paths already reaching that node, the path is discarded. The minimum cost of the already-found paths is also kept by the algorithm and a partial path is bounded when its new cost exceeds this value. Finally, we perform a connectivity check to verify that there is at least one path between the source and destination nodes of all edges that are not mapped yet. If the current partial path configures the switches in such a way that all possible paths between the source and destination nodes of at least one unmapped edge are blocked, the partial path is removed.

After the path with the minimum cost is found, it is established in the NoC by configuring all corresponding configuration switches within the path. Then, the algorithm continues with the next edge. The algorithm is repeated for all the edges of the application CTG. Once all task graph edges are mapped to a path in the NoC, the paths are analyzed for detecting potential deadlocks. To this end, all cyclic dependencies among paths are broken by adding a virtual channel in one of the nodes of the cycle.

In this procedure, we assume that the applications are run on the NoC one at a time. Nonetheless, simultaneous execution of multiple applications is a likely scenario in MPSoCs. We can easily support simultaneous execution of multiple applications by the same procedure. To this end, we combine the CTGs of the applications that may run simultaneously, or within overlapping time intervals, into a single CTG and perform the topology generation and route selection steps for the new task graph. The selected configuration will then be loaded into the network when the applications are run simultaneously.

1.4.2 Mapping and Topology Generation for Cluster-Based NoC

The algorithm developed for the baseline-reconfigurable NoC is extended to a four-step algorithm to support the generalized cluster-based version of the reconfigurable NoCs. Here, applications with v vertices and a reconfigurable NoC with n nodes arranged as n/k clusters of size k are the inputs of the algorithm. For the sake of simplicity, we assume that $v = n$. We outline the steps of this algorithm below. The details of each step can be found in [54].

CTG partitioning: Partition the input CTG into n/k partitions of size k such that the partitions are disjoint and have equal size, and the sum of the weights of the edges with endpoints in different partitions is minimized. We then allocate each NoC cluster to a partition. This graph partitioning approach aims to group the frequently communicating CTG nodes in the same partition and eventually place them in the same cluster in order to reduce the intercluster connections by localizing the traffic. Graph partitioning is a well-known problem in graph theory, and a large number of algorithms have been proposed to solve it. Here, we

use the Kernighan-Lin algorithm, one of the most efficient heuristic multilevel algorithms presented in [55].

Partition-to-cluster mapping: This step deals with allocating the NoC clusters to the CTG partitions. The graph partitioning algorithm of Step 1 guarantees that the endpoint nodes of heaviest communication flows are mapped into the same cluster, but there are still communication demands between nodes located at different partitions. The aim of this step is to reduce the intercluster communication by mapping partitions with high interpartition traffic loads into nearby clusters. We perform the mapping using NMAP. By considering the partitions of a partitioned graph as a super-node and grouping all edges between any two given partitions as a super-edge, we get a new CTG. This new CTG is then fed to NMAP to allocate the NoC clusters to the CTG partitions.

Partition-node to cluster-core mapping: Once the partitions are mapped, this step aims to reduce the traffic inside each cluster and figures out how to map different nodes of each partition into the nodes of the target cluster. We again use NMAP for this step.

Intercluster topology implementation: Once all nodes are mapped, the configuration switches should be configured to establish connections among the clusters, based on the current intercluster communication pattern. This step applies a modified version of the branch-and-bound algorithm of the previous section to find a path with minimum weight for intercluster edges.

