UNIVERSIDAD DE CANTABRIA

PROGRAMA DE DOCTORADO EN CIENCIA Y TECNOLOGÍA

TESIS DOCTORAL

MECANISMOS DE BAIPÁS EFICIENTES PARA REDES EN CHIP DE BAJA LATENCIA

PHD THESIS

EFFICIENT BYPASS MECHANISMS FOR LOW LATENCY NETWORKS ON-CHIP

Realizada por: **Iván Pérez Gallardo**

Dirigida por: **Julio Ramón Beivide Palacio**

Enrique Vallejo Gutiérrez

Escuela de Doctorado de la Universidad de Cantabria

Santander 2021

Resumen

Hace casi dos décadas, se produjo un cambio de paradigma en el diseño de procesadores por la aparición del *Power Wall*. El *Power Wall* es la creciente dificultad para disipar la potencia consumida por los procesadores dado el rápido aumento de su frecuencia de operación, gracias al proceso de miniaturización de los transistores. Este efecto, desencadenó el tránsito de procesadores secuenciales *mono-core* a procesadores paralelos *multi-core*. Desde entonces, el desarrollo de procesadores *multi-core* no ha parado de crecer y en la actualidad podemos encontrar procesadores comerciales con varias decenas de *cores*, comúnmente denominados *many-core*. Para poder aprovechar la capacidad de procesamiento paralela de estos procesadores, es necesario introducir una serie de elementos adicionales. Estos elementos están relacionados principalmente con el sistema de memoria y son la implementación de instrucciones de acceso a memoria atómicas, un modelo de consistencia y un protocolo de coherencia. Para habilitar la comunicación entre *cores*, implícita en los accesos a memoria de datos compartidos, los procesadores *multi-core* disponen de una red de interconexión dentro del chip o *Network on-Chip* (NoC). Estas redes comenzaron siendo simples buses de datos, dado el bajo número de cores a interconectar, pero poco a poco han ido siendo reemplazados por otras redes con mejor escalado en ancho de banda. En un primer paso los buses fueron reemplazados por anillos y cada vez es más frecuente el uso de mallas a medida que va creciendo el número de cores a interconectar.

En un procesador *many-core*, con decenas de *cores*, es esencial que la latencia de la NoC sea lo más baja posible, ya que esta constituye un elemento más dentro de la jerarquía de memoria y por tanto su latencia afecta al tiempo medio de acceso a memoria o *Average Memory Access Time (AMAT)*. De esta forma, los criterios que se tienen en cuenta a la hora de escoger una NoC habitualmente son que tenga el suficiente ancho de banda para que la red opere fuera de la zona de saturación, mientras se minimiza la latencia base de los paquetes y el coste de la propia red.

Basándonos en nuestras observaciones a partir de ejecuciones de aplicaciones reales en simuladores *Full-System* (FS), el tráfico generado por los cores tiene un volumen muy bajo en promedio. Como consecuencia, la latencia está determinada fundamentalmente por dos factores: la latencia de atravesar un *router* de la NoC y la distancia entre nodos que determina el número de saltos (*routers* atravesados) a realizar por los paquetes.

Las soluciones tradicionales para minimizar estos dos factores consisten en, por una parte diseñar *routers* de baja latencia o bien implementar topologías de alto grado, es decir con baja distancia máxima. Sin embargo, ambas soluciones presentan problemas que las hacen no recomendables para su uso en NoCs. La primera, diseñar *routers* de baja latencia, implica típicamente diseñar *routers* mono-etapa, es decir, sin segmentar el camino de datos mediante un *pipeline*. Esto puede implicar diseñar *routers* con un tiempo de ciclo muy grande, o con arquitecturas muy simples para lograr la frecuencia de operación deseada lo que repercute negativamente en el ancho de banda. La segunda, diseñar topologías de alto grado, suele implicar la implementación de *routers* con un gran número de puertos para poder establecer los enlaces necesarios. Sin embargo, aumentar el número de puertos significa replicar la lógica de cada puerto, así como incrementar la complejidad de los *allocators* (lógica de asignación de recursos) y el *corssbar* de los *routers*.

Es aquí donde entran en juego las NoC con *bypass*, las cuales son el tema central de esta tesis. Los *routers* con *bypass* son una solución eficiente para reducir la latencia de los *routers* en topologías simples como la malla. La idea fundamental de esta clase de

NoC consiste en saltar algunas etapas del pipeline de los *routers*, reduciendo el número de ciclos por salto. Para ello se pre-asigna el *crossbar* de los *routers* antes de recibir los paquetes, para que estos avancen directamente al *crossbar* del *router*, evitando los *buffers* de entrada. Esto permite ahorrar algunos ciclos en caso de que la pre-asignación sea satisfactoria, sin limitar la frecuencia de operación por la falta de un pipeline y sin degradar el ancho de banda máximo. Dentro de los *routers* con *bypass* distinguimos entre dos grupos: *single-* y *multi-hop bypass*. Por una parte, *single-hop bypass* es la propuesta inicial de *router* con *bypass* y la denominamos así porque trata de pre-asignar únicamente el *crossbar* del siguiente *router* en la ruta del paquete. Por otra parte, *multi-hop bypass* es una propuesta posterior presentada con el nombre de SMART. En este tipo de NoC los paquetes pueden solicitar la pre-asignación de los *crossbars* de múltiples *routers* en su ruta para después atravesarlos en un único ciclo, lo que denominamos como *multi-hop*. Volviendo a los dos factores que determinan la latencia en este tipo de NoCs, *single-hop bypass* reduce la latencia de los *routers*, mientras que *multi-hop bypass* reduce el número efectivo de saltos realizados por los paquetes.

Si bien este tipo de solución es una forma eficiente de reducir la latencia, la incorporación del *bypass* introduce una serie de conflictos adicionales que hay que resolver para evitar posibles *deadlocks*. Las propuestas originales, tanto para *single-* como *multi-hop bypass* implementan duras restricciones al control de flujo, permitiendo que los paquetes únicamente utilicen el *bypass* cuando los *buffers* a evitar están vacíos. *Multi-hop bypass* requiere adicionalmente que el *buffer* del siguiente *router* esté vacío a la hora de retransmitir paquetes. Esto conlleva la necesidad de usar configuraciones con varios canales virtuales o *Virtual Channels* (VC), para sacar provecho de los caminos de *bypass* y del ancho de banda que ofrece la topología. El uso de VCs es algo muy extendido en redes por diferentes motivos como pueden ser reducir *Head of Line Blocking* (HoLB), implementar ciertos algoritmos de enrutamiento o evitar *deadlocks*, por ejemplo. Sin embargo, su implementación introduce costes considerables y a medida que se incrementa el número de VCs se hace más compleja la lógica de control y asignación de los mismos.

Las contribuciones de esta tesis se centran en desarrollar mecanismos de *bypass* eficientes que relajen las condiciones necesarias para la transmisión de paquetes. A continuación se resumen las cuatro contribuciones de esta tesis:

BST

La evaluación y análisis de NoCs, y en general en arquitectura de computadores, se basa en técnicas de simulación en sus etapas iniciales debido a los altos costes y tiempo necesario de desarrollo de prototipos. Dentro de este contexto, *Bypass Simulation Toolset* (BST) es un conjunto de herramientas *open-source* para la simulación de NoCs con *bypass*. Dada la falta de simuladores de NoC con este tipo de *router*, desarrollar estas herramientas ha sido clave durante el transcurso de la tesis para evaluar el resto de contribuciones de la misma. BST esta compuesto por 4 elementos:

1. Una versión personalizada de **BookSim** que incluye una gran variedad de modelos de *router* con *single-* y *multi-hop bypass*. BookSim es un simulador funcional *open-source* con un modelo de *router* tradicional pero detallado y preciso. Este tipo de simulación es ideal para realizar un gran volumen de experimentos en un plazo de tiempo relativamente corto, así como facilitar el desarrollo de nuevas ideas y su depuración, gracias a su nivel de abstracción. Los modelos de NoCs con *bypass* incluidos en nuestra versión, implementan con detalle el pipeline y han sido validados frente a diseños reales HDL, demostrando que son precisos a nivel de ciclo.

2. Una versión actualizada de **OpenSMART** que incluye implementaciones de SMART++, y S-SMART++ (contribuciones de la tesis). OpenSMART es un diseño HDL *open-source* de una red SMART, *multi-hop bypass*, escrita en Bluespec System Verilog. Este tipo de diseño tiene gran valor en las etapas intermediadas y finales del desarrollo ya que por una parte permite validar los modelos funcionales, y por otra sintetizar en FPGAs o crear prototipos para estimar área, potencia y la frecuencia máxima de operación entre otras cosas.

3. Un *Application Program Interface* (**API**) para integrar BookSim en simuladores *Full-System* (FS). Las simulaciones FS, simulan el sistema completo como indica su nombre. Esto se traduce en que se simula en detalle todos los componentes de un procesador como son los cores y la memoria entre otros, pero también el sistema operativo donde se ejecutan las aplicaciones a evaluar. En lo que respecta a la evaluación de NoCs, esto es de gran utilidad para alimentar la red con tráfico real, es decir tráfico generado a partir de los fallos de cache derivados de la ejecución de aplicaciones. Con esta API proporcionamos un complemento al tráfico sintético de BookSim, útil para evaluar de manera rápida y efectiva las NoCs, pero alejado de las características estrictas que podemos encontrar en un sistema real.

4. Un conjunto de ***scripts*** para facilitar ciertas tareas como la creación de experimentos, la generación de gráficas de resultados o la depuración de los modelos de NoC. Los *scripts* más relevantes son los orientados a la creación de experimentos y generación de gráficos dado que son los usados con mayor frecuencia. Estos *scripts* proporcionan bastante flexibilidad al usuario y han sido utilizados para generar los resultados y gráficas de esta tesis.

NEBB

Como se ha mencionado anteriormente, las propuestas de *router* con *bypass* originales tienen unas restricciones muy estrictas a la hora de transmitir paquetes. Esto se debe a que al introducir el *bypass* pueden entremezclarse dos paquetes en el mismo *buffer* provocando comportamientos inesperados o corrompiendo datos. Para evitarlo, estas propuestas necesitan que los *buffers* de destino estén vacíos a la hora de enviar los paquetes.

Non-Empty Buffer Bypass (NEBB) [Perez2018; Perez2020a] es una propuesta para aliviar estas restricciones en *routers single-hop bypass*. Como su nombre indica, NEBB permite el *bypass* aunque los *buffers* a saltar no estén vacíos aumentando las oportunidades para transmitir paquetes y tomar el *bypass*. Proponemos tres variantes dependiendo de las reglas establecidas para el uso del *bypass*:

- **NEBB-WH**: esta versión está pensada para su uso en redes *WormHole* (WH), cuya característica principal es que el arbitraje y asignación de los puertos del *router* se realiza *flit* a *flit*[1]. Los *flits* pueden ser enviados al siguiente *router* siempre y cuando haya un VC libre con al menos hueco para un *flit*. Al igual que en la propuesta original, cualquier paquete puede tomar el *bypass* de un *router* cuando su *buffer* de entrada está vacío. Sin embargo, NEBB-WH también permite que los paquetes de un único *flit* puedan tomar el *bypass* cuando el *buffer* a evitar no está vació.

- **NEBB-VCT**: está versión se basa en *Virtual Cut-Through* (VCT), donde la asignación de los puertos del *router* se hace paquete a paquete. Los *routers* pueden

[1] El *flit*, *flow control unit*, es la unidad mínima de control de flujo en la que se fragmentan los paquetes de red. En NoCs suelen tener típicamente un tamaño igual a la anchura de los enlaces.

mandar paquetes al siguiente *router* siempre y cuando haya un VC con hueco para el paquete completo. En este caso no importa la ocupación del *buffer* que se pretende evitar ya que el puerto de salida se mantiene asignado para todo el paquete.

- **NEBB-Hybrid**: esta variante combina las dos anteriores para maximizar las posibilidades de *bypass* cuando el control de flujo es WH, es decir, *flit* a *flit*. En este caso, los *routers* pueden transmitir los paquetes siempre y cuando haya un VC libre con al menos hueco para un *flit* en el siguiente *router*. En canto al *bypass*, el único caso a destacar es en el que un paquete con varios *flits* trata de tomar el *bypass* en un *router* con el *buffer* de entrada ocupado. En esta situación se necesita espacio para todo el paquete en el siguiente *router* y el puerto de salida queda bloqueado para el paquete, hasta que el *flit* de cola de éste no es retransmitido.

Además de incrementar la utilización del *bypass*, NEBB permite usar mecanismos originalmente propuestos para *routers* tradicionales dado que su control de flujo es similar. En [Perez2020a] presentamos como ejemplo la adaptación de *Flit Bubble Flow Control* (FBFC) [Ma2015] a este tipo de *routers* para evitar *deadlock* en toros, lo que supone una alternativa más eficiente a los *routers* con *bypass* estándar en toros basados en Dateline [Dally2003].

Los resultados obtenidos de la evaluación muestran que NEBB, y especialmente NEBB-Hybrid, aumenta el uso del *bypass* a cargas medias-altas, lo que reduce la latencia y la potencia consumida. La reducción de potencia se debe a la baja utilización de *buffers* de los *routers* ya que son uno de los elementos que más energía consumen. Además NEBB es capaz de conseguir el mismo rendimiento que las propuestas originales, pero usando configuraciones más austeras con la mitad de tamaño de *buffer* y sin necesidad de usar VCs.

SMART++

SMART++ es una versión mejorada de SMART que utiliza de forma eficiente los *buffers* y el *bypass* de los *routers* de la NoC. Al igual que ocurre con los *routers single-hop bypass*, SMART únicamente retransmite paquetes cuando los *buffers* del siguiente *router* están vacíos. Por su parte, SMART++ elimina estás restricciones aplicando los siguientes tres mecanismos sobre SMART:

1. *Multi-Packet Buffers* (MPB): permite que los *buffers* acumulen *flits* de diferentes paquetes. Esta es una técnica muy común en *routers* tradicionales que permite retransmitir paquetes mientras haya espacio en los *buffers* del siguiente router. Dependiendo del control de flujo implementado, el espacio requerido es para un flit (WH) o un paquete completo (VCT). Pero, al usar esta técnica en *routers* con *multi-hop bypass*, hay que añadir restricciones adicionales al asignar el *bypass*, como comprobar que los *buffers* sobre los que se toma el *bypass* estén vacíos.

2. *NEBB*: al aplicar NEBB, inicialmente NEBB-WH porque el arbitraje es *flit* a *flit*, permitimos que paquetes de un único *flit* puedan tomar el *bypass* independientemente de la ocupación de los *buffers* sobre los que actúa el *bypass*.

3. *Packet-by-Packet Arbitration* (PPA): este es el tipo de arbitraje usado en VCT. Como se ha mencionado anteriormente SMART usa arbitraje *flit* a *flit* lo que puede provocar que los *flits* de paquetes *multi-flit* no se retransmitan en ciclos consecutivos. Al usar PPA evitamos este problema, permitiéndonos aplicar NEBB-VCT, lo

que a su vez permite que paquetes con múltiples *flits* puedan tomar el *bypass* independientemente de la ocupación del *buffer*, siempre y cuando el *buffer* del siguiente *router* tenga espacio para todo el paquete. En el caso de *multi-hop bypass*, puede ocurrir que un paquete adquiera el *bypass* pero que el paquete correspondiente no llegue a usarlo, debido a que el paquete se ha detenido en un *router* previo. Por ello, el *bypass* sólo se bloquea para el paquete si llega el *flit* de cabecera y se libera al pasar el *flit* de cola.

Los resultados muestran que SMART++ no necesita VCs para obtener el mismo rendimiento de SMART, o incluso mejor. El no usar VCs reduce drásticamente la lógica necesaria para su gestión, lo que a su vez mejora considerablemente el consumo energético y el camino crítico de los *routers*.

S-SMART++

Speculative-SMART++ (S-SMART++) es un diseño de NoC *multi-hop bypass*, orientado a reducir los dos factores que afectan a latencia mencionados anteriormente: la *latencia* de los *routers* y el número de saltos. SMART reduce el número de saltos efectivo en topologías sencillas como la malla al permitir que los paquetes puedan hacer múltiples saltos en un único ciclo. Sin embargo, después de cada salto, los paquetes son almacenados en los *buffers* y tienen que pasar por todo el *pipeline* de los *routers*, lo que requiere como mínimo 3 ciclos en SMART o SMART++.

Para reducir la latencia de los *routers* después de cada *multi-hop*, S-SMART++ hace uso de la especulación, suponiendo que los paquetes van a completar todos los saltos asociados a un *multi-hop*, para solicitar la preparación del siguiente *multi-hop* antes de que se reciba el paquete. En caso de que la especulación acierte, únicamente transcurre un ciclo entre que el paquete es recibido y es retransmitido, en vez de los tres ciclos de SMART. En caso de fallar, no hay ninguna pérdida de rendimiento ya que a este tipo de solicitud tiene menos prioridad que las peticiones normales.

Los resultados experimentales obtenidos, muestran que S-SMART++ reduce la latencia base notablemente llegando a estar muy cerca del rendimiento de SMART_2D. SMART_2D es la versión de SMART con menor latencia, pero a cambio de incrementar drásticamente la lógica de control y arbitraje. Y más importante aún, S-SMART++ reduce muy significativamente la dependencia de la latencia con el número máximo de saltos que se pueden realizar dentro de un *multi-hop*, factor denominado *Maximum Hops Per Cycle* (HPC_{Max}). HPC_{Max} es un parámetro de diseño fundamental para las redes SMART. El reducir la dependencia de este factor con la latencia flexibiliza los parámetros de diseño a la hora de integrar este tipo de redes en sistemas reales.

Por último, también se ha demostrado que gracias al uso de un control de flujo tradicional como es VCT permite que tanto SMART++ como S-SMART++ utilicen mecanismos originalmente pensados para NoCs tradicionales. En este caso, nos centramos de nuevo en la evitación de *deadlocks* en el toro, mostrando que tanto SMART++ como S-SMART++ son compatibles con *Bubble Routing* [Carrion1997], un mecanismo eficiente que no requiere de VCs y no introduce asimetría en el uso de los *buffers*.

Abstract

Computer architecture has taken advantage of the technology process miniaturization and frequency scaling to build more complex architectures with higher performance. The emergence of the Power Wall and the higher difficulty of extracting performance from sequential processors was the turning point in computer architecture in favor of multi-core processors. This change affected fundamentally the memory sub-system in the hardware architecture part, requiring additional mechanisms to share data between cores. One key feature is a new interconnection layer to enable the communication between cores. The demands for these interconnections grow with the number of cores, requiring more bandwidth while constraining costs and latency, as they are part of the memory hierarchy. For this reason, there has been a growing interest in using Networks on-Chip (NoCs) to interconnect cores for the last 15 years, as they are relatively easy to implement in tile-based arrangements. Nowadays, commercial processors are replacing traditional buses and rings with meshes, as they are NoCs topologies that scale better.

Nevertheless, meshes have an important issue: their latency grows with the number of nodes at a relative fast pace, and it can become comparable to the access time of the last-level cache. In a multi- or many-core CPU, the traffic generated by the cores is typically very low as most of the requests are filtered by the private caches, and only misses (including coherence misses) produce data requests to the shared levels that travel through the network. Thus, the main factors that determine the latency of packets in a NoC are the delay of the routers and the distance between nodes. The router delay is defined by the router architecture, which is typically implemented in a pipeline to increase frequency. The distance between nodes is mostly defined by the topology, because this kind of networks commonly use minimal routing. The most common solutions to minimize latency consist in simplifying the router architecture to minimize the pipeline stages of the router or use high-degree topologies to reduce the distance between nodes. However, the former usually limits the performance of the network in terms of bandwidth, and the latter implies high-radix routers, which shoot up costs.

In this thesis we focus on NoCs with bypass routers, which reduce latency while maintaining the low costs of mesh-like topologies. The idea of bypass routers is skipping pipeline stages to reduce latency by pre-allocating the switch of the routers before the arrival of packets. We distinguish between two types of bypass mechanisms: single-hop and multi-hop bypass. Single-hop bypass focuses on minimizing the delay of routers by skipping allocation stages in each hop. Multi-hop bypass focuses on minimizing the effective number of hops (topological distance) by pre-allocating and traversing multiple routers at a time to traverse (bypass) multiple routers in a single cycle. Our contributions to bypass routers are the following.

Starting with single-hop bypass, we analyze the implications of introducing a bypass path in the routers and determine that the original conditions to use the bypass are unnecessarily conservative. We propose three alternative implementations called Non-Empty Buffer Bypass (NEBB), that increase the bypass utilization to maximize its benefits in terms of latency and power. The most advanced NEBB mechanism, called NEBB-Hybrid, is able to match the performance of the baseline without using VCs, the baseline requires numerous VCs, and with half the total buffer space.

Following with multi-hop bypass, we perform the same analysis for the bypass conditions and flow control mechanism of SMART, the original multi-hop bypass proposal. We propose an improved version of SMART called SMART++ that combines three mecha-

nisms: multi-packet buffers, NEBB and packet-by-packet arbitration. SMART++ removes the requirement of using multiple VCs of SMART and takes advantage of the topology's bandwidth, without degrading performance. Thus, SMART++ enables low cost configurations with few or no VCs giving better performance than SMART.

The third contribution, called S-SMART++, combines the underlying ideas of single- and multi-hop bypass to reduce both the delay of routers and the effective number of hops. S-SMART++ stands for Speculative-SMART++ because it relies on speculative allocation of multi-hop bypass paths to minimize the router delay after each multi-hop. In this way, S-SMART++ almost reaches the latency of SMART_2D, the most performant but costly version of SMART, without incurring in its costs. More importantly, S-SMART++ reduces the dependency of the latency with the maximum length of multi-hops, relaxing the requirements to integrate multi-hop bypass in real designs.

The final contribution is a set of tools to simulate bypass NoCs called Bypass Simulation Toolset (BST). BST is open-source and is mainly based on BookSim, a functional NoC simulator, and OpenSMART, an HDL SMART design. BST contains most of the tools that we have used to evaluate our proposals, including the aforementioned contributions. Besides the custom versions of BookSim and OpenSMART, BST includes an API to integrate BookSim in Full-System simulators such as gem5 and a set of scripts to facilitate the generation of experiments and result charts.