1.5 EXPERIMENTAL RESULTS

To evaluate the performance of the proposed NoC architecture and its design procedure, we use some existing SoC designs that have been widely used in the literature, including Multi-Window Display (MWD) [56], Video Object Plane Decoder (VOPD) [22], GSM [57], and Multi Media System (MMS) [23]. The MMS benchmark contains H.263 decoder, H.263 encoder, MP3 encoder, and MP3 decoder applications. GSM also contains the GSM decoder and encoder applications. However, for the first two SoCs that have a single application task graph, we synthesize additional task graphs, called X-50% and X-25%, where X is the name of the application. The X-25% and X-50% task graphs are generated by replacing the source and destination nodes of the edges of task graph X with other randomly chosen nodes (i.e., moving the position of the task graph edges) with a probability of 25% and 50%, respectively. We assign a weight of 0.5 to the base task graph X, a weight of 0.25 to X-25% and X-50% task graphs, and then integrate the task graphs into a single NoC, according to the design flow described in the previous section.

The conventional NoC used for the sake of comparison applies the same mapping as its reconfigurable counterpart, but its topology is fixed during execution of the applications. The communication flows of the conventional NoC are directed by the conventional dimension-order deterministic routing algorithm.

Simulation experiments are performed using Xmulator NoC simulator [58] for a 64-bit-wide system with speculative four-stage pipelined wormhole routers [59] with 16-flit buffers. The power results are reported by Orion power library [60] in 65-nm technology. In simulation experiments, packets are generated with exponential distribution, and the communication rates between any two nodes are set to be proportional to the communication volume between them in the task graph.

The task graphs of different MMS applications are depicted in Fig. 1.7, where the edge tags represent the communication volume between the source and destination nodes of the edge in kilobits per second. The applications use the same set of processing cores,

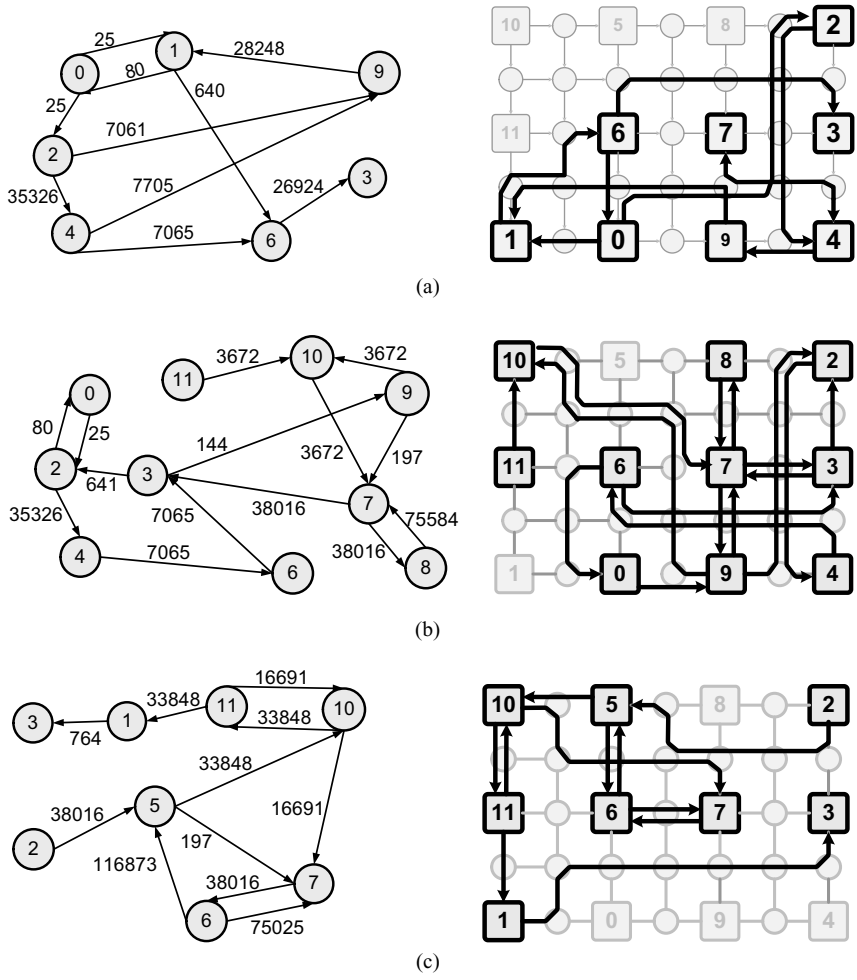


Figure 1.7 The communication task graph and corresponding topology for MP3 (encoder+decoder) (a), H263 decoder (b), H63 encoder (c).

but the traffic pattern among the cores is different for each application. In Fig. 1.7, the edges belonging to each individual application are bolded in the task graph corresponding to their applications. Different tasks of this application suite are mapped on 12 cores. The physical mapping is accomplished based on the average graph of the four input task graphs. The algorithm then finds a suitable topology for each application, which is illustrated next to the corresponding task graph in Fig. 1.7.

1.5.1 Baseline Reconfigurable NoC

Figure 1.8 displays the average packet latency and power consumption of the reconfigurable NoC and its equivalent conventional NoC for the mentioned multicore SoC benchmarks. The results show considerable power and performance improvements over a conventional NoC using NMAP. As the figure indicates, reconfiguration can effectively adapt the topology to different applications and reduce the power consumption and average packet latency of the NoC by up to 28% (16% on average) and 26% (10% on average), respectively.

As the results for MMS applications show, the power and performance gain obtained by the proposed architecture for MP3 encoder/decoder are higher than H.263 encoder/decoder. The reason is that the volume of the communication flows of H.263 applications, which are larger than the communication volumes of MP3 applications, biases the mapping for H.263 encoder and decoder. As a result, the source and destination nodes of H.263 applications are mapped near each other, while the nodes required by MP3 applications are placed at greater distances. However, we can configure the NoC to establish a direct connection between almost all communicating nodes of MP3 encoder and decoder. This leads to power and performance values close to the case when the communicating cores are mapped into nearby nodes.

Similarly, for the MWD and VOPD benchmarks, the weight of the original application is increased over the two synthetic applications which that make the mapping biased for the original application. As a result, reconfiguration provides more power and performance gains for the synthetic applications (X-25% and X-50%).

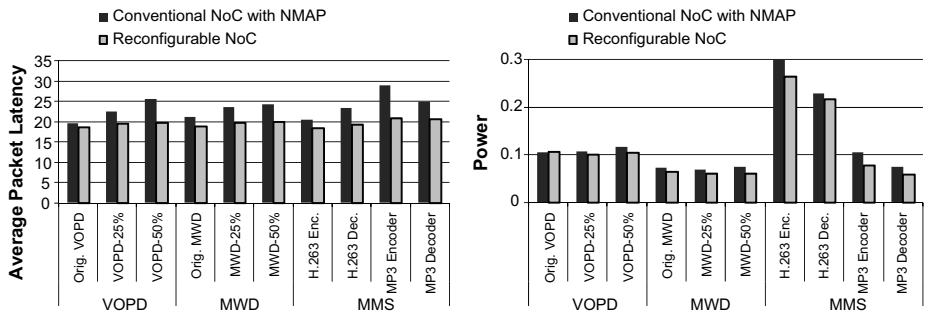


Figure 1.8 Power consumption (Watts) and packet latency (cycles for 8-flit packets) in a conventional and a reconfigurable NoC. The conventional NoC uses NMAP.

The reported power includes both dynamic and static parts. In this work, a dynamic power management scheme is used to deactivate unused router and switch ports in order to decrease the static power consumption of the reconfigurable NoC. The idle routers and links of the conventional NoC are also deactivated. The VOPD benchmarks, however, require more connections compared with other benchmarks, and thus involve more active switches, routers, and links. Consequently, the static power consumption of the reconfigurable NoC, when running the VOPD applications, is higher than the static power of a conventional NoC. This higher static power consumption may not be compensated by the obtained dynamic power saving, as reported for the VOPD original application in Fig. 1.8. Nonetheless, the proposed reconfigurable NoC enhances the NoC dynamic power consumption by 21%, on average.