In summary, these contributions extend bypass routers with more efficient, performant, and diverse architectures, which may help to address latency and power consumption challenges in the future.

Acknowledgments

First and foremost, I would like to acknowledge my advisors, Ramón Beivide and Enrique Vallejo, for their support and guidance through these years. It has been a privilege to learn from your vast knowledge and experience, not forgetting all the opportunities you have given me to expand my knowledge, experiences, and skills. Most of all, I am grateful for the friendly atmosphere. I have felt like at home.

I am grateful to all the members of the Computer Architecture and Technology group of the University of Cantabria for providing a stimulating and fun work environment for the past six years. We have shared a lot of experiences, and I have learn a lot from you. Special thanks to Esteban Stafford for the English revision of this thesis. For my colleges at the office: Borja, we have traveled a lot together; Cristobal, thank you for enduring my silly questions; Mariano, we have some (virtual) car races pending; Pablo, thanks for your guidance and our chit-chat sessions; Raúl, we need a regressive timer for future conversations...

I want to thank also my hosts during my internships in Barcelona Supercomputing Center and ARM: Miquel Moretó, Marc Casas, and Roxana Rusitoru. I do not have enough space to write the names of all my colleagues during practice, but I sincerely appreciate your hospitality. However, I want to make a special mention to Emilio and Adrián for all the time we spent together.

Finally and most importantly, I thank my family back home for their unconditional support and encouragement. Without you, this long journey would not have been possible. I hope to have your support during the next ones.

This work was supported by the Spanish Ministry of Science, Innovation and Universities, FPI grant BES-2017-079971, and contracts TIN2010-21291-C02-02, TIN2013-46957-C2-2-P), TIN2015-65316-P, TIN2016-76635-C2-2-R (AEI/FEDER, UE) and TIC PID2019-105660RB-C22; the European HiPEAC Network of Excellence; the European Community's Seventh Framework Programme (FP7/2007-2013), under the Mont-Blanc 1 and 2 projects (grant agreements n° 288777 and 610402); the European Union's Horizon 2020 research and innovation programme under the Mont-Blanc 3 project (grant agreement n° 671697). Bluespec Inc. provided access to Bluespec tools.

Para mis padres, María Jesús y Benicio, por todo su apoyo y sacrificio.

Contents

List of Tables

List of Figures

Acronyms

ALM	Adaptive Logic Module
ALUT	Adaptive Look-Up Table
AMAT	Average Memory Access Time
AMBA	Advanced Micro-controller Bus Architecture
BB	Bisection Bandwidth
BST	Bypass Simulation Toolset
BSV	Bluespec System Verilog
BW	Bisection Width
BW	Buffer Write
CAS	Compare-And-Swap
CMP	Chip-MultiProcessor
CPU	Central Processing Unit
CSV	Comma Separated Values
DAMQ	Dynamically-Allocated Multi-Queue
DOR	Dimension Order Routing
DSM	Distributed Shared Memory
EBB	Empty Buffer Bypass
EVC	Express Virtual Channels
EVCF	Empty VC Forwarding
FBFC	Flit Bubble Flow Control
FBFC-L	Flit Bubble Flow Control Localized
FBFLY	Flatted Butterfly
FPGA	Field-Programmable Gate Array
FS	Full-System
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HoLB	Head-of-Line Blocking
HPC	Hops Per Cycle
HPC	High Performance Computing
ILP	Instruction Level Parallelism
IQ	Input Queue
ISA	Instruction set architecture
LA	LookAhead
LA CC	LookAhead Conflict Check
LA-Arb	LookAhead Arbiter
LA-Gen	LookAhead Generator
LA-LT	LookAhead Link Traversal
LA-RC	LookAhead Route Computation
LL/SC	Load-Linked/Store-Conditional
LT	Link Traversal
MIMD	Multiple Instructions Multiple Data
MPB	Multi-Packet Buffer
MPI	Message Passing Interface
MUSA	MUlti-level SimulAtion methodology
NEBB	Non-Empty Buffer Bypass

NEBB-VCT	Non-Empty Buffer Bypass - Virtual Cut-Trough
NEBB-WH	Non-Empty Buffer Bypass - WormHole
NIC	Network Interface Controller
NoC	Network on-Chip
NUCA	Non-Uniform Cache Access
O3	Out-Of-Order
OS	Operative System
PPA	Packet-by-Packet Arbitration
pthreads	Poxis Threads
QoS	Quality of Service
RC	Routing Computation
ROI	Region Of Interest
RR	Round-Robin
RTL	Register Transfer Logic
S-SMART	Speculative-SMART
SA	Switch Allocation
SA-G	Switch Allocation Global
SA-I	Switch Allocation-Input
SA-L	Switch Allocation Local
SA-O	Switch Allocation-Output
SC	Sequential Consistency
SMART	Single-cycle Multi-hop Asynchronous Repeated Traversal
SMP	Symmetric Multi-Processors
SN	Slim NoC
SoC	System on-Chip
spec-SSR	speculative Switch Setup Request
SSR	Switch Setup Requests
ST	Switch Traversal
TFC	Token Flow Control
TLP	Thread Level Parallelism
TSO	Total Store Ordering
VA	Virtual channel Allocation
VC	Virtual Channel
VCD	Value Change Dump
VCT	Virtual Cut-Through
VN	Virtual Networks
VS	Virtual channel Selection
WH	WormHole
WiNoC	Wireless Network on-Chip
WPF	Whole Packet Forwarding

Chapter 1

Introduction to Networks on Chip

Networks on-Chip (NoCs) are a fundamental part of multi-core processors. Research in NoCs has been growing for the last 15 years and it is expected to be more relevant in the future due to their impact on the overall performance and power consumption of processors.

This chapter starts with an introduction to the evolution of commercial multi-core processors to motivate the importance of NoCs (Section 1.1). It continues introducing some computer architecture concepts that are relevant in NoC design (Sections 1.2 and 1.3). The main part summarizes the main NoC concepts and terminology relevant to the contributions of this thesis (Section 1.4), and motivates the utilization of bypass routers to improve performance and power consumption in future Chip-MultiProcessors (CMPs) (Section 1.5). The final part describes the organization of the reminder of the thesis (Section 1.6).

1.1 Introduction

CPU performance has been characterized by a fast evolution during the last 5 decades in a large extent due to steady advances of semiconductor technology. Each generation of semiconductors has reduced the feature size, leading to improvement transistor speed and efficiency, as well as integration density. As a result, CPU performance improvements were mainly a consequence of higher clock frequencies, wider data paths, cache memories to overcome the Memory Wall, and more complex mechanisms to increase the instruction execution rate through Instruction Level Parallelism (ILP). Some examples of these mechanisms are branch prediction, superscalar processors or out-of-order execution. However, increasing the operation frequency significantly has not been possible since the Dennard scaling [Dennard1974] breakdown, around 2005. The Dennard scaling states that the power density of transistors stay constant when reducing their size. This effect, in combination with the frequency scaling derived from reducing the size of transistors, involves power consumption increments that lead to the Power Wall, where increasing the frequency in not possible due to power dissipation issues. The combination of the stagnation in frequency scaling and the diminishing returns of ILP (ILP Wall) made CPU designers focus on Thread Level Parallelism (TLP) with CMPs.

Since the appearance of CMPs, the growth of the transistor integration density has mainly been used to increase the number of processing units. With more concurrent

processing units, parallel computing has become the dominant programming paradigm, increasing software and hardware complexity. Figure 1.1 shows how the number of cores in CMPs have grown since 1999. The trend line shows that the number of cores approximately duplicates every 4 years.

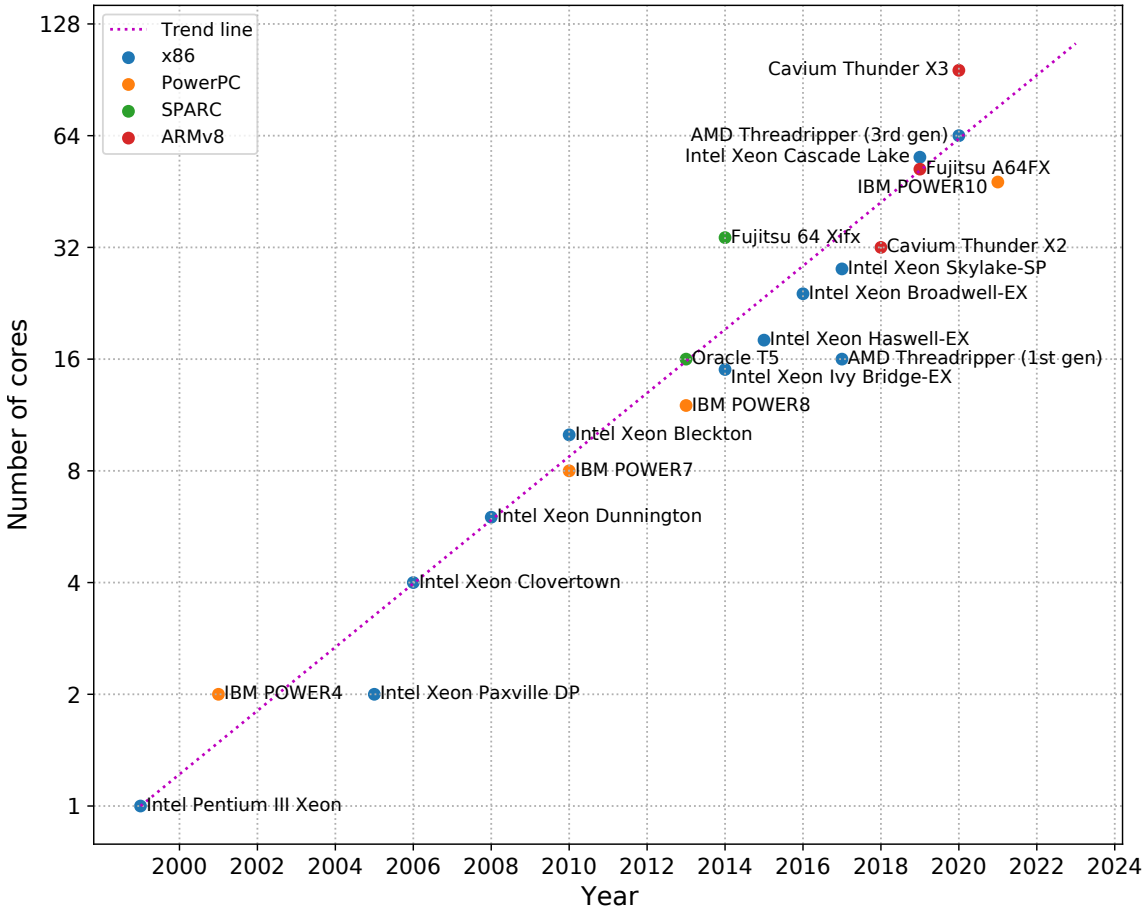


Figure 1.1: Evolution of high performance CMPs since 1999.

This type of product has mainly been guided by the demands of the server and supercomputing markets. Supercomputers have used CMPs since their arrival as they perfectly fit their philosophy of massive parallelism. Some historical top-1 supercomputers of the TOP500 list [Strohmaier2020] are: IBM's Blue Gene/L [Gara2005], listed in 2004 with PowerPC CMPs of 2-cores; Cray's Jaguar XT5 [Bland2009], listed in 2009 with 6-core AMD Opteron 2435 CMPs; Fugitsu's K Computer [Miyazaki2012], listed in 2011 with SPARC64 VIIIfx CMPs [Yoshida2012] of 8-cores; Sunway TaihuLight [Fu2016], listed in 2016 with Sunway's SW26010 CMPs of 256 lightweight compute cores plus 4 auxiliary cores; IBM's Summit [Kahle2019], listed in 2018 with POWER9 [Sadasivam2017] CMPs with 22-cores. More recently, supercomputers with ARM-based CMPs have broken in the market. For example, Cray's XC50 supports Cavium ThunderX2 processors. And the latest Fujitsu's supercomputer, the Fugaku, has become the first ARM-based supercomputer that leads the TOP500 list in November 2020. This supercomputer uses Fujistu's 48-core A64FX CMPs [Yoshida2018] for a total of 7,299,072 cores and 415,530.0 TFlops/s. It is nevertheless noteworthy that the MontBlanc projects [Rajovic2016] are precursors of ARM-based supercomputers.

1.2 Chip-Multiprocessors (CMPs)

According to Flynn's taxonomy [Flynn1972], CMPs are Multiple Instructions Multiple Data (MIMD) processors. Dealing with MIMD processors that exploit TLP parallelism is not easy and complicates the design of both software and hardware. This section summarizes the software implications (Section 1.2.1), the hardware organization of CMPs (Section 1.2.2) and the architectural support to use parallel programming (Section 1.2.3).

1.2.1) Software implications

In the software layer, the easiest way to exploit TLP is to run multiple independent tasks in different hardware threads (or cores) of the processor, which is known as multi-programming. Our daily basis multi-task Operative System (OS) uses multi-programming to distribute the active programs among the threads of the processors. This increases the responsiveness compared to running all the processes in a single core using time-sharing.

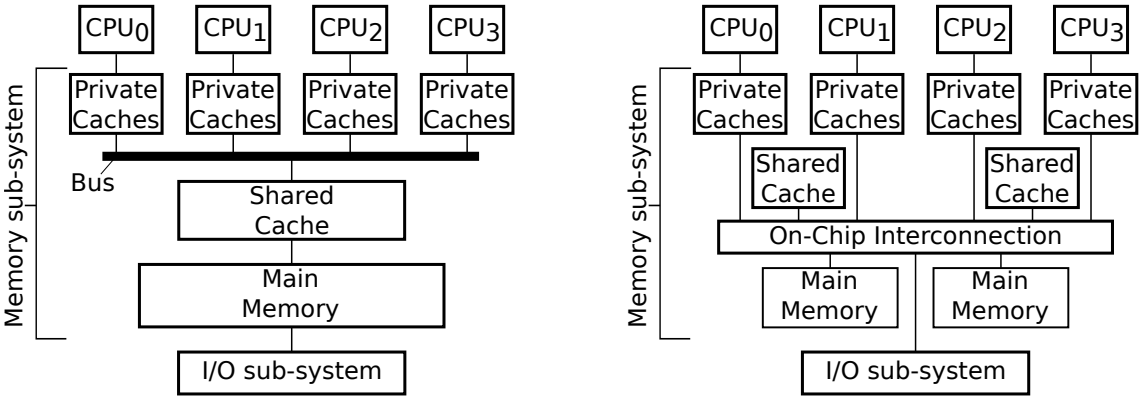
However, exploiting TLP to increase the performance of individual programs is more complicated. Programmers have to explicitly define the parallelism in the application code, which involves the use of synchronization and communication routines to manipulate shared data between threads. There are numerous programming models that abstract, to a greater or lesser degree, the complexity of parallel software development. The most well-known model is Posix Threads (pthreads) [Butenhof1997] that implements a set of mechanisms to synchronize and manage logical threads that are mapped to the physical threads of the CMP. Other popular models are OpenMP [Chandra2001; Chapman2008] and OmpSs [Duran2011] which offer similar mechanisms but at a higher abstraction level. One of the key features of OpenMP is the automatic parallelization of regular loops. Regarding OmpSs, its abstraction allows defining tasks and data dependencies between them. Then, a runtime is in charge of monitoring the dependencies and scheduling the tasks for their execution in the cores of the CPU, as well as other devices like GPUs, accelerators, FPGAs, etc. This programming paradigm is commonly known as task-based parallelism. OpenMP also includes this paradigm since version 3.0. Another well-known programming model is the Message Passing Interface (MPI) [Gropp1999], which defines a communication protocol between a set of processes. MPI is widely extended in supercomputing as it enables the communication between compute nodes in distributed-memory parallel machines, but it can be used to exploit TLP in CMPs as well [Krawezik2003; Graham2008].

1.2.2) Hardware organization

CMPs are basically processors with multiple CPUs, each with one or more levels of private cache, and generally a shared level of cache besides main memory. There are two classes of shared-memory multiprocessors. The first class is known as centralized shared-memory processors because they share one or more centralized levels of memory. They are also known as Symmetric Multi-Processors (SMP) because their shared memory levels provide uniform memory access for all the processors.

Figure 1.2a shows the basic organization of a SMP. SMPs generally have a few cores since centralized resources, such as the shared level of cache, become a bottleneck if the number of cores grows excessively. Buses or crossbars are typically used to connect and

arbitrate the access to centralized resources. The second CMP class is called Distributed Shared Memory (DSM) multi-processors. In this case, the shared cache is distributed into multiple banks to increase the available bandwidth, which allows the interconnection of more processors. An example of a DSM organization is depicted in Figure 1.2b. The distribution of shared caches in banks typically uses set-interleaving. Set-interleaving uses the set index bits of a given address to map addresses to banks. The set index bits are the lowest bits in the address after the offset bits that address within the memory blocks. Therefore, contiguous memory blocks are assigned to different cache banks. Thus, accesses from different private caches are evenly distributed to avoid hotspots. As shown in Figure 1.2b, it is common that the on-chip interconnect also connects the shared cache with main memory. Nonetheless, distributing the cache complicates data communication and may follow a Non-Uniform Cache Access (NUCA) architecture [ChangkyuKim2003], i.e., the access time depends on data location.



(a) SMP. ***(b)*** DSM.

Figure 1.2: Basic organization of a Symmetric Multi-Processor (SMP) and Distributed Shared Memory (DSM) multi-processor according to [Hennessy2019].

1.2.3) Architectural support for parallel programming

Almost all CMP designs are provided with a set of atomic instructions, a consistency model and a cache coherence protocol. The design of each element depends on the design of the rest and has an important influence on the behavior and performance of the memory sub-system.

First, atomic instructions, like load-link/store-conditional (LL/SC) or compare-and-swap (CAS), are used to implement thread synchronization routines needed to develop multi-threading programs.

Second, the consistency model specifies a set of rules for memory operations to ensure the correct global order of reads and writes to shared memory addresses. The simplest example of consistency model is Sequential Consistency (SC), in which all the memory operations are executed following the program order. A more relaxed example is Total Store Ordering (TSO) or processor consistency, which can be found in x86 processors and enforces the sequential execution of stores. The most aggressive scheme called Relaxed Consistency allows the reorder of almost any memory operation, requiring the use of fences to synchronize and order memory operations. This increases the implementation flexibility of synchronization routines, which maximizes the potential benefit of out-of-order execution. This type of model is used in ARMv8 and RISC-V processors.

Third, cache coherence protocols are in charge of maintaining the coherence among the multiple copies of shared data in different caches. In other words, if a thread modifies the value of a shared data, the cache coherence protocol is in charge of updating and notifying modifications to the rest of the sharers. The cache coherence protocol is key in how on-chip communication takes place trough the memory sub-system elements.

1.3 Cache coherence protocols

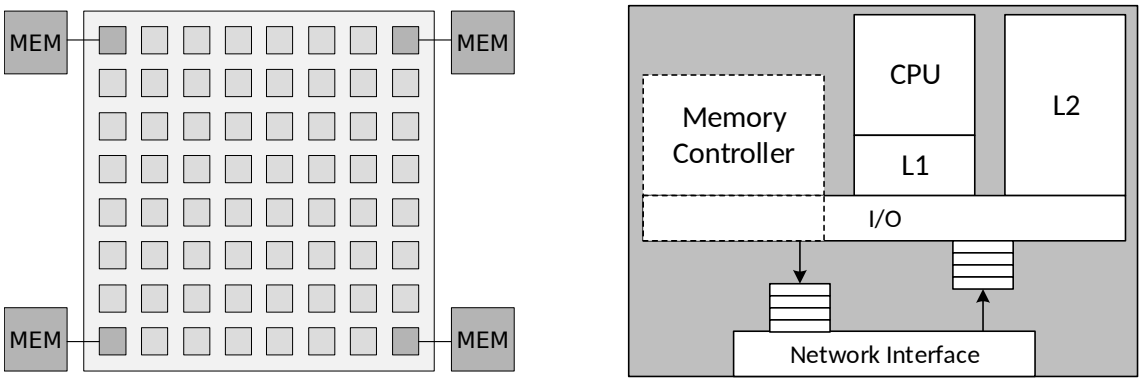
Cache coherence protocols are bound to NoCs as they define how is the communication between the elements that conform the memory hierarchy. There are two classes of cache coherence protocols that will determine how the communication interconnect is implemented [Nagarajan2020]: snooping and directory-based protocols.

Snooping protocols: are those where private caches monitor their cached memory addresses to detect changes in their data values. They rely on broadcast communication. For this reason, the interconnects usually are point-to-point networks (dedicated wires), or shared-medium interconnects such as buses or crossbars. This kind of design is simple but lacks scalability. For example, in point-to-point networks the number of wires grows quadratically with the number of interconnected elements, becoming prohibitive. Traditional buses have low bandwidth because only one controller can use the medium at a time, becoming a bottleneck when interconnecting many cores. Crossbars solve the bandwidth problem of buses, but they have a high growth ratio in terms of footprint area and power consumption.

Directory-based protocols: rely on monitoring the state of data in directories, which are typically distributed among the shared cache banks. Like with the distributed shared-memory banks, this increases the number of operations at a time, overcoming the lack of scalability of snoopy-based protocols. In this kind of protocol, the data modification in a cached address is notified to the directory that tracks its state. Directories are responsible for sending invalidation or update messages with the new values to the rest of sharers. However, this introduces overhead in terms of the number of messages in the communication sequences. DSM multi-processors typically use this type of protocol.

In this case, the typical interconnect is a Network on-Chip (NoC) instead of a bus or crossbar. NoCs are switched-media that scale in terms of throughput, area footprint and power. By multiplexing the communication, they achieve high bandwidth while making an efficient use of the wires. One of the key characteristics of NoCs for DSM multi-processors is that they fit well in their tile-based organization with directory-based protocols. In this type of organization each tile is composed by: one or more CPUs with their private levels of cache; one or more banks of shared cache; a router that interconnects the tile with the rest of the memory system. Some tiles also have memory controllers.