1.5.2 Performance Evaluation with Cost Constraints

In this section, we compare the proposed NoC with a conventional NoC under cost constraints to investigate what improvements would be obtained if the extra logic used in our reconfigurable NoC is invested to increase the NoC bit width or buffer depth.

First, we use the extra logic to make NoCs with additional buffering capacity. Our area model reveals that a conventional NoC with two virtual channels and 8-flit buffers (Conv. 2VC, 8-flit buffers, 128-bit), a conventional NoC with one virtual channel and 16-flit buffers (Conv. 1VC, 16-flit buffers, 128-bit), and a reconfigurable NoC with corridor width of one, one virtual channel, and 8-flit buffers (Reconfig-C1, 1VC, 8-flit buffers, 128-bit) all have approximately the same area.

The extra area overhead can also be invested to increase the NoC bit width. By increasing the NoC bit width, the area of the NoC data-path components (buffers, links, and crossbars) is increased. According to our area model, the area of a 128-bit reconfigurable NoC (Reconfig-C1, 1VC, 8-flit buffers, in Fig. 1.9) is almost the same as the area of a conventional NoC with bit width of 158 (Conv. 1VC, 8-flit buffers, 158-bit, in Fig. 1.9).

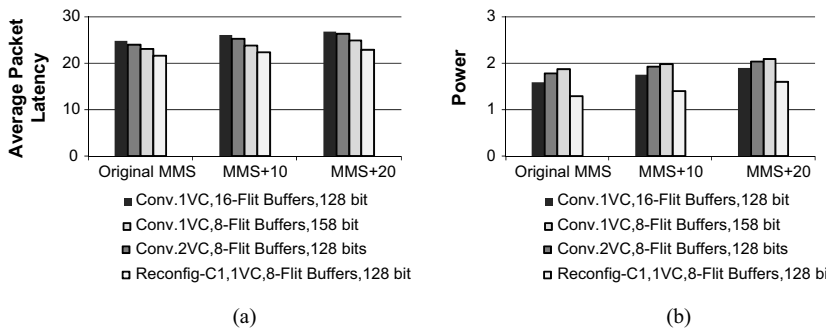


Figure 1.9 The average message latency (cycles for 8-flit packets) (a) and power (Watts) (b) of the three NoC configurations with the same area under the MMS traffic and its variants.

Figure 1.9 compares the power and latency results for these NoC configurations. The figure shows that our reconfigurable NoC efficiently exploits the area overhead to provide smaller latency and power consumption than the three equivalent conventional configurations. In this experiment, MMS benchmark of the previous section is used. In order to evaluate the impact of increasing on-chip traffic on performance improvement of the considered architectures, we generate two other task graphs, MMS+10% and MMS+20%, by randomly adding new edges to the MMS task graph in such a way that the volume of the MMS intercore traffic is increased by 10% and 20%, respectively.

Another possible way to invest the area overhead of the reconfigurable NoC is to use larger, more powerful processing cores. Our area model reveals that the reconfigurable NoC imposes less than 3% overhead to the entire area used by the MMS cores. This result is obtained by considering the reconfigurable NoC of Fig. 1.9 (Reconfig-C1, 1VC, 8-flit buffers) and an average size of 1 mm $\times$ 1 mm for each core. According to Pollack's rule [61], the increase in processor performance is roughly proportional to the square root of the increase in its area. Consequently, investing the area overhead to increase the area of the cores in the SoC with a conventional NoC leads to about a 1% increase in the performance of each core. As a result, investing this area overhead to increase the network performance would be a better choice, especially in the case that the network is in the critical path of the system.

1.5.3 Comparison Cluster-Based NoC

To compare the cluster-based and baseline reconfigurable NoCs, we again use the *GSM*, *MMS*, and *VOPD* applications. We adopt a cluster-based NoC with four clusters of size 2×2 for VOPD (with 16 cores), 9 clusters (arranged as a 3×3 mesh) of size 2×2 for *H263 + MP3* (with 36 cores), and 12 clusters (arranged as a 3×4 mesh) of size 2×2 for *GSM* (with 48 cores). The MMS application is the same as the application used in the previous section, but by using a different task-to-core assignment scheme its size is increased to 36 cores.