Figure 1.3 shows an example of a 16-tile CMP with 2 levels of cache. The first level is private while the second is shared with its banks distributed among the tiles. There are 4 memory controllers in the corner cores of the layout.



(a) Layout.

(b) Tile.

Figure 1.3: 16-core DSM multi-processor with tile-based organization.

1.4 Networks-on-Chip (NoCs)

A network on-chip (NoC) is a switched communication medium on an integrated circuit. NoCs can be found in a wide range of systems and are not limited to many-core CMPs. One of the most common uses is to interconnect modules of a System on-Chip (SoC).

The first part of this section describes the most fundamental NoC design parameters (Section 1.4.1). The second part summarizes the most common deadlocks in NoCs (Section 1.4.2). The final part analyzes the role of NoCs in DSM multi-processors (Section 1.4.3).

1.4.1) Network design parameters

Networks have five main design parameters: the topology, the routing, the flow control, the router architecture and link architecture. These are described next.

1.4.1.A) Topology

The topology defines the number of routers and channels and how they are interconnected to offer the desired connectivity among nodes or terminals. The topology sets the theoretical limits of latency and throughput of the network. The main characteristics of a topology are its diameter, bisection width, degree and symmetry. There are other important properties such as the average distance or the path diversity. Table 1.1 contains the definitions of the most relevant topology properties.

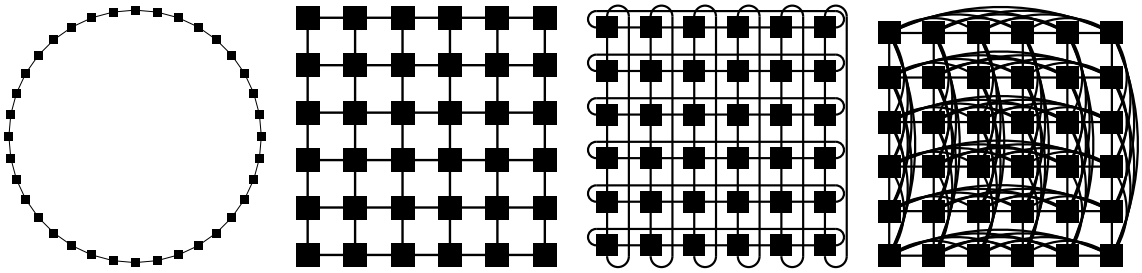
The most common NoC topologies used in CMPs are rings [Chrysos2014; Yoshida2018] and 2D-Meshes [Bell2008; Vangal2008; Sodani2016; ARM2018; Pellegrini2020]. The 2D-Torus [Dally1986], 2D-Flattened Butterfly [Kim2007] (2D-FBFLY) and 3D-Mesh [Rahmani2010] have been proposed too. Figure 1.4 shows a visual representation of some of these topologies with 36 routers.

All of them are direct topologies, i.e., each terminal node is associated to a router. Rings[1] and tori are also known as k-ary n-cubes, where k is the number of routers along

[1]Rings are k-ary 1-cubes, i.e., unidimensional tori.

Table 1.1: Summary of the most relevant topology properties.

Property	Definition
Degree (δ)	number of links connected to each router
Diameter (H_{max})	maximum distance (hops) between any pair of routers
Average distance ($\overline{H}$)	average distance between pairs of routers
Bisection Width (BW)	number of links across the smallest section that divides the network in two equal parts
Bisection Bandwidth (BB)	minimum bandwidth across the links of the BW
Node-symmetry	a node-symmetric network has no distinguished routers, i.e., the inter-connectivity of a router with the rest of the routers is the same for every router
Path diversity	minimal number of disjoint shortest paths between any two routers



(a) Ring ***(b)*** 2D-Mesh ***(c)*** 2D-Torus ***(d)*** 2D-FBFLY

Figure 1.4: Common NoC topologies with 36 routers.

each dimension and n the number of dimensions. Similarly, meshes are known as k-ary n-meshes. Presumably the future trend will be using topologies with more bisection bandwidth, as the number of cores in processors grows.

Table 1.2 summarizes the topology metric equations for the previous topologies and Figure 1.5 shows their evolution with the number of interconnected nodes. The evolution of each metric is described next.

The **degree** (or router radix) is one of the main factors that determines the footprint area and power consumption of the network. A larger degree implies larger routers, with more buffers and more complex allocators and crossbars. The more complex the router, the longer the critical path and therefore the lower the maximum operation frequency. Figure 1.5a shows that the degree is constant in the ring, 2D-mesh and 2D-torus (it only depends on the number of dimensions, n), but it grows following a square root law with the number of nodes in the FBFLY.

The **diameter** and **average distance** are indicators of the latency of a given topology at low traffic loads. The lower the distances, the shorter the paths and therefore the latency. Figures 1.5b and 1.5c show that both metrics grow linearly with the number of nodes in the ring, following a square root law in the 2D-mesh and 2D-torus, and they are practically constant in the 2D-FBFLY.

The **bisection width** and the **bisection bandwidth** are indicators of the worst-case performance of the network, as they set the maximum number of packets traveling from one half of the network to the other at a given time. Therefore, in terms of performance

Table 1.2: Properties of most common NoC topologies for N nodes. The equations of 2D topologies are for square networks, i.e., k is the same in both dimensions.

Property	Ring	2D-Mesh	2D-Torus	2D-FBFLY
Degree (δ)[23]	2	4	4	$2(\sqrt{N}-1)$
Diameter (H_{max})	$N/2$	$2(\sqrt{N}-1)$	$\sqrt{N}$	2
Average distance ($\overline{H}$)	$N/4$	$\approx 2\sqrt{N}/3$	$\approx \sqrt{N}/2$	$2-2/\sqrt{N}$
Bisection Width (BW)	4	$2\sqrt{N}$	$4\sqrt{N}$	$N^{3/2}/2$
Node-symmetry	Yes	No	Yes	Yes

the higher the better. However, in terms of costs they are indicators of the number of physical links deployed. Figure 1.5d shows that the bisection width is constant in the ring, grows following a square root law with the number of nodes in the 2D-mesh and 2D-torus, and a 3/2-power law in the 2D-FBFLY.

Rings, tori and FBFLYs have **node-symmetry**, while meshes do not. Node-symmetry is a property commonly forgotten but very important for two reasons.

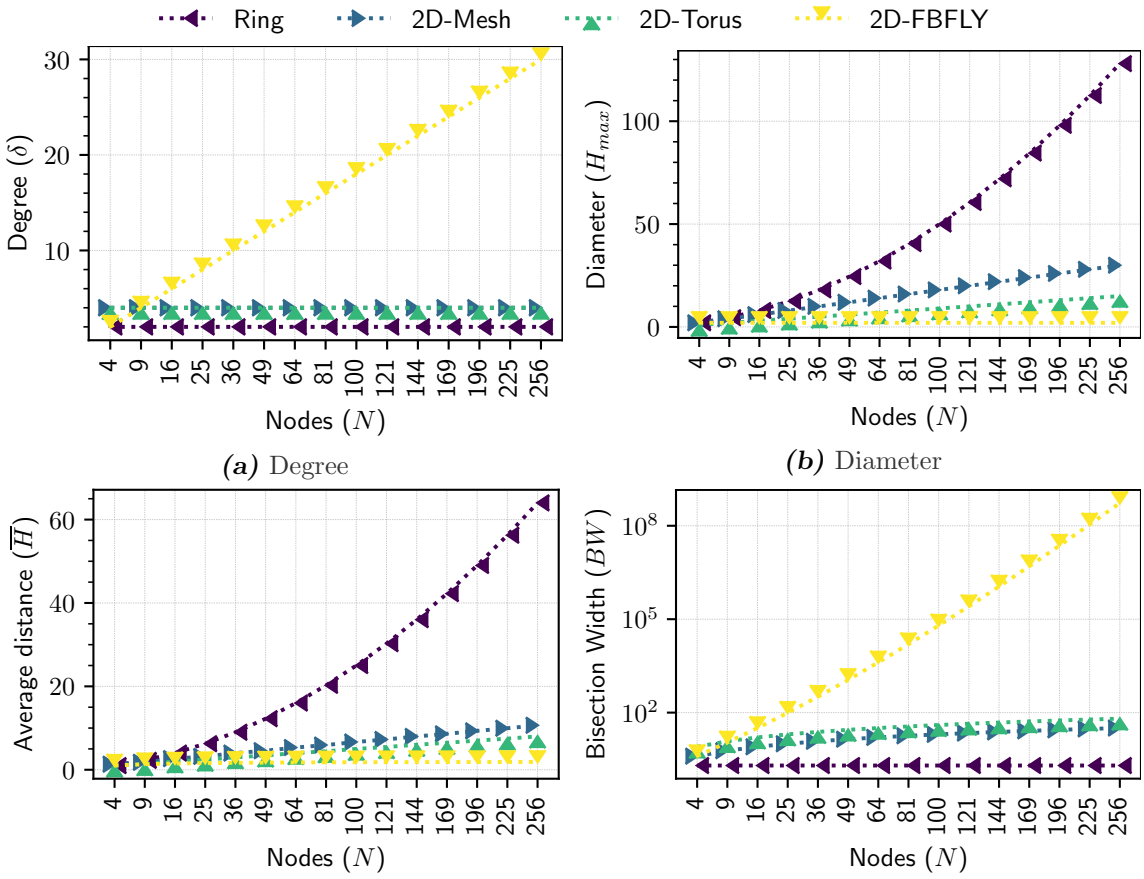
Firstly, it improves the balance in the utilization of the buffers and routers, distributing the load through the network, which has positive benefits in performance and heat distribution. This is especially true under uniform traffic, which is the purpose of using set-interleaved shared-cache in DSM processors since it seeks to distribute the traffic uniformly among its tiles. The mesh is an example of the opposite: its lack of symmetry increases the utilization of the links and routers in the center of the chip.

Secondly, it homogenizes the average distance of each node with respect to the rest of the network. In other words, the average distance of a node does not depend on the location of the node. In non node-symmetric topologies like meshes, the average latency of a node depends on its position in the network, causing load imbalance. Which, in turn, causes heterogeneity in the Average Memory Access Time (AMAT), leading to performance loss if not addressed properly. Edge-symmetry is another network property but is out of the scope of this work.

Figure 1.6 shows an example of load imbalance. The scenario represents the execution of a hypothetical embarrassing parallel program with a workload perfectly divided among the cores and a barrier at the end of the parallel region. It also supposes that all the NoC traffic is between the private caches and the shared-cache, which follows a set-interleaved bank distribution to produce uniform traffic. The workload is executed in a CMP with a 4-ary 2-mesh (4×4 mesh) and a 4-ary 2-cube (4×4 torus). Figure 1.6a shows the heterogeneity of the average distance ($\overline{H}$) depending on the location of the node in a mesh. This heterogeneity causes the performance imbalance shown in the execution trace depicted in Figure 1.6c. In this case, the execution time in each core is proportional to its $\overline{H}$. The execution time is determined by the cores with the highest $\overline{H}$, which are the ones located in the corners of the mesh. Besides, the sequential region after the barrier is executed in CPU_0 (common master thread ID), which in this case is located in one of the corners. In the 2D-torus all the nodes see the same $\overline{H}$, 2 in this case (Figure 1.6b), improving the overall performance while the CPU binding of the master thread is not relevant (Figure 1.6d).

[3]Without injection/consumption ports.

[3]The 2D-Mesh is irregular, the routers in the periphery have 3 ports and the ones in the corners have 2 ports.



(a) Degree

(b) Diameter

(c) Average distance

(d) Bisection width

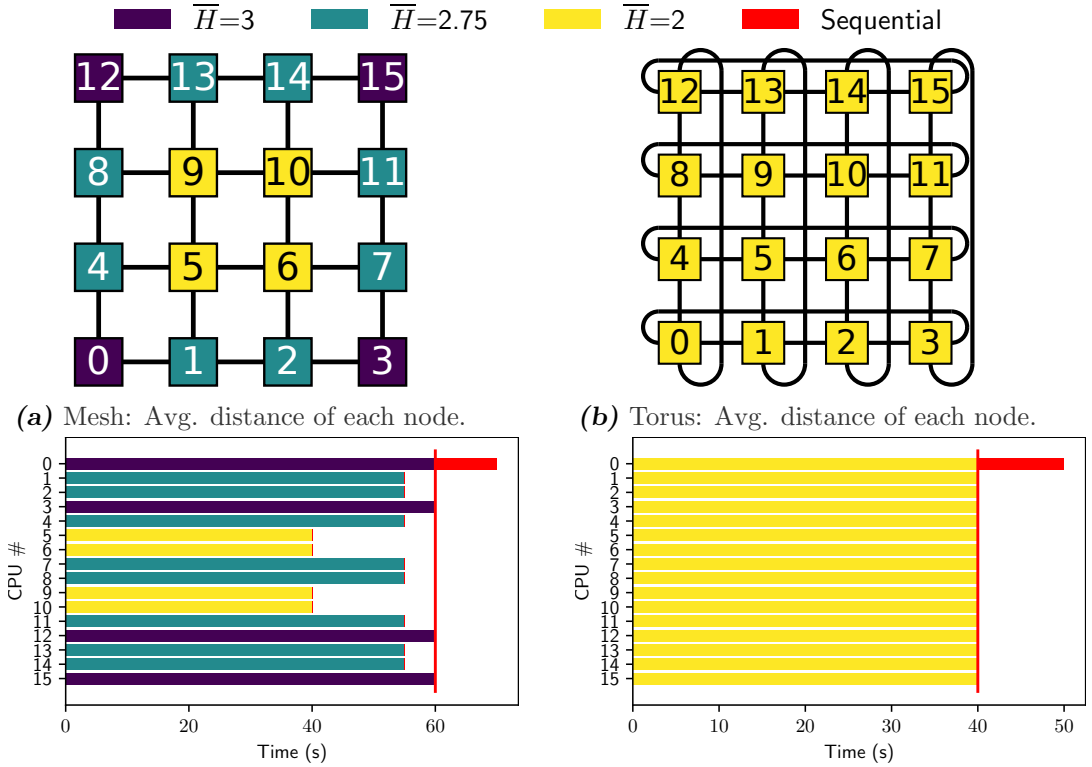
Figure 1.5: Topology metrics of rings and square 2D- meshes, tori and FBFLYs for different interconnected nodes (N).

In summary, rings do not scale, meshes and tori scale moderately, and FBFLYs scale well but are unfeasible for a large number of nodes. Between the mesh and the torus, the torus presents better topological metrics without incurring in a higher degree. However, as it is discussed in Section 1.4.1.C, their wraparound links make necessary additional mechanisms to avoid deadlocks.

1.4.1.B) Routing

The routing of the network determines the path between two nodes. The routing depends on the topology, which defines the available paths between the nodes. It has effect over the path length and the traffic balance among links and routers. There are two types of routing: oblivious and adaptive routing. Oblivious routing allows selecting between multiple routes independently of the network state. Deterministic routing is a particular case of oblivious routing that only defines a single route for each pair of nodes. Adaptive routing decides the route between two nodes depending on the traffic in the network. The route decision can be done in origin or in transit.

NoCs commonly use shortest-path deterministic routing. Shortest-path or minimal routing only selects paths that require the smallest number of hops to reach the destination, which is essential to minimize the average distance. Deterministic routing offers simplicity, which minimizes the computation cost required to determine the next hop.



(a) Mesh: Avg. distance of each node.

(b) Torus: Avg. distance of each node.

(c) Mesh: execution trace.

(d) Torus: execution trace.

Figure 1.6: Average distance of each node in a 4×4 mesh and torus, supposing uniform traffic. And, execution trace of an embarrassingly parallel program with a perfect workload distribution among the threads. The program has two phases separated by a barrier: the first contains the parallel region; the second part is a sequential region executed by the master thread (CPU 0 in this case).

1.4.1.C) Flow control

Flow control coordinates when data can move forward to the next router. It defines the packet transmission protocol between routers based on the status of the network resources. The typical network resources are the buffers and the crossbar of the routers, which are described in Section 1.4.1.D. Ideally, the flow control mechanism must assign the resources efficiently, without incurring in computation overheads. It may also include additional restrictions to guarantee deadlock freedom in some topologies like the torus as it is described in Section 1.4.2.

Flow control can allocate the resources for different size units. Messages are fragmented into packets, and packets may be divided into flits (flow control units). In addition, if the link width has fewer bits than the flit size, these can be fragmented into phits (physical units). The cache coherence protocols of CMPs have two types of messages: control and data. Control messages are data requests, invalidations, acknowledgments and other commands, and are small packets of typically 1 flit. Data messages are the longest messages and usually have only one packet of multiple flits that contain a cache line. For that reason NoCs message-based flow controls [Jerger2017] like circuit-switching are not used in general. The most common packet-based flow controls are Store-and-Forward and Virtual Cut-Through (VCT). Both require space for the whole packet in the buffer of the next router. In Store-and-Forward, the head flit of a packet has to wait until

the tail flit arrives to the router before advancing to the next. In contrast, with VCT the head does not have to wait for the tail flit. As for flit-based flow controls, WormHole (WH) and Virtual Channel (VC) are the most common. Both allow flits to advance when there is space for a flit in the next router. The difference is that in WH the links are assigned for whole packets. Whereas VC adds extra buffering resources per router input. So while the flits of packet are waiting for free space in the next router, the link can be used for flits of another packet. Thus interleaving flits of different packets assigned to different VCs (buffers).

VCs can reduce Head-of-Line Blocking (HoLB) and are also used to implement other mechanisms like adaptive routings and deadlock free techniques. For this reason, when we use the term VC, we refer to the buffering resources of the router. We also use terms VCT and WH to refer to VC flow control with packet-based restrictions (space for the whole packet) or flit-based restrictions (space for a flit), respectively.

The most frequently used flow control mechanisms in NoCs are VCT and WH. VCT is simpler than WH as it only computes allocation for head flits. However, the buffer has to be large enough to host the longest packet class in the network. Under low available space, WH is superior in terms of performance. This is because in WH, flits of different packets can be interleaved in a channel when there is not space to continue forwarding a packet. With enough space for packets both techniques are similar.

1.4.1.D) Router architecture

The router is the main component of a NoC, because links are common lanes in the metal layers of the chip used to interconnect the different modules of the circuit. Figure 1.7 depicts the 5-stage router micro-architecture presented in [Dally2003].

The router micro-architecture has 6 main elements:

1. **Input unit** is composed by Virtual Channels (VC) and a set of registers per VC (*G, R, O, P, C*) that monitor their state. Global state (G) tracks the state of the VC, which can be *inactive*, *routing*, *waiting for a VC*, *active* or *waiting for credits*. Route (R) holds the next output port granted in Routing Computation. Output VC (O) holds the output VC granted in VC Allocation. The head and tail flit pointers of the current packet are recorded in (P). Credits available (C) has the remaining empty slots in the output VC assigned.

2. **Route Computation unit (RC)** computes the direction (output port) of a packet's next hop. In multi-dimensional topologies like the mesh or the torus, the head flit of a packet typically carries the remaining hops per dimension. For example in DOR, RC decrements the remaining hops in the traveling dimension until it is 0, in which case it changes the traveling dimension.

3. **VC allocator (VA)** allocates an available VC in the next router to an input VC. This allocator has the same number of inputs and outputs: $\#VCs \times \#Ports$.

4. **Switch allocator** allocates a path in the switch to a flit. This allocator has the same number of inputs and outputs: $\#Ports$.

5. **Switch (or crossbar)** conforms paths between the different combinations of input and output ports.

6. **Output unit** contains a latch to store flits after traversing the switch and registers to control (*G, I, C*) the state of the next router VCs. Global (G) tracks the state

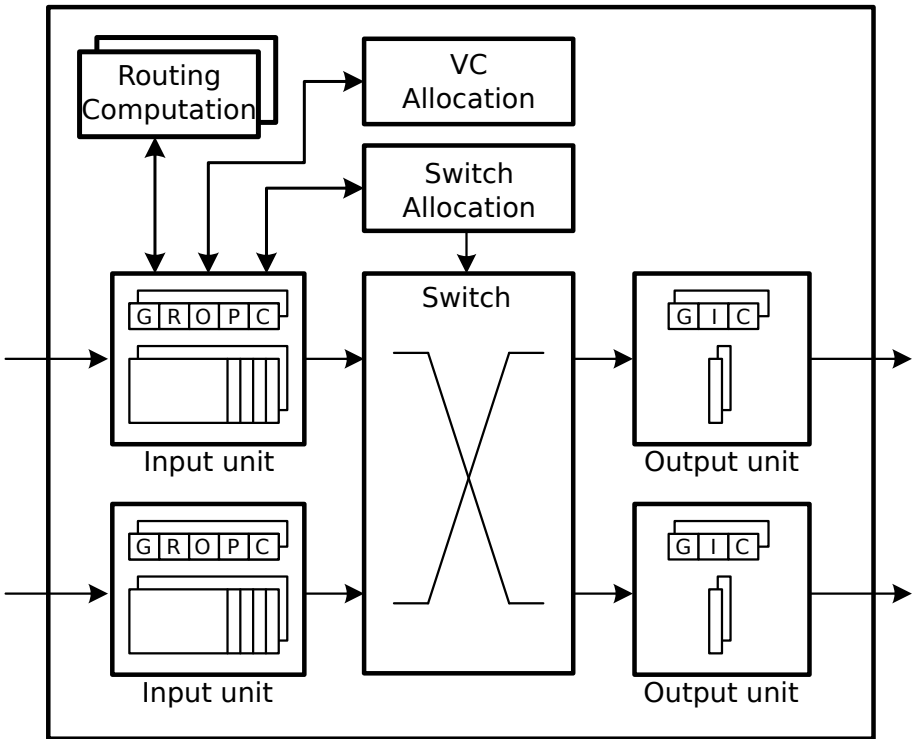


Figure 1.7: Router micro-architecture following [Dally2003]. Input unit control registers: Global state (G) can be inactive, routing, waiting for a VC, active, waiting for credits; Route (R) holds the output port after RC; Output VC (O) holds the output VC after VA; Pointers (P) to the head and tail flit of the current packet; Credits available (C) for the output VC assigned. Output unit control registers: Global state (G) can be inactive, active or waiting for credits; Input VC (I) holds the input VC, including the input port, that is forwarding a packet; Credit count (C) stores the number of free flit slots of this VC in the next router.

of the output VC which can be inactive, active or waiting for credits. Input VC (I) holds the input VC (including the input port) assigned after a match in VA. Credit count (C) stores the number of free slots of the VC in the next router.