We perform the entire design flow (graph partitioning, mapping, and topology generation steps) for one primary application of each set (*GSM encoder, H.263 + MP3encoder, and VPOD*). Next, to evaluate the adaption capability of the proposed architecture, we assume that a new application of each set (*GSM decoder H.263 + MP3 decoder, and VPOD-50%*) is added to the system after chip fabrication. Again, note that as the NoC is already synthesized, the design steps related to physical core-to-network mapping (graph partitioning, partition to cluster mapping, and partition-node to cluster-core mapping) cannot be done for the new application, and, hence, we directly head to the topology generation step.

Figure 1.10 shows the power and latency results offered by the three networks. The improvement over the baseline reconfigurable NoC comes from the better mapping algorithm (by applying a graph partitioning algorithm together with the NMAP algorithm), as well as the more efficient switch placement of the cluster-based NoC.

For the primary applications, for which the physical mapping is done, we again observe that the improvement is not considerable. The reason is that the source and destination nodes of *GSM encoder* are mapped near each other and this leaves little room

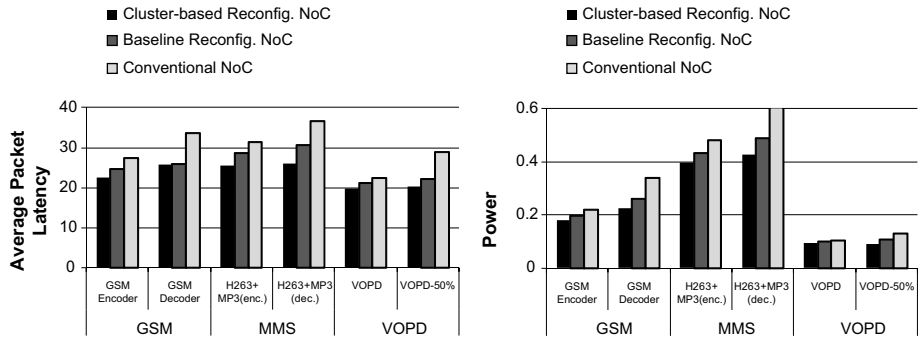


Figure 1.10 The average message latency (cycles for 8-flit packets) (a) and power (Watts) (b) of the three NoC configurations.

(the gap between the conventional and reconfigurable NoC results) for improvement. However, when a new application is run, such as *GSM decoder*, the nodes required by this application are potentially placed at farther distances (since the mapping is performed for some application with a different traffic pattern), so they can enjoy more from the reconfigurable long links.

We have also compared our proposed architecture with a state-of-the-art topology: the *Concentrated Mesh* (CMesh) topology proposed in [36]. In this topology, each router is connected to four processors. Although this topology takes full advantage of the locality of the intercore communication, it increases the switch size and introduces an additional router stage for switch preparation [36]. The simulation parameters and architectural properties of the cluster-based NoC are the same as the previous experiment. For the CMesh topology, we use our partitioning algorithm to create four-task partitions and apply NMAP to map the partitions onto the routers. Figure 1.11 shows the energy and latency results of the proposed NoC architecture and CMesh. As the figure

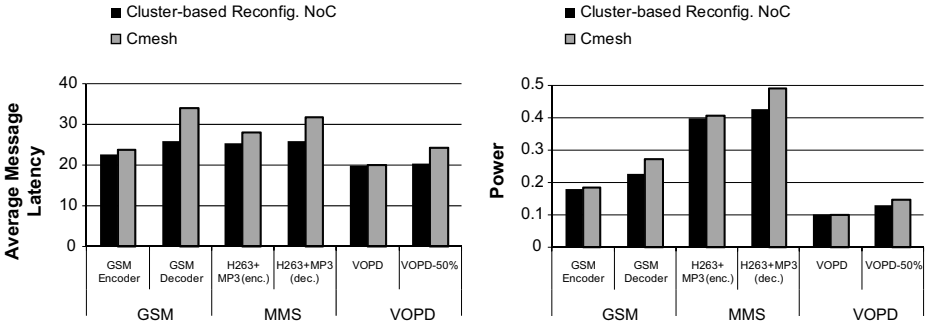


Figure 1.11 Comparing the average message latency (cycles for 8-flit packets) (a) and power (Watts) (b) with the CMesh topology.

indicates, our proposed architecture exhibits better average energy and latency results when compared with CMesh. Note that, again, the reconfigurability of our proposed NoC leads to more improvement over CMesh (with fixed connections) for the applications that are introduced after the NoC synthesis (i.e., GSM decoder, H.263+MP3 decoder, and VOPD50%).

1.6 CONCLUSION

We proposed a reconfigurable architecture for networks-on-chip (NoCs) on which arbitrary application-specific topologies can be implemented. Since entirely different applications may be executed on a SoC at different times, the on-chip traffic characteristics can vary significantly across different applications. However, almost all existing NoC design flows and the corresponding application-specific optimization methods customize NoCs based on the traffic characteristics of a single application. The reconfigurability of the proposed NoC architecture allows it to dynamically tailor its topology to the traffic pattern of different applications.

In this chapter, we first introduced the baseline reconfigurable NoC architecture. We next addressed the two problems of core to network mapping and topology exploration in which the cores of a given set of input applications are physically mapped to the network and then a suitable topology is found for each individual application. Experimental results, using some multicore SoC workloads, showed that this architecture effectively improves the performance of NoCs by 29% and reduces the power consumption by 9% over one of the most efficient and popular mapping algorithms proposed for conventional NoCs.

We then extended the baseline reconfigurable NoC to a generalized reconfigurable NoC architecture. This new cluster-based structure consists of several mesh clusters alongside a reconfigurable connection fabric that handles the intercluster communication. It can support both local and global traffic patterns in an efficient manner. Our evaluation results showed the effectiveness of the proposed architecture in reducing the latency and energy of on-chip communication.

In all, compared with previous reconfigurable proposals and regarding the imposed area overhead and power/performance gains, the proposed NoC introduces a more appropriate trade-off between the cost and flexibility.

REFERENCES

[1] J. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, and L.S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, Vol. 27, No. 5, pp. 96–108, 2007.

[2] Intel Xeon Processor E7 Family, http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family.html, Apr. 2012.

[3] P. Conway, et al., "Cache hierarchy and memory subsystem of the AMD Opteron processor," *IEEE Micro*, Vol. 30, No. 2, pp. 16–29, Mar.–Apr. 2010.

[4] SPARC T3 Data Sheet, http://www.oracle.com/products, Apr. 2012.

[5] J. Howard, et al., "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Proceedings of International Solid State Circuits Conference*, pp. 108–110, 2010.

[6] Vega Processor, http://www.azulsystems.com/products/vega/processor, May 2012.

[7] PC202 processor, http://www.picochip.com/page/75/, Apr. 2012.

[8] Am2045: Ambric's new parallel processor, www.ambric.info/pdf/MPR_Ambric_Article_10-06_204101.pdf, May 2012.

[9] V. Raghunathan, M.B. Srivastava, and R.K. Gupta, "A survey of techniques for energy efficient on-chip communication," in *Proceedings of the Design Automation Conference*, 2003.

[10] W.J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the Design Automation Conference*, pp. 681–689, 2001.

[11] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," *IEEE Computer*, Vol. 35, No. 1, pp. 70–78, 2001.

[12] A. Jantsch and H. Tenhunen, *Networks on Chip*, Kluwer Academic Publishers, 2003.