From these elements, the most relevant in the context of this thesis are the VC buffers and the allocators.

Buffers hold packets or flits when a packet cannot advance to its next hop. This occurs when there are conflicts with other packets, i.e., various packets request the same resources or there is no available space in the buffers of the next hop. There are three common buffer organizations: single private buffers, multiple private VC buffers and shared buffers between multiple VCs. In single private buffers, routers only have one buffer per input port. In multiple private VC buffers (represented in Figure 1.7), routers have one buffer per VC in each input port. In shared buffers between multiple VCs [Tamir1992], routers have one buffer per input port, which is shared among all the VCs.

As mentioned before, VCs have multiple uses like, avoiding deadlocks, implementing Quality of Service (QoS), reducing HoLB, etc. However, buffers are one of the most area and power demanding parts of the router. For example, the buffers of the TRIPS NoC prototype [Gratz2006] require 75% of all the area used by routers. Increasing the number

of VCs does not only increase the number of buffers, but also complicates the logic of their allocation and management. For these reasons, there are alternative designs that focus on reducing or eliminating buffers like in bufferless NoCs [Requena2008; Moscibroda2009] or centralized buffer routers [Hassan2013; Hassan2014]. Bufferless NoCs deflect packets when there are conflicts (deflection routing), so packets are always traveling through the network and adapting their route. Centralized buffer routers, replace the VC buffers at the input ports with a centralized buffer shared by all the input ports.

Allocators are in charge of assigning the resources of the router. In a traditional router like the one described in this section the resources are the output ports of the switch (SA) and the VCs of the next router (VA). Allocators combine multiple arbiters. Each arbiter assigns one resource to one of several possible requests. An allocator has as many arbiters as required to match all the resources. For example, the SA of a typical 5-port router of a 2D-mesh has 5 arbiters and each one assigns 1 output port to one of the 5 possible requests. There many arbitration policies [Dally2003] like round-robin or matrix arbitration, and allocators types like separable or wavefront allocators.

5-stage router pipeline. The router micro-architecture presented in [Dally2003] has a pipeline with 5 stages:

1. **Routing Computation (RC)**. Flits arrive to the router and are written in the VC buffer assigned in the previous router (Buffer Write, BW). In parallel, routing computation obtains the next hop output port for head flits at the front of the VC buffers. RC is only done by head flit packets in multi-flit packets.

2. **Virtual channel Allocation (VA)**. Packets request an idle VC in the next router with space for a flit when using flit-based flow controls like WH or for the packet size when using packet-based flow controls like VCT. This stage is only done by head flit packets in multi-flit packets.

3. **Switch Allocation (SA)**. Flits request the setup of the switch to interconnect the packet's input port with the output port obtained in RC.

4. **Switch Traversal (ST)**. Flits traverse to the output unit through the path conformed in SA.

5. **Link Traversal (LT)**. Flit traverse the link towards the next router.

Figure 1.8 depicts the pipeline of a router retransmitting 4 flits. The diagram shows the optimal operation considering a specific input unit of the router. With ideal traffic (without conflicts between packets) and allocators, all the input units can follow the same pipeline (without stalls) at the same time. The pipeline length determines the router delay, which is 5 cycles in this example.

1.4.1.E) Link architecture

Links are the wires that interconnect two routers. NoCs commonly use conventional full-swing logic in which signal levels go from 0 V to the supply voltage to transmit a 0 and 1, respectively. There is at least one NoC prototype [Krishna2010] that uses low-swing signaling to reduce the power consumption of the crossbar and links in bypass routers.

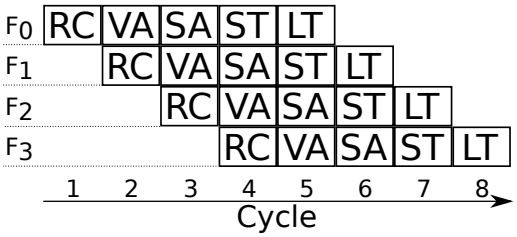


Figure 1.8: 5-stage router pipeline. The diagram shows four flits (F_0-F_3) that share the input and output of a router, crossing the router and a link.

Another common technique used to reduce the cost of long links uses asynchronous repeaters. The delay of a wire grows quadratically with its length [Rabaey2003], preventing the interconnection of distant nodes. Asynchronous repeaters divide long wires in multiple smaller ones, as shown into Figure 1.9, avoiding the quadratic increase.

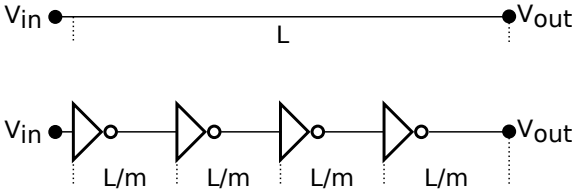


Figure 1.9: At the top, normal wire of length L without repeaters. At the bottom, wire segmented in m parts of length L/m with asynchronous repeaters.

1.4.2) Deadlock avoidance

Designing a deadlock free NoC is essential to guarantee the correct operation of the system. Deadlocks occur when there are dependency cycles between the paths of different packets. The typical deadlock [Dally2003] occurs when packets have freedom to choose the order in which they travel through the dimensions of a multi-dimensional network such the ones considered in this thesis. Defining a Dimension Order Routing (DOR) algorithm in which packets can travel through the networks is enough to avoid this type of deadlock. This is a shortest-path deterministic routing algorithm that guarantees deadlock freedom among dimensions [Duato2003; Dally2003] and is the one used through this thesis. DOR is enough to make meshes and flattened butterflies free from deadlock.

Designing deadlock free NoCs that implement torus topologies is more complicated. Their wraparound links may generate cyclic dependencies between packets when traveling through the same dimension. To avoid this kind of deadlock it is necessary to implement additional restrictions in the flow control mechanism. The most common types of restrictions are Dateline and Bubble flow controls.

Dateline [Dally1987] breaks cyclic dependencies by dividing dimensions in two parts and enforcing a specific order in the utilization of the VCs. The dateline is the dividing line between the two parts of a dimension, for example crossing the wraparound links. The set of VCs is also divided in two subsets, the first one is assigned only to packets that start to travel through the dimension until they cross the dateline. After, only VCs of the second subset can be used.

Bubble Flow Control [Carrion1997] prevents deadlock by avoiding filling up all the buffers of the dimension by leaving empty spaces called *bubbles*. The main restriction

consists in requiring space in the buffer of the next router for the packet being forwarded, plus a bubble when there is a change of dimension (the injection of a packet in the network is also considered a change of dimension). Thus, packets already in a dimension can advance occupying the bubble reserved and freeing space in their previous buffer like in a sliding puzzle. There are multiple implementations of Bubble Flow Control [Carrion1997; Chen2011; LizhongChen2013; Wang2013; Hassan2014; Ma2015; Parasar2019] with different types, sizes and number of bubbles. The number of bubbles can be minimized by globally checking that there is at least one bubble in the dimension like in Critical Bubble Scheme [Chen2011].

The original Bubble Flow Control is based on VCT. It checks for local bubbles whose size is that of the longest packet class. For NoCs, one of the most suitable implementations is Flit Bubble Flow Control (FBFC) [Ma2015] which is based on WH and the bubble size is one flit. The authors propose two versions of FBFC, one with local bubbles and other with a global bubble per dimension.

Dateline requires multiple VCs but underutilizes part of them. Bubble flow control sets a minimum buffer size and underutilizes some buffer space. However, requiring more space for individual buffers is usually simpler in terms of logic than requiring multiple VCs.

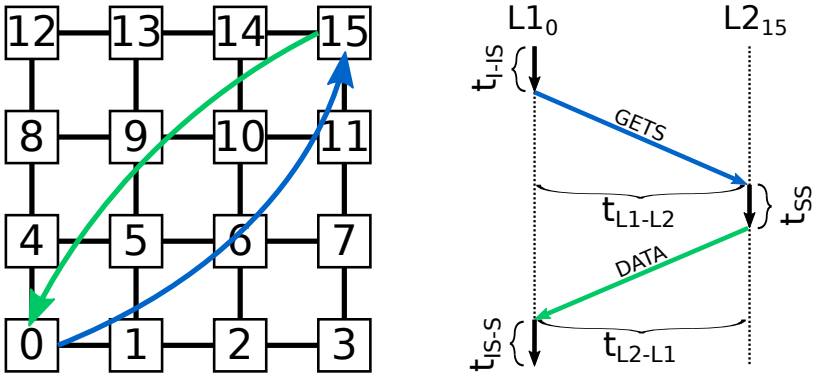
Another type of deadlock may occur when the network traffic has request-replay messages, like in the case of CMPs with cache-coherence protocol. This type of deadlock is caused by dependencies between both classes of messages attempting to acquire the ejection queues of the nodes, which may not be able to consume packets because it is waiting for a replay message. This is typically solved by segregating the different classes of traffic in several sub-networks, either physical or virtual.

1.4.3) NoCs in CMPs

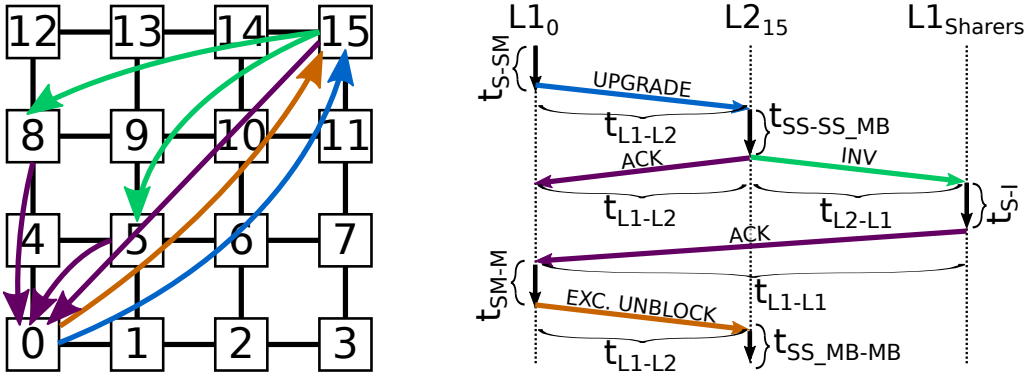
As shown in previous sections, NoCs play a key role in the memory sub-system of DSM multi-processors. NoCs are in charge of transporting messages between the different levels of the memory hierarchy. These messages are generated when there are cache misses, which trigger the communication defined in the cache coherence protocol.

Figure 1.10 shows two examples of the messages produced by a cache coherence protocol. The protocol presented in this case is a directory-based MESI with two levels of cache. Both examples have the same memory configuration: private L1-Data caches and a shared L2 distributed among 16 tiles. The first example (Figure 1.10a) shows the execution of a load instruction in CPU_0 that misses in L1. The miss in L1 produces a read data request message (GETS) to the L2 directory in tile 15 that responds with a data message (DATA). The second example (Figure 1.10b) shows an example of the execution of a store from CPU_0 to a shared memory address used by two other CPUs, of tiles 5 and 8. $L1_0$ sends a control message (UPGRADE) to the L2 directory in tile 15 to inform to the rest of the sharers that the data has been modified. The L2 sends invalidations (INV) to $L1_5$ and $L1_8$, who send acknowledgment (ACK) messages to $L1_0$ to inform that they have invalidated the corresponding cache lines. Finally, $L1_0$ sends a last message (EXCLUSIVE_UNBLOCK) to the directory in $L2_{15}$ to inform that the rest of the copies have been invalidated so it can exit the blocking state (SS_MB).

Both examples illustrate the fundamental role of the NoC in memory operations, firstly because the latency of the NoC is part of the total memory access time, and secondly because the cache coherence protocol produces multiple messages per operation. Thus, the NoC should have the minimum latency without incurring in prohibitive costs.



(a) Load to address in shared state.



(b) Store to address in shared state.

Figure 1.10: Sequence of messages generated after a load and store misses when the data requested is in shared state. The meanings of the commands are the following: GETS is a read data request; DATA is a data response; UPGRADE indicates that a shared data has been modified; INV are messages that invalidate the copies of data sharers; ACK indicates that a requested action has been completed; EXC. UNBLOCK indicates to the directory that a transaction has been completed. The meaning of the main cache states represented in $t_{State-State}$ are: I, Invalid; S, Shared; M, Modified; B, Blocked; The rest are transient states between the previous ones.

To provide low latency the NoC has to operate outside the saturation region, where the throughput reaches its peak but the latency skyrockets. Figure 1.11 shows the latency curve versus the offered load of a network. The region in green shows the acceptable operation range of a CMP NoC while the undesirable saturation region is painted in red. The saturation point is typically located where the latency is 3 times the zero-load latency (T_0).

One of the key factors that determine the traffic load is the miss ratio of the caches, which usually is very low. Therefore, focus on minimizing the zero-load latency, which defines the base latency of the NoC when there is no traffic, is critical when designing a NoC for a CMP. Equation 1.1 evaluates the zero-load latency (T_0) when using minimal routing.

$$T_0 = H_{min} \times T_r + T_s \tag{1.1}$$

T_0 is function of the number of hops following minimal routing (H_{min}), the router delay (T_r) and the packet serialization time (T_s). The number of hops is given by the topology

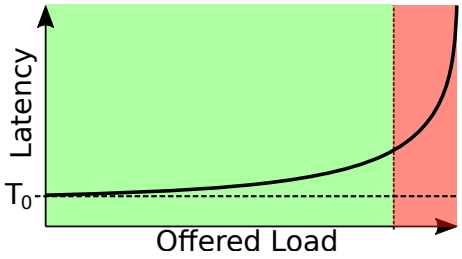


Figure 1.11: Average packet latency vs offered load curve of a network. The green region represents the operation range of a CMP NoC. The red region represents the saturation region.

of the network. The router delay is given by the pipeline length of the architecture. The serialization time depends on the packet size in flits, which is given by the link width.

The main goal of bypass routers is the reduction of the zero-load latency by improving the router delay (T_r, single-hop bypass) or the number of hops (H_{min}, multi-hop bypass). Improving these mechanisms is the main target of the thesis and are described in Chapter 2.

1.5 Motivation

Traditional NoC routers have long pipelines, which make latency quickly increase with the network size. High degree topologies like the FBFLY [Kim2007] or the Slim NoC [Besta2018] have low average distances at the expense of using high degree routers, which increase the complexity of allocators, the crossbar size and the buffer area. A way of reducing the buffer area is to use centralized buffers [Hassan2013], however this does not reduce the allocation complexity and crossbar size in such topologies. The more complex the router, the more area, power consumption and delay of some pipeline stages. The area and power of the NoC cannot be underestimated as it may represent a large portion of total CMP area and power budget [Li2009].

The objective of this thesis is the design of efficient low latency NoCs for future many-core processors. This thesis introduces efficient router architectures based on bypass routers, which reduces the two main factors of the zero-load latency. We segregate bypass routers in singe- and multi-hop bypass. Single-hop bypass focuses on reducing the router delay (T_r) while multi-hop bypass focuses on reducing the effective number of hops (H_{min}). The proposed mechanisms improve the bypass and buffer utilization, attaining substantial savings in terms of VC count which reduces the complexity of routers. The final proposed mechanism, combines both types of bypass to reduce, even further, the zero-load latency. Additionally, we adapt some of these mechanisms to torus topologies as they provide more throughput, have symmetry and have similar costs than the mesh.

1.6 Organization

This thesis is structured as follows. Chapter 2 presents the required background on what are bypass routers. We describe the fundamentals of single-hop and multi-hop bypass

routers, including their micro-architecture and pipeline. Chapter 3 describes BST (Bypass Simulation Toolset), a simulation toolset developed to carry out the experimentation required in the pursuit of this thesis. Chapter 4 presents NEBB (Non-Empty Buffer Bypass), a proposal that relaxes the restrictions to forward packets and use the bypass in single-hop bypass routers. NEBB achieves better performance than the original architecture with smaller buffers and saving energy thanks to increasing the utilization of the bypass. Chapter 5 describes SMART++, a multi-hop bypass network with an efficient flow control mechanism that adapts NEBB to multi-hop bypass routers. SMART++ increments the utilization of the bypass and uses buffers efficiently, not requiring VCs to achieve the same performance as SMART, drastically reducing router complexity. Chapter 6 outlines S-SMART++, another multi-hop bypass network that reduces the router delay of SMART by means of speculative allocation. S-SMART++ efficiently reduces the latency giving a performance similar to the most costly and prohibitive implementations of SMART. Chapter 7 lists the most relevant related work. Finally, Chapter 8 summarizes the conclusions of the thesis.

Chapter 2

Background

This chapter presents the required background to understand what are NoCs with bypass routers, why they are useful to minimize latency and how to implement them.

Minimizing the NoC latency is critical for a CMP as it directly affects the AMAT of the processor. The router architecture and the topology are the most important factors that determine the latency of a NoC. The router architecture, particularly the router pipeline, defines the router delay. The topology establishes the distance between nodes and therefore the number of hops done by packets. Therefore, to reduce latency, one has to reduce the router pipeline length and the average distance between nodes.

Decreasing the pipeline stages by merging them incurs in lower operation frequencies, and simplifying the functionality of routers reduces throughput. Nonetheless, there have been multiple single-stage router architectures used and proposed [Hoskote2007; Olofsson2016; Bohnenstiehl2017; Kwon2017; Rovinski2019]. Decreasing the distance between nodes typically incurs in increasing the router degree, which increments and complicates logic. Examples of this kind of solution are the flattened butterfly [Kim2007], express cube topologies [Grot2009a] or ruche channels [Ou2020].

To reduce both factors we instead focus on bypass routers. The key idea of this type of routers is to try to allocate router resources before packets arrive to them. In this way, once packets arrive, they have already set a destination VC and configured the switch, so they can skip some pipeline stages by traveling through a bypass path, reducing latency. We distinguish between two different types of bypass: single-hop and multi-hop bypass. The first reduces the router delay without merging pipeline stages, while the second reduces the effective number of hops without increasing the degree of the routers. Single-hop bypass is described in Section 2.1 and multi-hop bypass in Section 2.2.

2.1 Single-hop bypass routers

Single-hop bypass routers, also known as LookAhead bypass routers, takes advantage of lookahead routing [Galles1997] to skip some stages of the router pipeline. Lookahead routing calculates the route of the next hop in the previous router (upstream router). The main idea of bypass routers is to forward the lookahead routing information before forwarding a packet. This information is sent in a signal called LookAhead (LA), or advance bundle in [Kumar2007], which requests a destination VC and an output port in the downstream router one cycle before the arrival of the packet. As occurs with

flits, LAs request the router resources (including the bypass paths) in the allocators to resolve possible conflicts. In case of success, the downstream router prepares a bypass path so the packet can advance directly to the switch, skipping the initial pipeline stages. Besides reducing the router delay, the mechanism also reduces power consumption by reducing the utilization of buffers. Buffers are one of the most consuming element of the NoC [Gratz2006; Kahng2009; Sun2012], thus reducing the number of writes and reads in them results in important dynamic energy savings.

This section starts describing the router micro-architecture (Section 2.1.1), continue with the pipeline (Section 2.1.2), and ends with some implementations considerations (Section 2.1.3).

2.1.1) Router micro-architecture

Single-hop bypass routers are built on the architecture of traditional routers described in Section 1.4.1.D. The architecture of single-hop bypass introduces changes in some units of the traditional router and adds two new units.

Figure 2.1 depicts a diagram of the router architecture with the changes highlighted in red. This architecture follows the fundamentals of the bypass NoCs proposed in [Kumar2007; Kumar2008; Krishna2010]. The changes made to support bypass are described next.

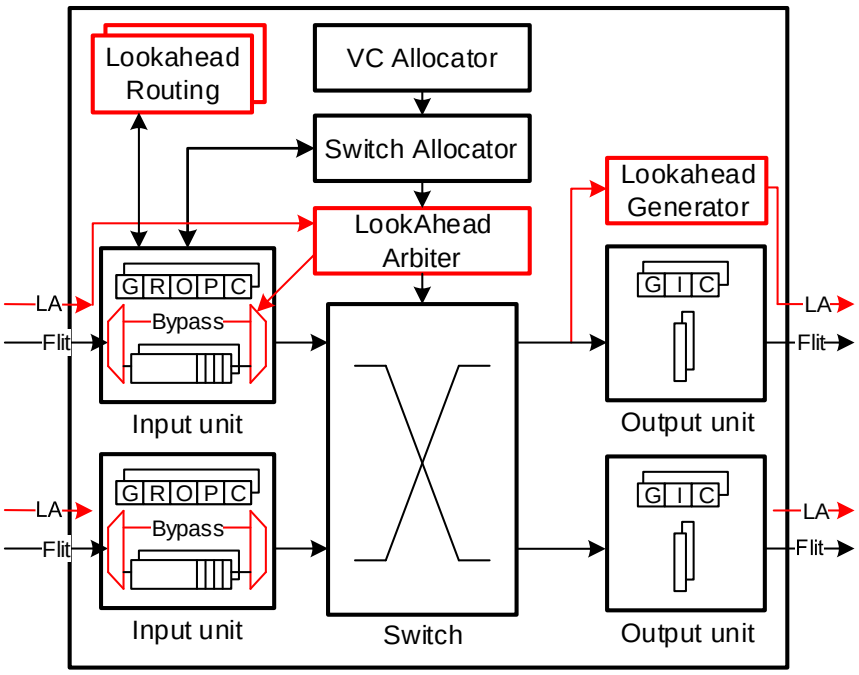


Figure 2.1: Single-hop bypass router architecture.

2.1.1.A) Changes in standard units

The changes in the units of traditional routers affect the input unit and the routing computation:

Input Unit. The input unit of single-hop bypass routers includes the bypass path formed by a pair of multiplexer and demultiplexer. These are controlled by the LookAhead Arbiter

(describer later). The bypass path connects the input port directly to the switch of the router. Every packet has an associated VC despite not using the corresponding buffer when taking the bypass. Consequently, the VC registers of the input unit still track the status of the resources for every packet.

LookAhead Route Computation (LA-RC). Single-hop bypass routers require LA routing to generate the LA signals that request the switch of the downstream router before the arrival of flits. LookAhead (LA) routing computes the route, i.e., the next output port, in the previous upstream router, instead of doing it when the packet is at the head of the input VC. Thus, when this information arrives at a router, it already carries the next output port to request in VA and SA. Implementing LA routing with deterministic minimal routing does not imply any complication. The only difference with respect traditional RC consists in including an RC unit in the Network Interface Controller (NIC) to obtain the route of the initial hop. Otherwise, the initial router has to compute two routes, the next hop route and the LA route for the LA signal. The logic of the RC units does not change when using DOR, and the packet head has the same fields to keep track of the remainder hops in each dimension. These units compute the next hop output port based on these fields and decrease the corresponding one to the traveling dimension.

2.1.1.B) New bypass units

The two additional units in single-hop bypass are the LookAhead Arbiter (LA-Arb) and the LookAhead Generator (LA-Gen):

LookAhead Arbiter (LA-Arb). Routers can receive multiple LAs from different input ports in the same cycle. LA-Arb is an allocator that matches the bypass and output ports of the router to the input ports of those LAs. In other words, it is equivalent to SA but for LAs instead of flits or packets. In addition, it is in charge of arbitrating among flits and LAs that competes for the same output port. Figure 2.2 depicts the organization of LA-Arb in a router with N inputs and M outputs. While SA assigns output ports to flits, LA Allocator does the same with LAs. Once both assignments are complete, the multiplexers at each output port resolve possible conflicts between LAs and flits following one of two policies: priority to local flits or to bypass requests. These policies are described later in Section 2.1.3.B and shown in form of pseudo-code in Figure 2.2.

LookAhead Generator (LA-Gen). LA-Gen produces LA signals to acquire the switch in the next router. The generation of LAs starts when the router grants an out-port to a flit. One LA-Gen unit is integrated in each of the output units. LAs are forwarded to the downstream router when the associated flits are traversing the switch.

2.1.2) Router pipeline

Single-hop bypass routers have two pipelines. These are shown in Figure 2.3. The first pipeline uses the standard path in which flits pass through 4 stages, which is used by R_0 in the figure. This pipeline is similar to the pipeline of the traditional router described in Section 1.4.1.D, but with one stage less due to LA routing. The second pipeline uses the bypass path in which flits only traverse the switch and link. This is used by R_1 in the figure. The pipelines are described next.

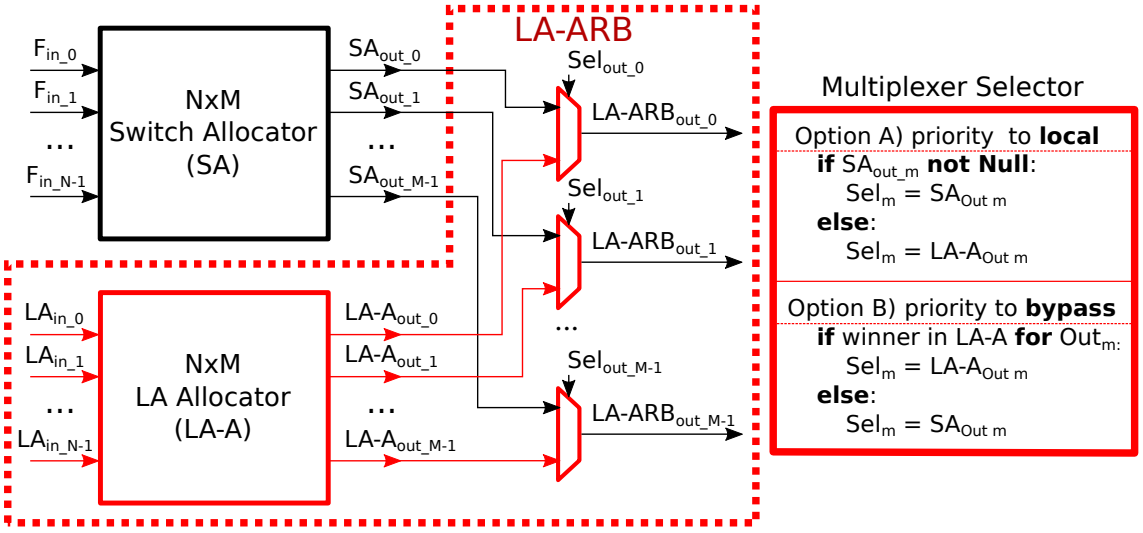


Figure 2.2: LookAhead Arbiter (LA-Arb) organization. New units and signals are represented in red. F_n are flit requests to SA from input n. LA_{in_n} are LA requests to LA-Arb from input n. SA_{out_m} are Switch Allocation outputs for output m. $LA-A_{out\_m}$ are LA-Allocation outputs for output m. $LA-ARB_{out\_m}$ are LA-Arbitration outputs for output m. Sel_{out_m} contains the control information to set up the switch and select the path at the input unit, from input n to output m.

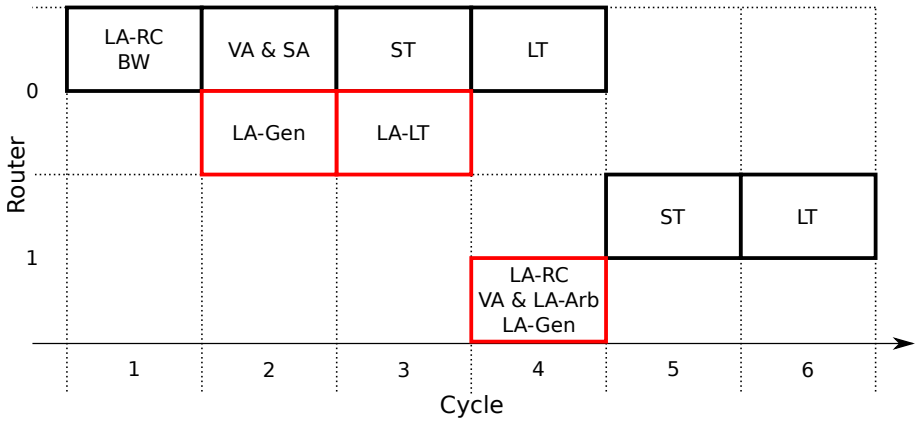


Figure 2.3: Pipeline of single-hop bypass routers. Example of a flit using the traditional pipeline in router 0 and the bypass pipeline in router 1. Stages executed by an LA are highlighted in red.

Stage 1) **LA-RC** and **BW**. When a flit arrives at the router, the input multiplexer selects between the standard or the bypass paths. If the standard path is selected, the router computes the next hop route of the packet (LA-RC) only when the flit is the head of a packet and writes the flit in the buffer (BW). If the bypass path is selected (like occurs in router 1 at cycle 5 in Figure 2.3), the flit takes the bypass path to the input of the switch to perform ST (Stage 3) in the current cycle.

Stage 2) **SA**, **VA** and **LA-Gen**. The router reads the flit at the front of the buffer. If the flit read is the head of a packet, the router attempts to allocate a destination VC in VA. In parallel the router also tries to allocate the switch to the flit in SA, independently of the type of flit. VA is executed at the same time but not

speculatively [Mullins2004], hence head flits have to win both allocations. When a flit successes, LA-Gen prepares an LA for the next cycle.

Stage 3) **ST** and **LA-LT**. In this stage flits traverse the switch of the router while the LAs traverse the links towards the next routers.

Stage 4) **LT**, **LA-RC**, **LA-Arb** and **LA-Gen**. The flit traverses the link while the associated LA requests the switch in LA-Arb in the next router. If the LA successes, the LA-Gen of the downstream router prepares another LA in the output unit to forward the signal in the following cycle. Additionally, if the LA is associated with a head flit, LA-RC for the next hop takes place in parallel.

2.1.2.A) Pipeline walk-trough

This section presents a walk-trough example of a packet traveling through the network to describe in detail how the router micro-architecture works. The example, depicted in Figure 2.4, illustrates how a packet (highlighted in green) traverse three routers, using the standard pipeline in the first and the bypass pipeline in the last two. The configuration of the example is as simple as possible: with single-flit packet, without VCs and with routers of only one input/output port. The example is described next, cycle by cycle.

Cycle 1) In the initial state of the network the packet is in the input latch of R_0 and the bypass path of that router is not selected. The packet has only one flit, therefore the flit is the packet's head, and consequently R_0 computes in advance the route for the downstream router (LA-RC). We assume that the next hop route has been calculated in the upstream router or NIC. Meanwhile, the flit is written at the front of the VC buffer (BW).

Cycle 2) The packet wins SA and VA in R_0, acquiring the switch and a destination VC after a successful LA-Arb, since there are not conflicts with LAs. After acquiring the switch, it travels to the second pipeline latch. R_0, after granting access to the switch, generates an LA (LA-Gen), which contains the routing information computed in the previous cycle.

Cycle 3) The packet in R_0 traverses the switch (ST) towards the latch at the output port, while the LA traverses the link (LA-LT).

Cycle 4) The flit traverses the link (LT) from R_0 to R_1. Meanwhile, the LA in R_1 requests the setup of the bypass in LA-Arb while LA-RC computes the LA routing information of the next hop. The LA wins LA-Arb, acquiring the switch and a destination VC. LA-Arb prepares the bypass path of the input unit of R_1 for the next cycle. LA-Gen generates a new LA for R_2 with the routing information obtained in LA-RC.

Cycle 5) The packet, in R_1, takes the bypass path and performs ST. The LA, generated in R_1 in the previous cycle, traverses the link (LA-LT) towards R_2.

Cycle 6) The packet traverses the link (LT) to R_2. The LA, in R_2, acquires the bypass and a destination VC in LA-Arb and VA.

Cycle 7) The packet traverses the switch (ST) of R_2 passing through the bypass path.

Cycle 8) The packet traverses the last link (LT) leaving R_2.

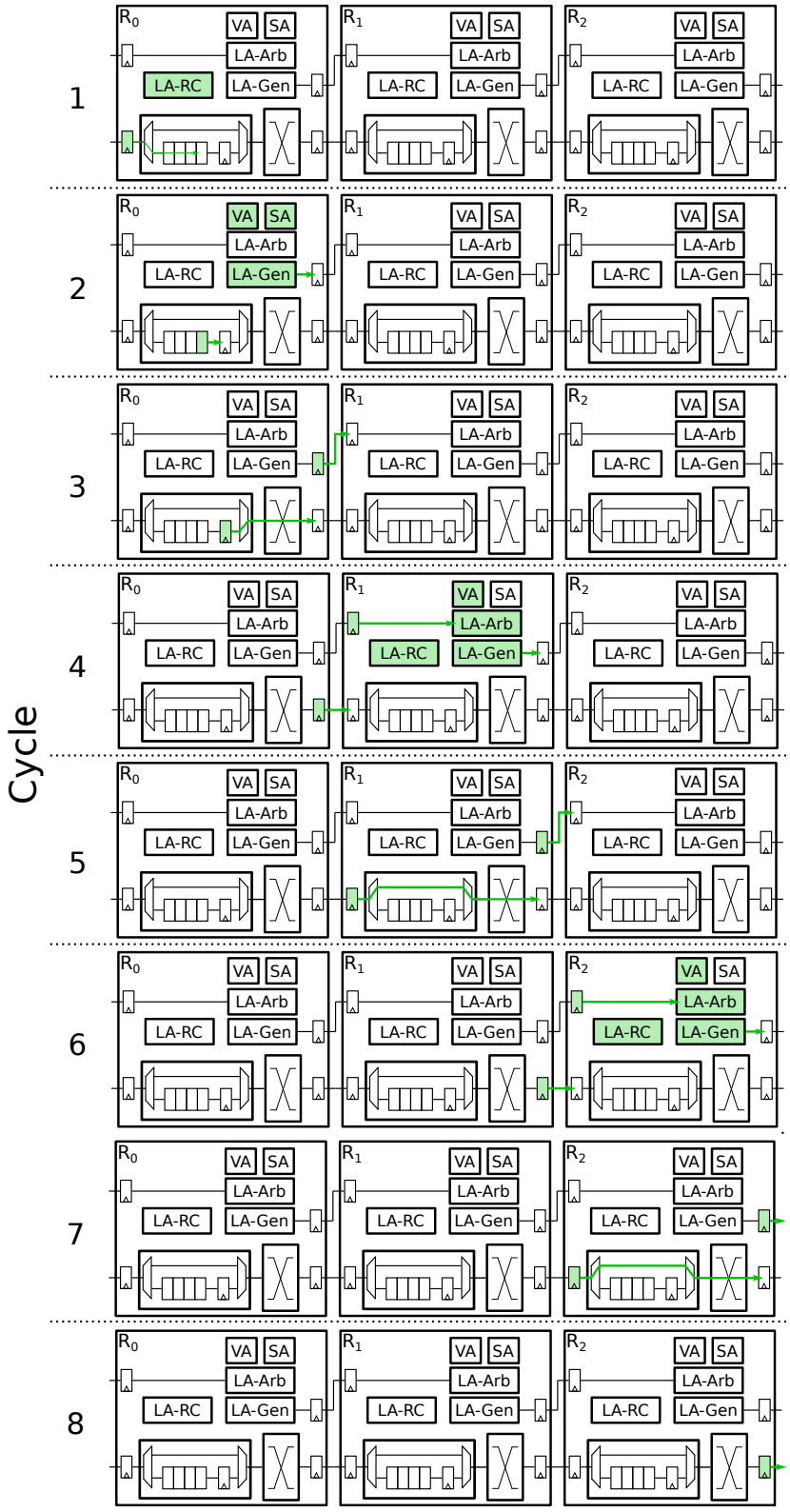


Figure 2.4: Single-hop bypass pipeline walk-through example.

2.1.3) Implementation details

This section reviews some micro-architectural details omitted in earlier sections to facilitate the understanding of single-hop bypass routers. Some of the following implementation details are proposed to reduce the timing constraints and hardware costs but are not required for a correct operation of the network.

2.1.3.A) Conditions to use the bypass

The utilization of the bypass is not straightforward. An incorrect flow control implementation may result in a network design prone to suffer from deadlocks. To avoid this problem, Kumar et al. [Kumar2007] only allow the use of the bypass when the following conditions are met:

1. *There is no flit already in the buffer at the input port where the advanced bundle arrives.*
2. *There is no output port conflict with existing flits, i.e., there is no flit already in the SA stage of the router which is waiting to use the same output port as the one requested by the advanced bundle.*
3. *There is no conflict with new flits, i.e., there is no output port conflict between multiple advanced bundle signals arriving in the same cycle, i.e., not more than one advanced bundle signal comes, simultaneously requesting the same output port.*

Conditions 2 and 3 are not required when using LA-Arb, which was introduced as LookAhead Conflict Check (LA CC) in [Kumar2008], as it is in charge of resolving possible conflicts caused by LAs. Chapter 4 studies why these conditions are required to propose efficient flow control mechanisms concerning resource utilization.

2.1.3.B) Arbitration policies

The inclusion of LAs brings on the necessity of establishing priorities between LAs and flits. The architecture proposed in [Kumar2007] ignores LAs when there are conflicts with flits or other LAs according to the conditions 2 and 3 mentioned in Section 2.1.3.A.

Introducing an LA-Arb makes possible the resolution of this kind of conflict. LA-Arb has two arbitration phases. The first phase arbitrates in conflicts between LAs that arrive from different input ports that request the same output port, like SA does with local flits. The second phase arbitrates in conflicts between data flits that win SA and LAs that win in the first phase.

The arbitration policy of the first phase is implemented similarly to SA, as both have the same number of inputs and outputs. In the second phase there are two possibilities, giving priority to local flits or giving priority to LAs. Giving priority to local flits avoids the possibility of one packet passing another one in the same flow, i.e., packets that share the source and destination. However, this policy does not make the most of the bypass opportunities. Giving priority to LAs increases the utilization of the bypass at the cost of modifying the order of the packets of a data stream. Nevertheless, this policy has the disadvantage of potentially causing starvation. The router architecture proposed in [Kumar2008] implements a starvation avoidance mechanism. It consists of keeping track of the number of consecutive flits bypassed for each output port. If there is a local flit waiting to use an output port more cycles than a specified threshold then the priority is inverted.

2.1.3.C) Virtual Channel Implementation

As mentioned in Section 2.1.2, VA is executed at the same time as SA, but not speculatively. A head flit has to win both allocations to advance to the next stage; body flits already have a VC assigned by their corresponding heads. This also affects LAs that win LA-Arb because they have to allocate a VC in the same cycle in which they complete the arbitration. To minimize the impact on the critical path of the stage, the router implementations of [Kumar2007; Kumar2008; Krishna2013; Kwon2017] use a simplified version of VA called VC Selection (VS). It relies on a pool of free destination VCs for each output port. The assignment of the VC only requires to take one VC from the pool. Flits have to retry SA in the next cycle when the pool is empty.

2.1.3.D) Switch Allocator

The SA of the single-hop bypass router depicted in Section 2.1.1 is similar to the SA of a traditional router. However, Krishna et al. proposed an alternative implementation in [Krishna2010]. Basically, SA is divided in two, Switch Allocation-Input (SA-I) and Switch Allocation-Output (SA-O). SA-I is executed in the first pipeline stage and selects a flit among the VCs of each input port to place a request in SA-O. This is implemented with a $\#VCs : 1$ arbiter for each input unit, where $\#VCs$ is the number of VCs. Then, SA-O is executed in the second pipeline stage and matches the output ports with the input ports, which are the winners of SA-I. SA-O is implemented with a $\#INs : \#OUTs$ allocator, where $\#INs$ and $\#OUTs$ are the number of inputs and outputs of the router, respectively.

2.1.3.E) LookAhead signaling

According to Kumar et al. [Kumar2007; Kumar2008], the information carried by LAs is the routing information. The number of bits depends on the route encoding and the network size. A very common encoding keeps track of the remaining hops in each dimension and the next output port. Two other types of information is required besides the route. First, LA-Arb has to distinguish between head and body flits, so an extra bit is required. Identifying the type of flit is essential to determine whether the flit has to acquire a destination VC (head) or it already has one assigned (body). Second, the LA has to carry the VC assigned after winning VA (and SA or LA-Arb) so the router can check the 3 conditions in Section 2.1.3.A. If the LA corresponds to a body flit, this VC is also used to read the routing information recorded in the control registers of the input unit.

Table 2.1 gathers the information carried by LAs in a 2D-Mesh without concentration and DOR encoding the remaining hops.

Table 2.1: LA signal bits for a k-ary 2-mesh without concentration and with DOR encoded with remaining hops per dimension and next output port.

Remaining Hops X	$log_2 k$
Remaining Hops Y	$log_2 k$
Next out-port	3
Flit type	1
VC assigned	$log_2 \#VCs$

2.1.3.F) Buffer management

In Sections 2.1.1 and 2.1.2 we have assumed the utilization of a private buffer for each VC. An alternative consists in using shared buffers among multiple VCs also known as Dynamically-Allocated Multi-Queue (DAMQ) buffers [Tamir1992]. This is the buffer organization followed in [Kumar2007; Kumar2008; Krishna2010]. Besides, these architectures use on/off signaling [Dally2003] instead of credits. This signaling mechanism consists in enabling a signal towards the upstream router when there is more available space than a given threshold to tolerate the round-trip delay. One buffer slot is reserved per VC to avoid deadlock.

The buffer configurations used in these router architectures have many VCs. The main reason to use DAMQs is the conservative restrictions of this bypass mechanism, which requires a multitude of VCs to exploit the NoC bandwidth. Section 4.1 discuses these requirements in detail.

2.2 Multi-hop bypass routers

NoCs built with multi-hop bypass routers were introduced in SMART [Krishna2013]. SMART, which stands for Single-cycle Multi-hop Asynchronous Repeated Traversal, allows packets to traverse multiple routers in a single cycle by using asynchronous repeaters within each router. The objective of SMART is to minimize the latency by reducing the effective average number of hops. By applying SMART to mesh topologies, the resulting NoC has a similar latency to high-radix NoCs like the FBFLY [Kim2007], without incurring in their area and power costs. Krishna et al. estimate in [Krishna2013] that an 8×8-mesh SMART reduces the latency of an equivalent size FBFLY with traditional routers while decreasing drastically the area (by 8.6×) and power (dynamic power at saturation by 1.5× and leakage by 10×).

Like single-hop bypass routers, SMART uses LA routing to set up multi-hop bypass paths. Instead of sending LAs with the routing information to adjacent routers, SMART broadcasts Switch Setup Requests (SSRs) to the routers within the route of packets. SSRs are broadcast up to a maximum distance of HPC_{Max} (maximum Hops Per Cycle). Figure 2.5 illustrates the fundamental idea of SMART. In *Cycle 1*, the green packet in R_0 broadcasts an SSR to the next three routers. The SSR acquires the switch of R_0-R_2 in Switch Allocation Global (SA-G), which is described later in Section 2.2.1, and the bypass in R_1-R_2. In R_3, the SSR of the green packet loses against the blue packet due to the arbitration policies explained in Section 2.2.3.B. Therefore, in *Cycle 2*, the green packet traverses the first three routers, the second and third though the bypass, and stops at R_3 where it lost against the blue packet.