[13] Arteris NoC Customers, http://www.arteris.com/customers, Apr. 2012

[14] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, Vol. 38, No. 1, pp. 1–51, 2006.

[15] J. Cong, Y. Huang, and B. Yuan, "A tree-based topology synthesis for on-chip network," in *Proceedings of International Conference on Computer-Aided Design*, pp. 651–658, 2011.

[16] J. Chan and S. Parameswaran, "NoCOUT: NoC topology generation with mixed packet-switched and point-to-point networks," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 256–270, 2008.

[17] S. Murali and G. De Micheli, "SUNMAP: A tool for automatic topology selection and generation for NoCs," in *Proceedings of Design Automation Conference*, pp. 914–919, 2004.

[18] U. Ogras and R. Marculescu, "Application-specific network-on-chip architecture customization via long-range link insertion," in *Proceedings of Design Automation Conference*, 2005.

[19] A. Jerraya, "System compilation for MPSoC based on NoC," in *the 8th International Forum on Application-Specific Multi-Processor SoCs*, Netherlands, 2008.

[20] M. Kim, J. Davis, M. Oskin, and T. Austin, "Polymorphic on-chip networks," *in Proceedings of International Symposium of Computer Architecture*, pp. 101–112, 2008.

[21] Y. Hoskote, S. Vangal, A. Singh, N. Bokar, and S. Bokar, "A 5-GHz mesh interconnect for a Teraflops processor," *IEEE Micro*, Vol. 27, No. 5, pp. 51–61, 2007.

[22] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proceedings of Design Automation and Test in Europe*, pp. 896–901, 2004.

[23] J. Hu, and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 1, pp. 551–562, 2005.

[24] M. Modarressi, H. Sarbazi-Azad, and A. Tavakkol, "An efficient dynamically reconfigurable on-chip network architecture," in *Proceedings of Design Automation Conference*, pp. 310–313, 2010.

[25] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal block placement and replication in distributed caches," in *Proceedings of International Symposium of Computer Architecture*, pp. 184–195, 2009.

[26] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, Vol. 31, No. 4, pp. 6–15, July–Aug. 2011.

[27] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of International Symposium of Computer Architecture*, pp. 365–376, 2011.

[28] N. Goulding-Hotta, et al., "The GreenDroid mobile application processor: an architecture for silicon's dark future," *IEEE Micro*, Vol. 31, No. 2, pp. 86–95, Mar.–Apr. 2011.

[29] R. Mullins and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proceedings of International Symposium of Computer Architecture*, pp. 188–197, 2004.

[30] P. Abad, V. Puente, J. Gregorio, and P. Prieto, "Rotary router: An efficient architecture for CMP interconnection networks," in *Proceedings of International Symposium of Computer Architecture*, pp. 116–125, 2007.

[31] A. Kumar, L.S. Peh, P. Kundu, and N.K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in *Proceedings of International Symposium of Computer Architecture*, pp. 150–161, 2007.

[32] L.S. Peh and W.J. Dally, "A delay model for router microarchitectures," *IEEE Micro*, Vol. 2, No. 1, pp. 26–34, 2001.

[33] K. Kim, S. Lee, K. Lee, and H.J. Yoo, "An arbitration look-ahead scheme for reducing end-to-end latency in networks-on-chip," in *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, pp. 2357–2360, 2005.

[34] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. Yousif, C. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proceedings of International Symposium of Computer Architecture*, pp. 4–15, 2006.

[35] P. Meloni, S. Murali, S. Carta, M. Camplani, L. Raffo, and G. De Micheli, "Routing aware switch hardware customization for networks on chips," in *Proceedings of NanoNet*, 2006.

[36] J. Balfour and W.J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the International Conference of Supercomputing*, pp. 178–189, 2006.

[37] A. Weichslgartner, S. Wildermann, and J. Teich, "Dynamic decentralized mapping of tree-structured applications on NoC architectures," in *Proceedings of the International Symposium on Networks-on-Chip*, pp. 201–208, 2011.