Traversing multiple routers in a single cycle would not be possible without asynchronous repeaters. Moreover, according to Krishna et al. [Krishna2013] the use of repeaters also reduce area and power with respect to pipeline registers used in single-hop traversal NoCs. They also demonstrate that the maximum propagation distance is 16 mm when operating at 1 GHz and using a 45 nm technology process. Isolating the link traversal, the propagation distance is inversely proportional to the frequency, therefore, the maximum propagation distance is 8 mm at 2 GHz. In their experimentation they assume a distance of 1 mm based on the tile size of [Hoskote2007; Haring2012] and a frequency of 1 GHz. After aggregating the propagation delay of the multiplexers in the input

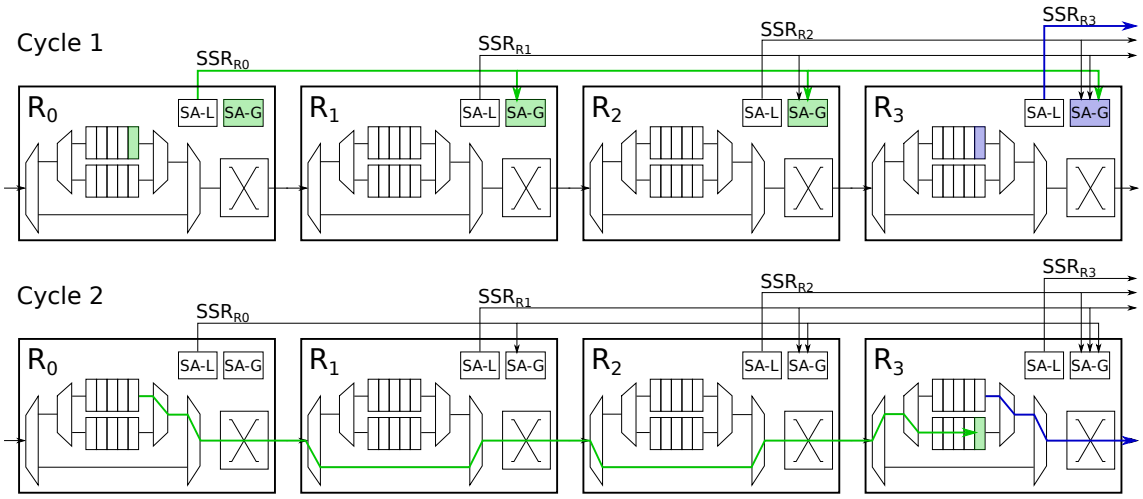


Figure 2.5: Overview of SMART's multi-hop bypass.

units and the crossbar, their theoretical HPC_{Max} is 11. The experimental evaluations of the contributions presented in Chapters 5 and 6 consider the same assumptions.

Next, Section 2.2.1 describes the router architecture of SMART. Section 2.2.2 shows the pipeline organization in detail with a walk-through example. Section 2.2.3 explains implementation details as well as alternative implementation ideas.

2.2.1) Router micro-architecture

The micro-architecture of multi-hop bypass routers is similar to the one of single-hop bypass routers. The fundamental differences affect LA communication (now refer as SSR) that expands to multiple routers, and LA-Arb (now refer as Switch Allocation-Global) that has to arbitrate among multiple requests per input port.

Figure 2.6 shows the router micro-architecture of SMART[1]. The architectural modifications required to implement multi-hop bypass affect three components: the computation of the route, the allocation of the switch, and the output unit organization.

2.2.1.A) LookAhead Routing Computation

SMART uses LA routing to compute in advance the output, not only of the adjacent router, but also in all the routers involved in the multi-hop. The number of hops to compute in advance is the minimum between HPC_{Max} and the remaining number of hops. In the case of implementing SMART_1D, the remaining number of hops refers only to the traveling dimension, and the output port is the same for all the routers involved. The particular characteristics of SMART_1D and SMART_2D are described in Section 2.2.3.C. SSRs carry the potential number of hops (H) of the next multi-hop. This information is filtered then in each receptor router by comparing the H value with the distance to the source router. If the value is greater or equal to such distance, the SSR passes to SA.

[1]To facilitate the comparison with other router micro-architectures presented, we maintain the VC allocator unit instead of VC Selection

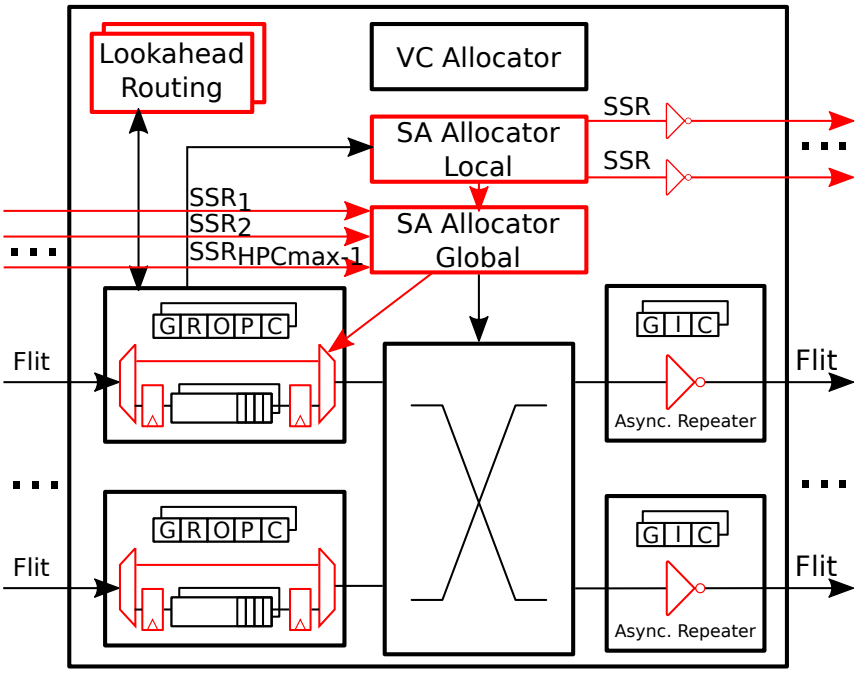


Figure 2.6: Multi-hop bypass router architecture.

2.2.1.B) Switch Allocation

SA is divided in two units: Local (SA-L) and Global (SA-G). SA-L is equivalent to SA in traditional and single-hop bypass routers. It decides which flits, i.e., local requests, can progress to SA-G to request access to the router's crossbar. SA-G arbitrates between the aforementioned local flits and SSRs. Depending on the priority policy (described in Section 2.2.3.B), one type of request has priority over the other. SA-G prepares the switch for the corresponding flit and enables the bypass of the input unit if the winner is an SSR. SA-L is an $N \times \#VCs : M$ allocator, where N is the number of inputs, $\#VCs$ the number of VCs and M the number of outputs.

SA-G is an $N \times HPC_{Max} : M$ allocator. Figure 2.6 only represents the SSR input signals of the west input port due to space limitations. The rest of the signals, i.e., flits and output SSRs, are represented per in/out port.

2.2.1.C) Output unit

As mentioned in the introduction of Section 2.2, SMART uses asynchronous repeaters to reduce the delay of long wires. These repeaters are located at the output unit, replacing the pipeline register found in traditional and single-hop bypass routers. With this organization, packets can traverse the router without being latched, merging the ST and LT stages as shown in Section 2.2.2. Another change, in some way related with the output unit, is the additional output signal per direction required to propagate the SSRs. These are similar to LAs of single-hop bypass routers, just including asynchronous repeaters to cover the HPC_{Max} distance. Each SSR wire interconnects the router with each of the neighbors routers in a range of $HPC_{Max} - 1$, as shown in Figure 2.5.

2.2.2) Pipeline organization

The original pipeline of SMART described in [Krishna2013] consist of 3 cycles, the first two cycles for preparing a multi-hop and the last cycle to perform it. The pipeline is described next and depicted in Figure 2.7. The original pipeline implements VS, which has been presented in Section 2.1.3.C, but we use VA to simplify the comparison with traditional routers (Section 1.4.1.D). Section 2.2.3.A explains the role of VS in multi-hop bypass NoCs.

Stage 1) **LA-RC, VA & SA-L and BW**. Assuming that a router just received a flit in the previous cycle, the router writes the flit in the designated VC buffer (in VA of the upstream router or node). In parallel, the flit places a request in SA-L and VA. Like occurs in single-hop bypass routers, VA is executed in parallel but not speculatively. Additionally, the router computes the next multi-hop route if the flit is the packet's head. Notice that, to execute LA-RC and SA-L in parallel, head flits have to carry the LA route, so they can request the correct output port in SA-L. Otherwise, the computation of SA-L has to be done sequentially after LA-RC, extending the length of the stage. In addition, the router prepares the SSRs corresponding to the winners of SA-L.

Stage 2) **SSR and SA-G**. The SSRs prepared in the previous cycle are broadcast to the routers in the range of the next multi-hop. Local flits that won SA-L in the previous router request access to the switch in SA-G. Neighbor routers compute SA-G after receiving the SSRs. Routers in which local flits have won SA-G only set up the switch, and those in which SSRs have won set up both the bypass path and the switch.

Stage 3) **ST and LT**. Flits traverse the router's switch and the links until finding the first bypass path disabled.

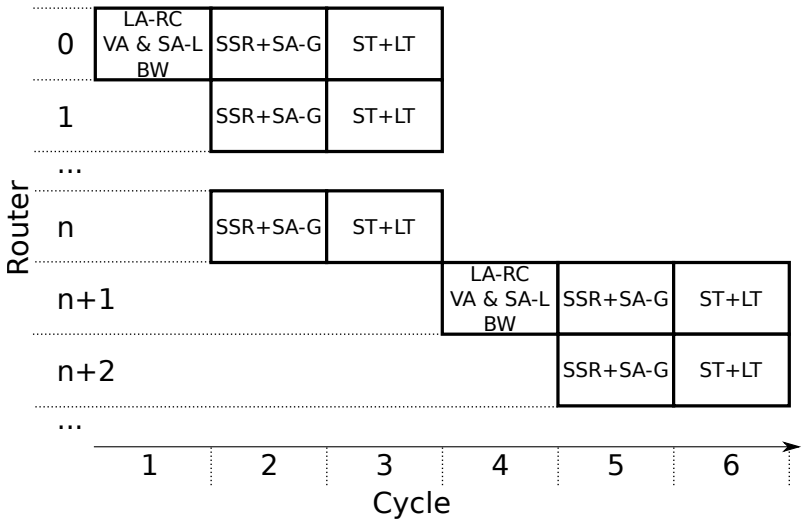


Figure 2.7: SMART pipeline.

2.2.2.A) Pipeline walk-through

This section uses a walk-through example to explain the behavior of the pipeline in detail. Figure 2.8 shows the walk-through of two packets in a 6-ary 1-mesh. The packets traverse the network as follows:

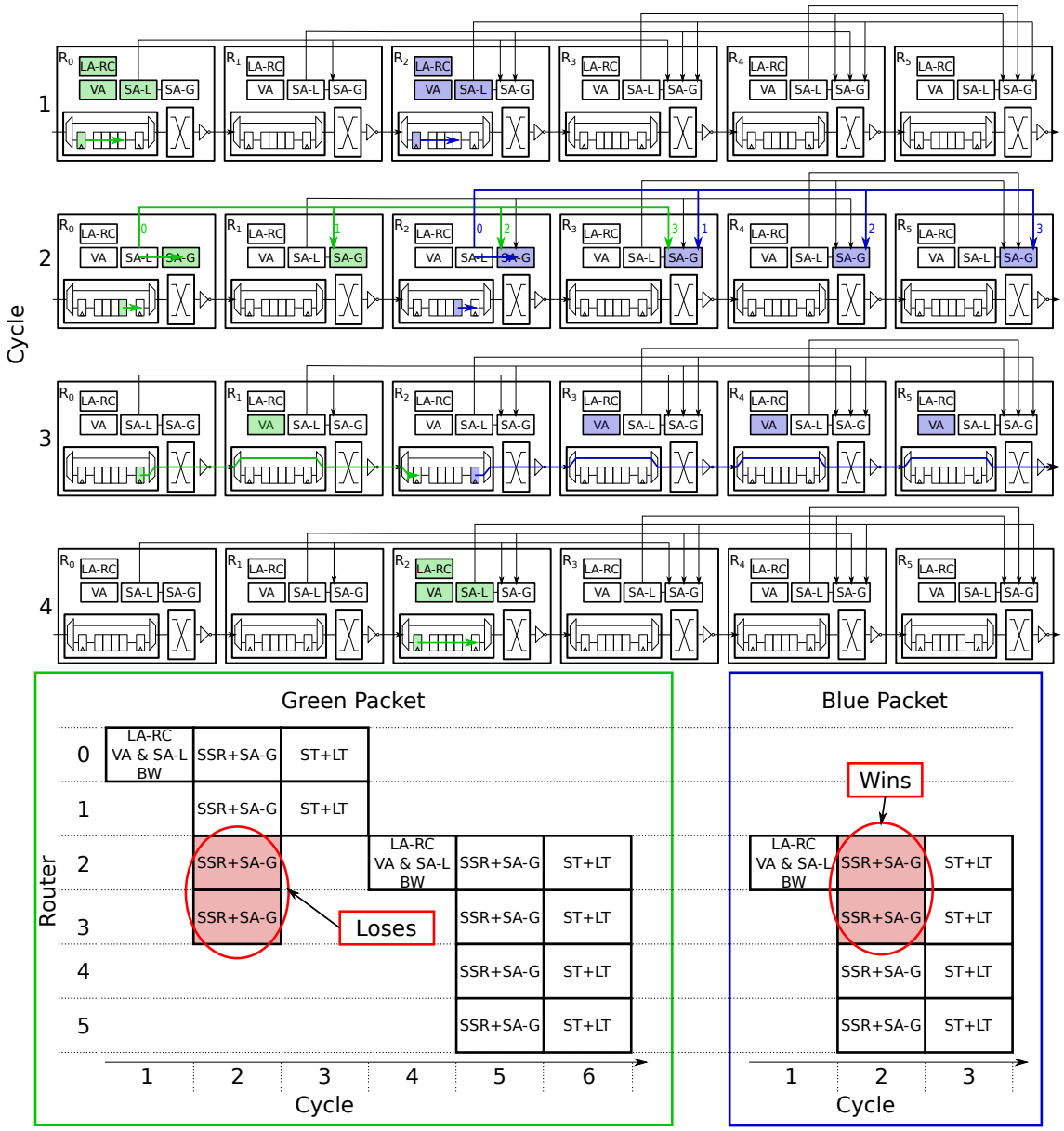


Figure 2.8: SMART's pipeline walk-through. The *SSR+SA-G* stages highlighted in red depict conflicts between SSRs of the green and blue packets.

Cycle 1) The initial state of the network assumes two single-flit packets: one in R_0 (highlighted in green) and the second in R_2 (highlighted in blue). The destination of both packets is beyond R_5 and HPC_{Max} is 4. Both complete the first pipeline stage computing the next multi-hop route (LA-RC), winning SA-L, acquiring a destination VC (VA), and being buffered in the input VC buffer (BW).

Cycle 2) R_0 and R_2 broadcast the SSRs of the green and blue packet respectively. The

SSR generated by R_0 arrives to R_{1-3} but not to R_4 because this router can only receive the packet since its distance to R_0 is 4 hops (HPC_{Max}). The SSR generated by R_2 arrives to R_{3-5}. The SSR of the green packet wins in R_{0-1}, while the one of the blue packet in R_{2-5}. In R_{2-3}, the blue packet wins SA-G to the green because the network gives priority to local flits (Prio=Local in Section 2.2.3.B). In other words, the lower the distance between the receptor and the sender of the SSR, the higher the priority. For example, in R_2 the distances to the start routers are 0 for the blue packet and 2 for the green.

Cycle 3) Both packets perform the multi-hop. The blue packet exits the depicted part of the network, while the green packet stops at R_2 because the bypass is disabled from the loss in the previous cycle.

Cycle 4) The green packet, in R_2, starts the same process to perform the next multi-hop, which effectively occurs in cycle 6 according to the diagram of Figure 2.8.

2.2.3) Implementation details

There are important implementation details to consider in order to obtain a performant and balanced pipeline. Additionally, alternative mechanisms have been proposed to reduce wiring overhead or improve the efficiency. These aspects are presented next.

2.2.3.A) Virtual Channel Selection

The previous sections show an implementation of SMART with Virtual Channel Allocation (VA) to allocate VCs of adjacent routers to packets. In terms of functionality, there is no reason to avoid using VA. However, the original implementation of SMART uses Virtual Channel Selection (VS) to reduce the cycle time as its implementation is simpler. VS has been already introduced in Section 2.1.3 for single-hop bypass routers. It consists of allocating an input VC to packets when they arrive at a router, instead of allocating a destination VC in the upstream router before forwarding them. The communication between neighbor routers uses on/off signaling instead of credits. The on/off signals are denoted *free_vc* and indicate that there is at least an available input VC for the packet in the upstream router. The downstream router has a pool of free VCs and, if there is at least one VC in the pool, it enables the *free_vc* signal. For body flits, the receiver router has a hash table in the input unit, indexed by the packet's source router, to determine the input VC assigned to each packet. This table has a size equal to the number of multi-flit VCs. SMART requires VCs with enough space to store entire packet, because routers that retransmit packets do not know what is the state of the other routers involved in a multi-hop.

2.2.3.B) SA-G arbitration policies

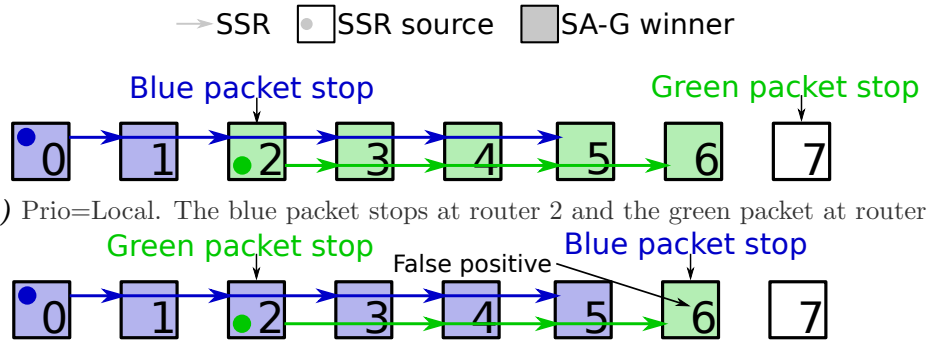
As mentioned in Section 2.2.1, Switch Allocation Global (SA-G) assigns the switch and bypass to SSRs. To ensure the correct operation of the SMART NoC, all the routers must implement the same priority policy. Otherwise, packets may stall waiting for resources while the downstream router is waiting for a specific packet; or vice-versa, a downstream router receives a not expected packet.

The two priority policies proposed in [Krishna2013] are: Prio=Local (priority to local flits) and Prio=Bypass (priority to the bypass). In both cases, the decision criteria is the

distance in hops between the routers that broadcast and receive an SSR. The distance is determined by the input SSR port, as each of the ports is connected to a specific upstream router. This distance must not be confused with H (information carried by SSRs), which indicates the requested multi-hop length. Prio=Local gives preference to SSRs with a closer origin, i.e., the priority is inversely proportional to the SSR distance. Prio=Bypass gives preference to SSRs with the farthest origin, i.e., the priority is proportional to the SSR distance.

Figure 2.9 shows an example with two packets in a 1-ary 8-mesh when using both policies. When using Prio=Local (Figure 2.9a), the blue packet loses SA-G in routers 2-5 against the green. In total, both packets perform seven hops: two the blue packet, and five the green. When using Prio=Bypass (Figure 2.9b), the blue packet wins in all the routers, but the green packet cannot progress. Therefore, only the blue performs six hops. As this example shows, Prio=Bypass does not maximize the total number of hops done per cycle, experiencing a reduction in the maximum achievable throughput. Krishna et al. demonstrate in [Krishna2013] the superiority of Prio=Local over Prio=Bypass.

Figure 2.9b serves also as an example of false negatives. False negatives occur when an SSR wins SA-G, the router sets up the bypass, but it does not receive the corresponding packet in the next cycle. This kind of situation occurs when at least one of the previous upstream routers involved in the multi-hop loses SA-G. There are different reasons for such a situation, like a lack of a free VC in the adjacent router or a conflict with other SSR or flit. In the example of Figure 2.9b, this occurs in router 6, in which the green packet has won SA-G but cannot reach the router because it has lost against the blue packet in routers 2-5. The existence of false negatives does not produce correctness issues.



(a) Prio=Local. The blue packet stops at router 2 and the green packet at router 7.

(b) Prio=Bypass. The green packet does not move and the blue packet stops at router 6.

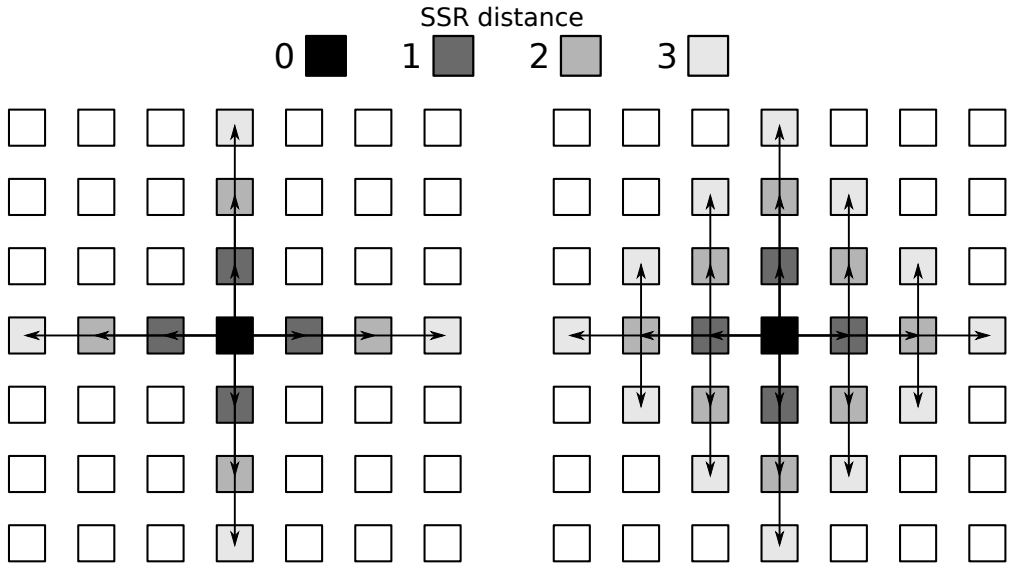
Figure 2.9: SA-G arbitration policies. Boxes represent routers; arrows illustrate SSRs; circles within the routers indicate the packet initial location; shadowed boxes represent that the packet corresponding to the color has won SA-G in that router.

2.2.3.C) Multi-hop traversal in multi-dimensional networks.

Previous sections used one dimensional meshes to put the focus on the router micro-architecture and pipeline. However, multi-hop traversal in multi-dimensional networks introduces some complexity. Two types of SMART implementations are proposed in [Krishna2013] depending on whether packets can take the bypass or not at turns: SMART_1D and SMART_2D.

SMART_1D. This implementation only bypass packets along the dimension, i.e., packets cannot do a dimension change during a multi-hop. SMART_1D only broadcast SSRs

along one dimension as shown in Figure 2.10a. In this implementation, conflicts between SSRs only arise when they are broadcast in the same direction, which are resolved by applying one of the arbitration policies described in Section 2.2.3.B. The main disadvantage of this implementation is that when a packet has to turn to another dimension it is always buffered. Therefore, assuming an ideal HPC_{Max}, packets that have to travel through both dimensions have to do at least two multi-hops to reach their destinations.



(a) SMART_1D. ***(b)*** SMART_2D.

Figure 2.10: SSR propagation on SMART_1D and _2D in a 7-ary 2-mesh with DOR-XY.

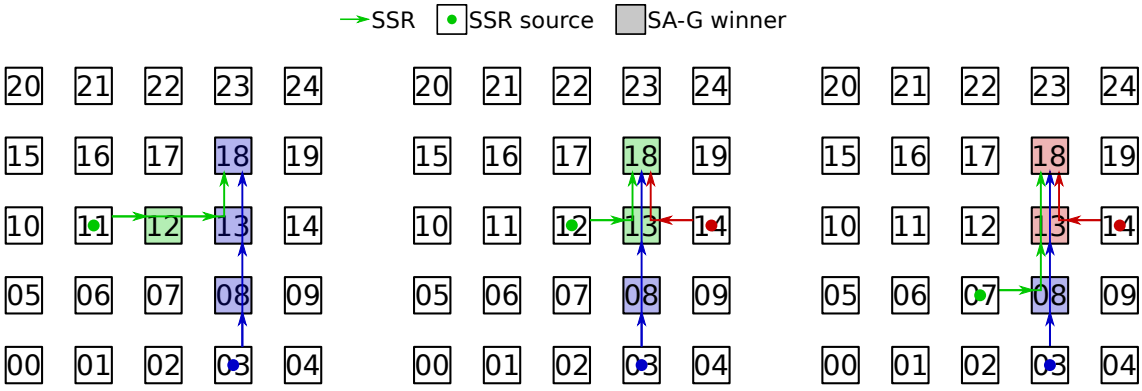
SMART_2D. This type of SMART allows the bypass in any direction. The name is specific for two-dimensional topologies. With an ideal HPC_{Max}, bypassing routers at turns allows every packet in the network to go from source to destination in just one multi-hop.

However, in order to cover the SSR broadcast range, the number of dedicated wires grows quadratically as depicted in Figure 2.10b. This growth also affects the micro-architecture of SA-G, increasing the number of SSR entries per input port and requiring an extra arbitration layer. The additional entries come from neighbor routers that are not connected in a straight manner. The extra arbitration layer is required because a SA-G can receive SSRs from different routers that are at the same distance, unlike in SMART_1D. This extra layer gives priority based on the direction of the SSR, only in those cases in which the distance to the source router is the same. Any priority scheme is valid as long as it resolves this type of conflict and enforces the same priority in every router. The following scheme is chosen by Krishna et al. in [Krishna2013]: Straight > Left Turn > Right Turn.

Figure 2.11 shows some arbitration examples when employing the previous scheme. The first example (Figure 2.11a) shows two packets (green and blue) competing in SA-G in R_{13} and R_{18}, in which the blue packet wins as its SSR travels straight. The second example (Figure 2.11b) shows three packets (green, blue and red) competing in SA-G of R_{13} and R_{18}. The green packet wins because its SSR turns to the left vs the right turn of the red packet. The blue packet loses against both because the distance to the source router is higher (Prio=Local). In the third example (Figure 2.11c), the red packet wins

because it is the one with the closest SSR source to R_{13} and R_{18}, so the extra priority layer is not applied.

Another important consideration to keep in mind is that SSRs in SMART_2D have to propagate the destination of the packet, i.e., the route. Otherwise, intermediate routers of a multi-hop cannot determine which output port an SSR is requesting, as there are multiple possibilities in this type of SMART.



(a) Straight vs Left. ***(b)*** Left vs Right. ***(c)*** Prio=Local.

Figure 2.11: SMART_2D examples of Straight > Turn Left > Turn Right.

2.2.3.D) Bypass at the destination router

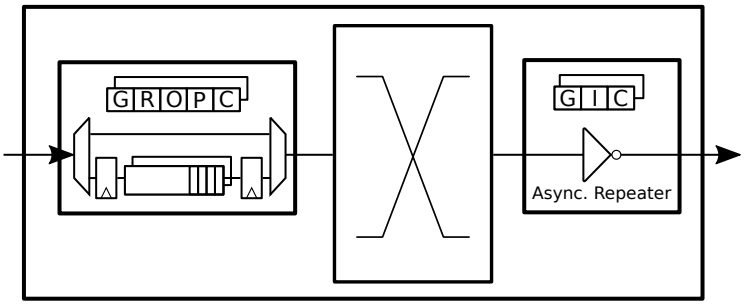
The previous section has shown that a NoC has to implement SMART_2D in order to allow packets to change the traveling dimension during a multi-hop. This also applies to packets using the bypass in their destination routers, as they are changing the traveling dimension when using the ejection ports. Krishna et al. [Krishna2013] proposed an optimization for SMART_1D to bypass the destination router too. In order to do so, SSRs have an additional *ejection* bit that indicates that the multi-hop being requested reaches the packet's destination router. Thus, if an SSR arrives to a router with this bit enabled and an H value equal to the distance to the SSR source router, then it can set up the bypass and the crossbar.

2.2.3.E) Buffer bypass vs router bypass

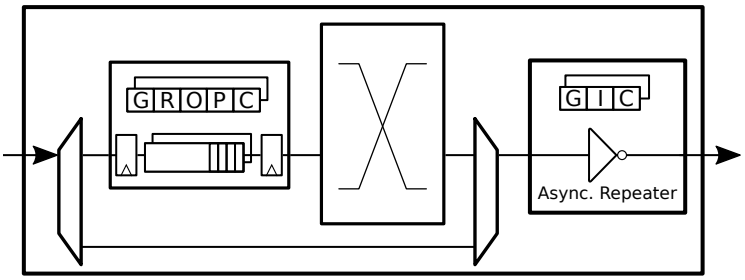
We consider two possible bypass implementations: *buffer bypass* or *router bypass*. Figure 2.12 depicts a diagram of both implementations. In *buffer bypass*, the bypass path connects the input port with the switch of the router, thus it bypasses the buffer and allocation units. The router micro-architecture depicted in Section 2.2.1 implements this type of bypass. Alternatively, *router bypass* connects the input ports with the opposite output ports, e.g., the West input to the East output. Hence, *router bypass* skips the buffer, the allocation units and the switch. The main difference is the placement of the output demultiplexer: in *buffer bypass* it is placed before the switch while in *router bypass* after it. Additionally, SA-G does not have to set up the switch when a non-local SSR wins when implementing *router bypass*.

Buffer bypass is required when packets can change the traveling dimension through the bypass path. This is the case of single-hop bypass routers, SMART_2D (Section 2.2.3.C), and the destination router bypass optimization (Section 2.2.3.D). The benefit of *router*

bypass is that it removes ST from the bypass path, which may increase HPC_{Max} or the operation frequency. However, it is only recommended for SMART_1D as it builds dedicated paths to interconnect the input ports with the output ports. In the case of SMART_1D, this is just one path per direction (West → East, North → South, and vice versa), whereas SMART_2D requires a path per port combination complicating the logic.



(a) Buffer bypass.



(b) Router bypass.

Figure 2.12: Schematics of *buffer bypass* and *router bypass*.

Chapter 3

BST: Bypass Simulation Toolset

This Chapter introduces BST [Perez2020], which is used in Chapters 4-6 to evaluate the details of the proposed mechanisms to improve bypass based NoCs. BST, which stands for *Bypass Simulation Toolset*, is a set of tools that we have created during the development of the bypass mechanisms for their evaluation. This chapter starts in Section 3.1 with an introduction to the alternatives for simulation infrastructure and methodology used to evaluate NoCs. Section 3.2 describes BST without entering in details about the implemented models, as these are described in the next chapters when explaining the experimental methodology of each mechanism. The chapter ends in Section 3.3 with a brief comparison with other NoC evaluation tools.

3.1 NoC modeling and evaluation tools

Beyond simple analytical models, such as the ideal zero-load latency introduced in Chapter 1, the most relevant NoC evaluation tools can be characterized by three aspects: the type of model implemented; the kind of injected traffic; and the metrics measured. These aspects are discussed next.

3.1.1) Type of model: RTL vs software

The two types of NoC models that are most widely employed are RTL and software models. RTL (Register Transfer Logic) is the closest model to reality, since real circuits can be created from them. This kind of model is used in conjunction with tools such as Synopsys Design Compiler and Cadence Genus Synthesis to create the final design, including simulation tools to evaluate the behavior of the circuit and measure timing, analyze the critical path, and evaluate the area and power consumption. Xilinx Vivado and Intel Quartus are similar tools for FPGA synthesis, which are commonly used to test and debug designs as well as perform initial estimations of the maximum operation frequency, area (FPGA resources), and power. However, the low abstraction level of these models increases drastically the development and simulation times compared with alternative software models. Besides, these designs have more limitations and more rigid configurations.

By contrast, software models present a higher abstraction level that gives more flexibility, eases the development and debug processes, and are various orders of magnitude faster. However, their main downside is that they require a validation process with an RTL model to verify their accuracy. For this reason, this kind of model is ideal at the early stages of development to get an estimation of the viability of new designs and at intermediate stages, after the validation, to mass-produce fast simulations for setup tuning or debugging.

3.1.2) Type of traffic

The traffic injected into the NoC model can be essentially divided into two categories: synthetic traffic and real traffic.

Synthetic traffic is generated using analytical models. Due to its simplicity, it is ideal to measure the overall performance of NoCs and compare mechanisms in a wide range of configurations. Besides, it is useful to analyze and debug NoCs, as designers can predict their behavior for specific traffic patterns. Thus, designers can create traffic patterns to test NoCs under specific conditions, while with complex patterns generated from real applications, it is not feasible or very time-consuming. Table 3.1 summarizes most of the synthetic traffic patterns described in [Dally2003].

Table 3.1: Common synthetic traffic patterns. Notation: s_i, d_i are the source and destination nodes of a packet. N is the number of nodes, i identifies the bit in position i of the node index and $b = \lceil log_2 N \rceil$, i.e., the total number of bits in a node index. k only applies to k-ary n-topologies such as the mesh, the torus or the FBFLY, representing the dimension size.

Traffic pattern	Description
Random-Uniform	$d_i = random(N)$
Bit-Complement	$d_i = \neg s_i$
Bit-Reversal	$d_i = s_{b-i-1}$
Shuffle	$d_i = s_{i-1 \bmod b}$
Transpose	$d_i = s_{i+b/2 \bmod b}$
Tornado	$d_i = s_i + \lceil k/2 \rceil - 1 \bmod k$
Neighbour	$d_i = s_i + 1 \bmod k$
Hotspot in x	$d_x = s_i$

Besides the traffic spatial pattern, there are other characteristics to model like the injection process, the size of the packets, the packet classes and their relationship, and the message order. There are multiple factors that determine these characteristics in a CMP, for example the cache coherence protocol as mentioned in Section 1.4.3. It explicitly defines the message classes, which typically are divided into control (requests, invalidations, forwarded requests, acknowledgments, etc) and data messages. Commonly, the link width is equal to the flit size, control packets have one flit, and data packets multiple flits. Additionally, the packet classes have a request-reply relationship, e.g., data messages are generated in response to a request from a cache miss. Moreover, the protocol may require point-to-point message order to operate correctly. All these characteristics can be resembled in synthetic traffic. However, the traffics generated by applications executed

in CMPs are so complex that analytical models created from them [Badr2014] are quite difficult to understand. For this reason, the effects of a NoC running under synthetic traffic in isolation can not been directly extrapolated to a CMP.

Real traffic covers the shortcoming of synthetic traffic by modeling traffic generated by real programs. One of the most common and accurate ways of generating this type of traffic consists in simulating the whole CMP with Full-System (FS) simulations. In an FS simulation, programs are executed in an operative system running over the simulated CMP. This methodology copies the sources of NoC traffic, which are triggered by the memory instructions executed from real programs.

In this kind of evaluation it is essential to choose adequate workloads. In our experimentation we have used the PARSEC benchmark suite [Bienia2008], which focuses on recognition, mining, synthesis and other large-scale multi-threaded programs. These benchmarks are suitable for evaluating NoCs of many-core processors due to their large diversity in terms of synchronization, data sharing, locality, etc. The main benefits of this type of simulation are the possibility of measuring the real impact of a NoC design, analyzing the traffic characteristics (such as spatial and temporal traffic distributions of the offered load), and testing and debugging designs under real conditions. However, their high cost in terms of simulation time and computational resources makes them not suitable for agile development.

3.1.3) Performance vs Costs

NoC models can be categorized also in terms of what they measure. Software models mainly focus on performance at a cycle level, but there are also models to predict area, power or circuit timing. Section 3.3 reviews some of the most relevant.

Regarding RTL designs, suites such as Xilinx Vivado or Intel Quartus include their own tools to measure both performance and costs. The low level of these designs allows obtaining accurate results at the expense of higher simulation times.

3.2 Bypass Simulation Toolset

BST is a collection of tools developed during the course of this thesis to evaluate our single- and multi-hop bypass NoCs. It is an open-source project that can be found at: `https://www.atc.unican.es/software.html`. BST is composed of four modules, as shown in Figure 3.1.

1. **BookSim** [Jiang2013] is a functional cycle-accurate network simulator widely used in NoC research. BST includes a custom version with single- and multi-hop bypass router models.

2. **OpenSMART** [Kwon2017] is an RTL implementation of SMART. BST includes a custom version with support for SMART++ and S-SMART++ (see Chapters 5 and 6). It also includes some fixes and minor changes in the original SMART implementation.

3. An **API** to integrate BookSim in Full-System simulators or trace-based traffic generators.

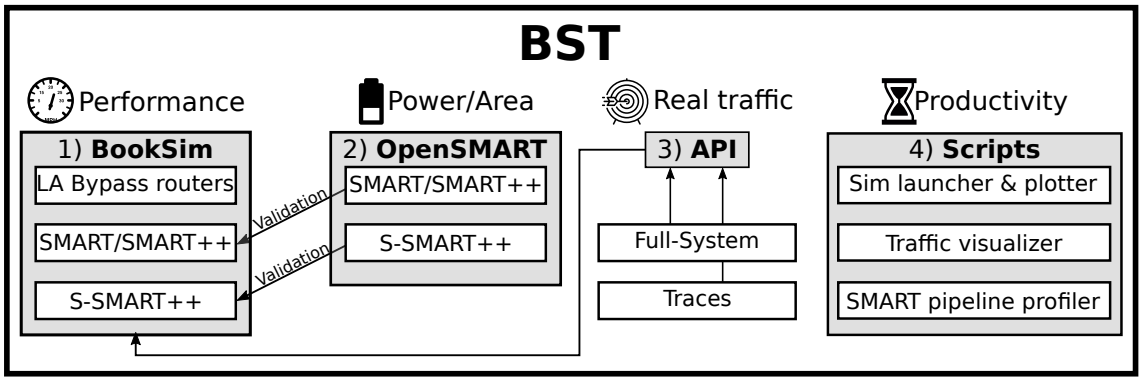


Figure 3.1: BST tools.

4. A set of **scripts** to reduce the effort of creating experiments, producing results charts or debugging NoCs.

Each of these components is described next in more detail.

3.2.1) BookSim

BookSim is a functional (software model) network simulator written in C++. It contains a cycle-accurate model of a traditional Input Queue (IQ) router similar to the one described in Section 1.4.1.D. It has been used in multiple previous works both for simulating system networks [Mubarak2012; Besta2014; Won2015; Jiang2015; Jain2016] or NoCs [Badr2014; Kannan2015; Hesse2015; Hong2016; Alazemi2018]. It has a flexible architecture with the main elements of the network well-structured in independent classes and multiple alternative implementations. It already includes a notable number of allocators, topologies, routing algorithms, buffer policies, etc. It is a standalone simulator that generates synthetic traffic to feed its network models. It already includes a large variety of traffic patterns, as well as allowing the combination of different traffic classes. Each class is characterized by the packet size, injection rate, assignable VCs, etc. This includes the generation of reactive traffic with closed-loop simulations (request-reply) with finite queues, which approximates the behavior of cache coherence protocols mentioned in Section 1.4.3.

It has been integrated with different simulation tools, e.g., in GPGPU-Sim [Bakhoda2009], a GPU simulator to evaluate CUDA workloads. It has been also used to developed SynFull [Badr2014], a synthetic traffic model generator that resembles the behavior of the cache coherence protocol from execution traces of real applications on an FS simulation environment.

However, its IQ router model has only support for some optimizations like speculative VC allocation and lookahead routing, which are not sufficient to model high-performance contemporary NoCs. For this reason, we have extended the simulator with new models of the single- and multi-hop bypass routers described in Chapter 2. From these models we have built the proposals described in Chapters 4, 5, and 6.

Next, we summarize the most significant extensions of the custom version of BookSim provided in BST.

3.2.1.A) Flow control mechanisms

The conventional IQ router of BookSim implements WH with VC support, i.e., VC flow control as mentioned in Section 1.4.1.C. We have extended this router model with the following alternative flow control mechanisms: Virtual Cut-Through (VCT), Bubble flow control [Carrion1997], and Flit Bubble Flow Control Localized (FBFC-L) [Ma2015]. The last two are specifically designed to avoid deadlocks in tori, the former for packet-level allocation and the latter for flit-level allocation. The implementation of the flow control mechanisms relies on credits to monitor the buffer occupancy of adjacent routers in every router model. Single-hop bypass routers are compatible with these mechanisms, but multi-hop bypass routers only with VCT and Bubble flow control as they implement packet-level allocation. Section 1.4.1.C introduced more information about these flow control mechanisms.

3.2.1.B) Bypass routers

The implemented bypass-router models are cycle-accurate and faithful representations of the single-hop bypass architectures described in [Kumar2007; Perez2018; Perez2020a] and the multi-hop bypass NoCs described in [Krishna2013; Perez2019] and Chapter 6. The implementations partially follow the organization of BookSim's IQ router, but their pipelines have a fixed number of stages to avoid over-complicating the model.

BookSim's IQ router has two phases per pipeline stage, evaluation and update. The evaluation phase models the computation of each stage without writing the inter-stage signals to avoid interfering with the next pipeline stage. The update phase modifies the inter-stage signals after all the evaluation phases end. This avoids the propagation of unintended information between stages within a cycle because routers are executed sequentially, unlike in a real system. The bypass-router models in BST follow a different strategy to correctly define cycle boundaries: the pipeline stages of each router are executed sequentially in reverse order, i.e., link traversal, switch traversal, arbitration stages, route computation, flit reception. Pipelines of different routers are separated by channels that introduce a delay, avoiding unintended data communication within cycles. Bypass routers support most of the configurable router parameters of BookSim, like the allocator implementation, the buffer policies, etc. Table 3.2 compiles the most relevant parameters specific for bypass routers.

3.2.1.C) Single-hop Bypass

The enhanced BookSim simulator in BST has five different bypass router models divided into two categories. The classic single-hop bypass explained in Section 2.1, which requires empty buffers when bypassing buffers, has two variants: EVCF and EBB. There are also three variants of NEBB: NEBB-WH, NEBB-VCT and NEBB-Hybrid. These models are explained in Chapter 4.

LAs and LA links are precise models based on the BookSim classes that implement flits and flit links, denoted as channels in the simulator. Among the specific configuration parameters, three are the most relevant: *disable_bypass*, which deactivates the bypass and is useful to obtain reference results; *lookaheads_kill_flits*, which indicates the priority policy of the LA-Arb multiplexers (see Figure 2.2); and *guarantee_order*, which preserves an in-order delivery of packets as required by some coherence protocols.

NEBB-Hybrid has been adapted to support Dynamic input buffer management (shared buffers, implemented as DAMQs, [Tamir1992]). As shown in Chapter 4 with more detail,

Table 3.2: Representative parameters related to bypass routers in BookSim from BST.

BookSim parameter	Description
router	Specifies the router type, e.g., *iq*, *bypass_arb* (single-hop bypass), *smart* (multi-hop bypass), etc.
Single-hop Bypass	
bypass_empty_vc	Specifies if empty destination VCs are required to forward packets
disable_bypass	Used to enable or disable router bypass.
lookaheads_kill_flits	Specifies the priority used in the *LA priority Mux*: Priority to LAs or to local data flits.
guarantee_order	Avoid data reordering caused by buffer bypass requests that target the same output as a packet in a local buffer.
Multi-hop Bypass	
smart_type	Specifies the version of SMART to use from among SMART, SMART ++ (including partial implementations), and S-SMART ++. The router model must be SMART ("router=smart")
smart_max_hops	Sets the maximum number on hops within a multi-hop (HPC_{max})
smart_priority	Indicates the priority.[1]
smart_dimensions	Indicates whether SSRs can be propagated to multiple dimensions ("nD", equivalent to SMART_2D in k-ary 2-meshes) or not ("oneD", which is SMART_1D).

Hybrid combines WH and VCT virtual channel allocation, and packets may have *bubbles* caused by channel interleaving. Buffer management must be aware of which flow control is used by each bypassed packet to prevent deadlock. When bypassing a packet using VCT, buffer slots are reserved for the whole packet, i.e., credits are reduced by the packet size. This guarantees that any packet that starts advancing to a buffer will have space for it, regardless of any bubble. In contrast, when transferring a packet using WH, credits are reduced flit by flit (both for bypass and non-bypass paths).

Besides the standard statistics of BookSim, simulations using single-hop bypass routers provide the following new metrics: *bypass utilization*, as the ratio of the number of hops that use bypass paths over the total number of hops done; *buffer* and *crossbar conflicts*, as the number of times that these resources are not available for flits or LAs; *number of SA-O winners killed by LAs*, as the number of local flits that win SA and are later killed due to a conflict with LAs in LA-Arb.