[38] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 233–239, 2003.

[39] D. Bertozzi, A. Jalabert, S. Murali, R. Tamahankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 2, pp. 113–129, 2005.

[40] B. Zafar, J. Draper, and T.M. Pinkston, "Cubic ring networks: a polymorphic topology for network-on-chip," in *Proceedings of International Conference on Parallel Processing*, pp. 443–452, 2010.

[41] M. Stensgaard and J. Sparsø, "ReNoC: A network-on-chip architecture with reconfigurable topology," in *Proceedings of International Symposium on Networks-on-Chip*, pp. 55–64, 2008.

[42] S. Vassiliadis and I. Sourdis, "Flux networks: Interconnects on demand," in *Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 160–167, 2006.

[43] M. Modarressi and H. Sarbazi-Azad, "Power-aware mapping for reconfigurable NoC architectures," in *Proceedings of the International Conference on Computer Design*, pp. 417–422, 2007.

[44] S. Murali, M. Coenen, R. Radulescu, K. Goossens, and G. De Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proceedings of Design Automation and Test in Europe*, pp. 118–123, 2006.

[45] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Virtual point-to-point connections in NoCs," in *IEEE Transactions on Computer Aided Design for Integrated Circuits and Systems*, Vol. 29, No. 6, pp. 855–868, June 2010.

[46] N. Teimouri, M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Energy-optimized on-chip networks using reconfigurable shortcut paths," in *Proceedings of the Conference of Architectures for Computing Systems*, pp. 231–242, 2011.

[47] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Pubs., New York, 1984.

[48] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Application-aware topology reconfiguration for on-chip networks," in *IEEE Transactions on Very Large-Scale Integrated Circuits and Systems*, Vol. 19, No. 11, pp. 2010–2022, Nov. 2011.

[49] K. Lee, et al., "Low-power network-on-chip for high-performance SoC design," *IEEE Transactions on Very Large-Scale Integrated Circuits and Systems*, Vol. 14, No. 2, 2006.

[50] S. Bourduas and Z. Zilic, "A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing," in *Proceedings of the International Symposium on Networks-on-Chip*, pp. 195–204, 2007.

[51] R. Das, S. Eachempati, A.K. Mishra, N. Vijaykrishnan, and C.R. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," in *Proceedings of International Symposium on High Performance Computer Architecture*, pp. 175–186, 2009.

[52] G. Ascia, M. Catania, and M. Palesi, "An evolutionary approach to network-on-chip mapping problem," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 112–119, 2005.

[53] K. Srinvasan, K. Chatha, and G. Konjevod, "Linear programming-based techniques for synthesis of network-on-chip architectures," in *IEEE Transaction on Very Large-Scale Integrated Circuits and Systems*, Vol. 14, No. 4, pp. 407–420, 2006.

[54] M. Modarressi and H. Sarbazi-Azad, "Reconfigurable cluster-based networks-on-chip for application-specific MPSoCs," in *Proceedings of the International Conference of Application-Specific systems, Architectures, and Processors*, 2012.

[55] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal of Sci. Comput*ing, Vol. 1, No. 20, pp. 359–392, Dec. 1998.

[56] K. Srinvasan and K. Chatha, "A low complexity heuristic for design of custom network-on-chip architectures," in *Proceedings of Design Automation and Test in Europe*, pp. 130–135, 2006.

[57] M. Schmitz, Energy minimization techniques for distributed embedded systems, Ph.D. thesis, University of Southampton, 2003.

[58] Xmulator NoC Simulator, 2008, http://www.xmulator.org, May 2012.

[59] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan-Kaufmann Publishers, 2004.

[60] A. Kahng, B. Lin, L. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of Design Automation and Test in Europe*, 2009.

[61] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the Design Automation Conference*, pp. 746–749, 2007.