3.2.1.D) Multi-hop Bypass

The BST version of BookSim includes three main multi-hop bypass routers representative of SMART [Krishna2013], SMART++ [Perez2019] and S-SMART++ [Perez2021]. The last two NoC are explained in Chapters 5 and 6. Partial versions of SMART++ evaluated in [Perez2019] and Chapter 5 are supported too. All the models implement VCT flow control. However, SMART++ and S-SMART++ are the only mechanisms that guarantee consecutive reception of packet flits, since they allocate buffers, switches and bypass paths

[1] Only "Prio=Local" is implemented due to its superior performance but it is prepared to add new policies.

on a packet by packet basis, instead of flit by flit.

As explained previously, flits can traverse multiple routers and links in a single cycle thanks to multi-hop bypass. However, each channel in BookSim requires a minimum delay of one cycle, making impossible the traversal of multiple links in a single cycle without modifying the structure of BookSim to a large degree. Instead, these router models account for the delay of multi-hop link traversals in each of the routers involved. In a multi-hop path, each router calls the reading function of the next router that evaluates whether its bypass is enabled or not. If it is enabled, the same procedure occurs with the subsequent router, until finding a disabled bypass or reaching the end of the multi-hop. The flit is directly saved in a pipeline register in the last router, prepared for BW in the following cycle.

Additionally, SSR channels are not modeled, as they would introduce significant modifications to already implemented topologies or topology replications just for this type of NoC. Instead, SSR signals are placed directly in the routers within each multi-hop, after wining the local switch allocator. Despite not implementing detailed models of SSR links, this implementation is cycle-accurate and functionally equivalent to the actual proposed implementation as we demonstrate in subsequent chapters.

SMART simulations employ three additional parameters to select the version of SMART, to define HPC_{max} and indicate whether the bypass is only along the traveling dimension (SMART_1D) or along multiple (SMART_2D). These parameters are indicated in Table 3.2. In addition to the default statistics of BookSim, SMART simulations include: number of multi-hops; length of each multi-hop; and bypass utilization as the ratio of flits bypassed over the total number of flits transmitted by each router.

3.2.2) OpenSMART

OpenSMART [Kwon2017] is an RTL implementation of SMART written in Bluespec System Verilog. BST includes a custom version of OpenSMART with implementations of SMART++ and S-SMART++. The original OpenSMART implementation only supports single-flit packets, and the same goes for the models included in BST. The pipeline model presented in OpenSMART significantly differs from the original proposal in SMART. For example, ST is implemented in the second stage after winning SA-L, instead of the third stage as described in Section 2.2.2. This complicates the comparison between the functional models in BookSim and the results from Bluespec. Additionally, initial testing of the OpenSMART models presented execution errors, with packets that were missed or not delivered to their correct destinations, preventing any productive use of the tool[2]. To be able to compare the BookSim and OpenSMART models, we have reorganized the router stages. Our implementation follows the original organization from SMART. Additionally, our model corrects some errors and provides working models that deliver all packets to the correct destinations. Some specific changes are detailed next.

Several issues with Bluespec rules prevented successful compilation, specifically cyclic dependencies of rules. Rules are the main coding block of Bluespec to describe how data is moved from state to state. The Bluespec compiler detects data dependencies and schedules rules in an appropriate order, which may be explicitly given in the code. Cyclic dependencies in these rules may prevent compilation. The following changes simplified the implementation and mitigated the identified issues. First, OpenSMART employs a custom library to implement FIFO structures. These modules have been replaced with the Bluespec built-in modules FIFO and FIFOF (the latter including explicit *full* and

[2]Using the most recent repository version at the time of writing, commit d4f5095.

empty signals). These changes are implemented in the Credit Unit, Input Unit, Smart Flag, Smart Router and Traffic Generator Units. Similarly, the CReg module that implements an Ephemeral History register [Rosenband2004] has been replaced by explicit combinational logic, which is less prone to generate cyclic rule dependencies when modifying code.

The implementation of SMART++ and S-SMART++ essentially required modifications to the VC Selector (VS). The original SMART VS implements a pool listing free VCs in the neighbor router. An empty VC is extracted when a packet wins SA-G, and inserted when a credit is received. The implementation is quite simple as SMART requires empty destination VCs to forward packets, ignoring buffer depth and occupation. In contrast, SMART++ employs buffer depth and occupation. Flow control credits, already implemented in OpenSMART, are leveraged to monitor the available space in each VC. The increase in logic is very moderate.

S-SMART++ additionally requires logic to handle speculative SSRs (details are explained in Chapter 6) and its corresponding bypass route. OpenSMART implements *router bypass* instead of *buffer bypass* like the original proposal of SMART (Section 2.2.3.E). As explained, *router bypass* is ideal for SMART_1D, which is what OpenSMART implements, but this is not the case for S-SMART++ as it will allow packets to take the bypass when changing dimensions like SMART_2D. The S-SMART++ implementation also uses *router bypass* despite being suboptimal, given the complexity of altering this and other related parts of the design, such as SA-G. Thus, the implemented design has dedicated bypass paths from each input port to all the output ports, including the respective logic to control them.

3.2.3) API

BookSim is a standalone simulator but has been integrated in different full-system simulators like GPGPU-sim [Bakhoda2009], GEMS [QinhongZhang2015], gem5 [Li2016; Das2018] or FeS2 (Simics) [Magnusson2002; Ma2015]. However, most of the times the integration is not made available publicly or is hard to maintain because of changes in the FS simulator or BookSim.

BST provides an API to simplify the integration of BookSim with other simulation environments, such as gem5. The API abstracts the internal structure and presents simple commands to interact with the simulation tool. This section explains the integration with gem5, assuming that the Ruby memory hierarchy model is employed. Ruby defines the elements of the memory hierarchy, the cache coherence protocol, and the NoC.

The API supports the compilation of BookSim as a library, instead of integrating its code into the other simulator like in GPGPUSim [Bakhoda2009]. The main advantage of linking BookSim as a library is the automatic synchronization from the standalone version to the integrated one. Additionally, this decouples the development and maintenance of both tools, and speeds up the (time-consuming) compilation process when changes are done only to one simulator.

Some code changes required to support the API and the compilation as a library include a new namespace that encapsulates the complete BookSim code; a new compilation target for the library version; and a new traffic manager class to interact with the API functions.

3.2.3.A) API functions

The API, defined in the *BookSimWrapper* class, consists of four main functions:

GeneratePacket creates a packet if there is enough space in the injection queue of the specified node. This method needs the following information:

- An integer that identifies the **source** node of the packet.
- An integer that identifies the **destination** node of the packet.
- The packet **size**.
- The **class** of the packet, i.e., the type of message, which is used to identify the virtual or physical network the packet belongs to.
- The **queueing time** of the packet in the network interface. This is the number of cycles since the creation of the packet in the FS simulator.

GeneratePacket returns the packet ID, which is used to keep track of the packets that are inside the network. For example, the interface for gem5 provided within BST, implements a hash table to save the message data and the packet destination in the Ruby domain (the memory hierarchy model of gem5). This information is then retrieved when packets are retired from the NoC with *RetirePacket*, to enqueue the message in the corresponding cache, directory, etc.

RetirePacket retires a packet from one of the ejection queues of BookSim. The type of data that it returns is a structure called *RetiredPacket* which contains the packet ID, the packet's class and size, and statistics like the packet latency or the number of hops done. The packet ID is used to gather the message information from the hash table mentioned in *GeneratePacket*. This function is intended to iterate through until it returns a packet with ID "-1".

RunCycles executes the network model the number of cycles specified in its only argument. The number of cycles is an integer greater than 0. A value larger than 1 cycle means that the NoC (BookSim) has higher frequency than the caches (FS simulator). For example, if the value is 1, the frequency is the same in both domains, while if the value is 2, the frequency of the NoC doubles the frequency of the caches.

CheckInFlightPackets returns a boolean that indicates whether there are packets inside the network or not. The purpose of this function is to avoid the execution of unnecessary cycles in BookSim in event-driven simulators like gem5. Thus, the gem5 interface only schedules events to run BookSim when there are packets. This is useful to reduce computation when the network is empty.

3.2.3.B) Topology mapping

Unlike the gem5 NoC models, like Simple Network, Garnet 2.0 [Krishna2017], or Heterogarnet [Bharadwaj2020], BookSim ignores the topology defined in gem5. The topology modeled in BookSim is defined in its own configuration parameters, the same ones used when running in isolation. This way, the user has the freedom to simulate custom topologies and rearrange the nodes (caches, directories, and DMAs) as it pleases. For example, a user can define a tile per cache, memory, or DMA, and interconnect them as desired by just adjusting the configuration of BookSim. However, this implies that the user must be aware of how Ruby tiles map into BookSim nodes.

Ruby tiles typically comprise caches from different levels of the memory hierarchy. Some of them include a memory directory, and a few may include a DMA controller too. Figure 3.2 shows an example with a 16-tile CMP arranged in a 4×4 mesh. There are two types of tiles represented in light yellow and dark red. Light yellow tiles have a private bank of L1 cache and a shared bank of L2 cache connected to a Ruby Router (RR). Dark red tiles also include a memory directory (DIR) and a DMA controller (typically, only one tile has a DMA controller).

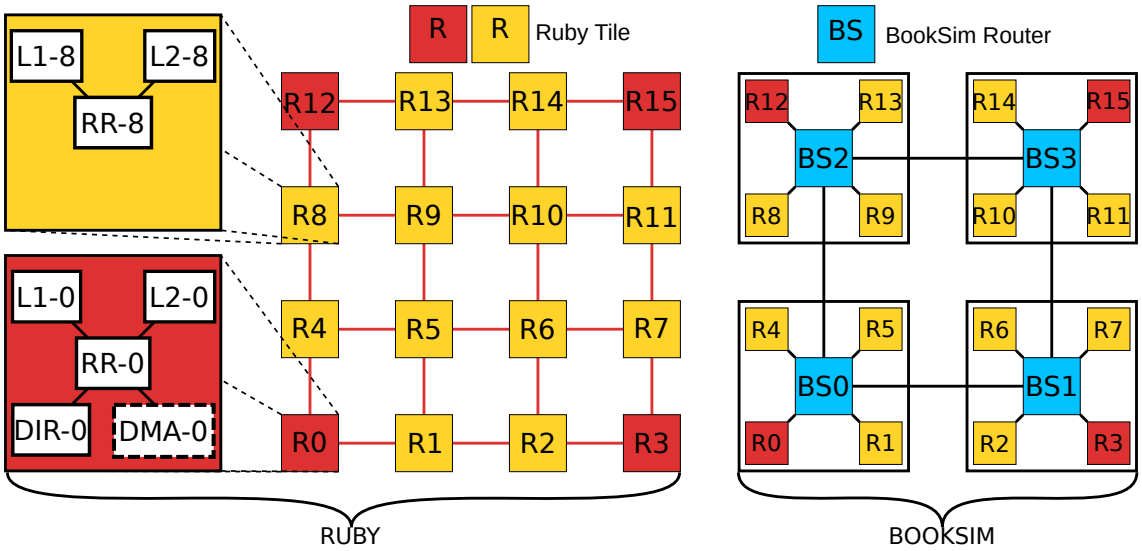


Figure 3.2: Example of 16-tile network mapped as a 2×2 mesh with concentration 4 in BookSim. The only relevant information in the Ruby domain is the location of the caches (L1 and L2), memory directories (DIR), and DMA controllers (DMA). The interconnection topology is ignored.

The topology configuration from gem5 specifies a tile organization in Ruby, assigning a consecutive tile ID to each Ruby Router. The actual gem5 topology is irrelevant, since BookSim only employs the tile ID to map the resources.

A mapping algorithm has been adapted to take into account relative location of each resource in the NoC placement. Figure 3.2 presents an example in which tiles are arranged in a 2×2 mesh with concentration 4. A naive mapping using increasing tile index would map tiles with consecutive IDs to the same router; hence, the first router ($BS0$) would include two Ruby tiles with attached memory controllers ($R0$ and $R3$), altering the tile placement in the chip. The mapping of tiles to BookSim nodes implemented in BST preserves the location of the tiles with memory controllers in the corners of the chip. To achieve this arrangement, the node mapping from the concentrated mesh topology in BookSim has been modified: tiles 0, 1, 4 and 5 are connected to the first router ($BS0$), and so on. BST includes support for the most frequent topologies: meshes, tori and flattened-butterfly, with and without concentration. Users must be aware of this peculiarity when implementing custom topologies as it may have a significant impact on their results.

3.2.4) Scripts

BST includes a set of practical scripts, most of them written in Python. The purpose of the most relevant scripts is the automatization of the most frequent tasks, such as the creation of experiments and the production of charts from the results. Figure 3.3 shows

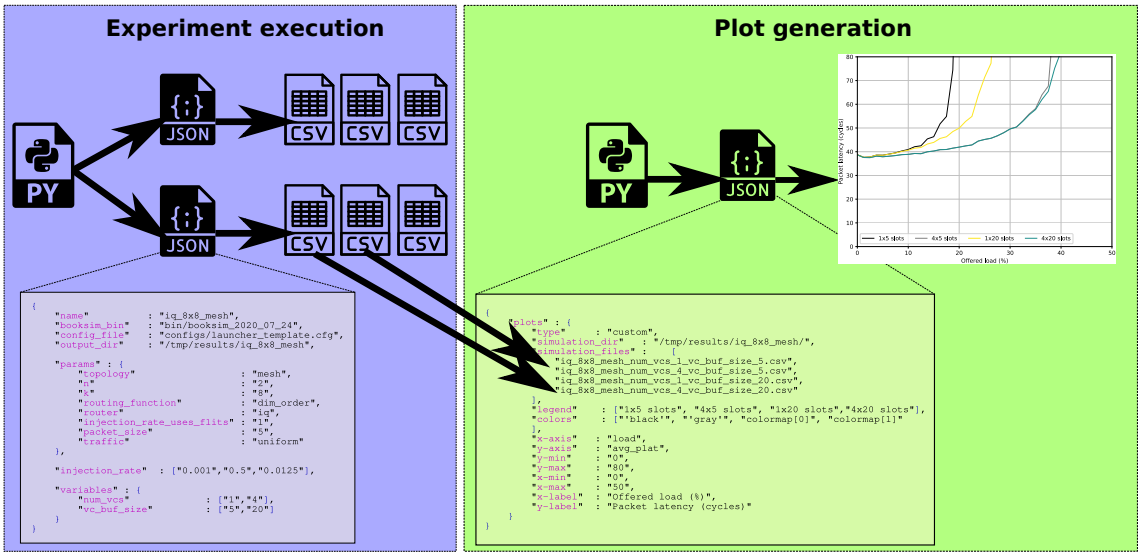


Figure 3.3: Experiment and chart generation workflow.

a diagram of the typical workflow of creating and running an experiment in BookSim (also applicable to gem5 and OpenSMART with minor differences) and generating results graphs. The workflow is divided into two tasks. The first task consists of defining and launching the experiments. We consider an experiment as a set of simulations with the same parameters, except for the injection rate in the case of synthetic traffic. The user can define the parameters of the simulator in a JSON file. In this file there are two types of parameters, constants and variables. The launching script generates an experiment for each combination of parameters. The launcher generates a Comma Separated Values (CSV) file for each experiment with its statistics. This script is specifically adapted to run on the local machine or in a cluster with a SLURM workload manager [Yoo2003]. However, it is easily extendable to other systems.

In the second task, the user creates another JSON file to select experiment CSV files and define the plots. The plotting script uses the Python's Matplotlib module [Hunter2007]. There are also scripts intended to debug the network models, visualize the evolution of the router pipelines or analyze specific behaviors.

3.3 Other NoC evaluation tools

There are numerous open-source NoC models available besides BookSim and OpenSMART. Some examples are Garnet, Noxim [Catania2015], Topaz [Abad2012], CONNECT [Papamichael2012] or DART [Wang2014]. However, as far as we know, besides OpenSMART only Garnet has a patch[3] to include SMART, but it is not fully cycle-accurate.

Table 3.3 summarizes the characteristics of BST and compares with the original versions of BookSim, OpenSMART and Garnet. First, Garnet and BookSim are functional simulators while OpenSMART is a real NoC design written in Bluespec System Verilog

[3]SMART_1D patch for Garnet2.0 from Synergy Lab (Georgia Tech): `https://synergy.ece.gatech.edu/tools/garnet/`

(BSV). All of them have a standard IQ router model but with some differences: BookSim has a detailed model with configurable pipeline that goes from 3 to 5 stages depending on the optimizations enabled; OpenSMART includes a single-cycle router (which increases to two stages when accounting for Link Traversal)[4]; Garnet has a configurable pipeline in which the user can define the router delay.

Table 3.3: Current state of NoC simulators

Characteristic	BST-BookSim	BST-OpenSMART	BookSim 2	Garnet	OpenSMART
Type of model	Functional	BSV	Functional	Functional	BSV
IQ Router	3/5 stage router	single-cycle router	3/5 stage router	1/n-stage	single-cycle router
Single-hop bypass	Standard and NEBB				
Multi-hop bypass	SMART, SMART++ and S-SMART++			SMART	SMART
Accuracy	Cycle accurate	Real design	Cycle accurate	Acc./Approx.	Real design
Simulation speed	Fast	Moderate-Slow	Fast	Fast	Moderate-Slow
Flexibility	High	Low	High	Moderate	Low
FS integration	API (gem5)		GPGPUSim	gem5	

Regarding single-hop bypass routers, they are only available in the custom version of BookSim included in BST. Multi-hop bypass routers are available in OpenSMART and Garnet in their SMART_1D implementation. BST has models of SMART, SMART++ and S-SMART++ in BookSim and OpenSMART, and BookSim also includes SMART(++)_2D variants.

In terms of accuracy, BookSim models are cycle-accurate. BookSim's IQ router has been validated against a real implementation [Jiang2013] and our bypass router models have been validated against the real designs created in OpenSMART. These validations are presented in Sections 5.4.3.A and 6.2.3.A for SMART++ and S-SMART++, respectively. Garnet (versions 2.0 and heterogarnet) models accurately single-cycle routers. However, to model pipelines with more than one stage it just adds extra waiting time at the input units. Given that SMART has a standard pipeline of 3 stages and that the patch is applied over the previous router, this model is not cycle-accurate. BookSim and Garnet are faster and more flexible than OpenSMART given that they are software models. Finally, regarding the FS integration, BookSim is a standalone simulator but thanks to the BST API it can be easily integrated in other simulators, including gem5 for which we provide the integration code[5]. Garnet is a NoC model integrated within the Ruby model of gem5. Garnet has less configurable options than BookSim because it just implements the router and link models. Ruby implements the network topology in addition to the rest of the elements of the memory hierarchy, which limits the scope of Garnet.

Apart from BookSim and Garnet, there are also software models to estimate frequency, area, and power like DSENT [Sun2012] or Orion [Kahng2009; Kahng2015]. These models are several orders of magnitude faster than RTL based estimations. The main advantage of these type of models is that they are technology-independent, so they can be used to characterize NoC designs under different technological parameters. The problem with this kind of models is that they are specific for a given router micro-architecture as the router components are built from standard cell models. Thus, implementing new models for bypass routers, specially with multi-hop bypass that have router components substantially different from a standard router, is a tedious task. DELPHI [Papamichael2015] is a tool

[4]OpenSMART also includes a Chisel version of the single-cycle router.
[5]Last gem5 commit tested: af8d107191cc69a77624e2af34f108dc9c1ff03f

to automatically generate DSENT models, but as it uses RTL designs as input, it does not avoid the process of creating them. For this reason, we only use DSENT to approximate the hardware costs of single-hop routers, given that LA-Arb is similar to SA. To evaluate multi-hop bypass we use Quartus to synthesize and measure the performance of SMART++ and S-SMART++ with respect to SMART in an FPGA.

Chapter 4

NEBB: Non-Empty Buffer Bypass

Non-Empty Buffer Bypass (NEBB) is a mechanism for bypass routers that allows the bypass of buffers that are not empty. Thus, NEBB increases the opportunities to use the bypass, reducing the latency at medium-high loads and HoLB (without requiring VCs), as the bypass paths can be enabled when there are packets blocked in the buffers. This chapter focuses on single-hop bypass router but NEBB is also applicable to multi-hop bypass router as explained in Chapter 5.

The chapter starts describing the packet-interleaving problem in Section 4.1. Next, Section 4.2 introduces NEBB. Section 4.4 describes the most relevant implementation details. Then, Section 4.3 explains how to adapt NEBB to k-ary n-torus when using an efficient deadlock avoidance flow-control. The chapter ends with an evaluation of NEBB in Section 4.5 and a summary of the main conclusions in Section 4.6.

4.1 Packet interleaving in bypass routers

The introduction of a bypass mechanism in traditional routers is not trivial. The VC control casuistic increases considerably with respect to non-bypass routers and the bypass path can only be enabled under certain conditions. Without the application of additional flow-control restrictions to manage the bypass, a problem that we denote as *packet-interleaving* can arise, causing data corruption or router miss-behavior that ends in a deadlock.

This section describes the packet-interleaving problem in Section 4.1.1 and the original restrictions to avoid it in Section 4.1.2.

4.1.1) Packet-interleaving

Packet-interleaving can occur when a multi-flit packet (A) takes the bypass and there is already another packet (B) at the front of the buffer bypassed. The issue happens when only a part of packet A takes the bypass while the second part has to be buffered, occupying a position behind packet B. In this situation there are two possible scenarios:

1. Packet B overwrites the routing information of packet A, which is stored in the VC control registers, after being assigned a destination VC. Thus, the buffered part of packet A will be miss-routed.

2. Packet B cannot be forwarded because packet A cannot release the destination VC until its tail flit is forwarded, causing a deadlock.

Figure 4.1 illustrates the problem caused by carelessly relaxing the bypass conditions with an example. Specifically for this example, packets may bypass a router even when the associated buffer is not empty, and forwarding may occur towards non-empty buffers. In the example, there are three packets, one in each router. The following analysis focuses on the bypass of the blue packet in R_0, which travels East, while gray packets in R_1 and R_2 are blocked waiting for turning South.

Cycle 1) The head of the blue packet performs ST in R_0 while its LA is doing LT. The state of VC_0 is ST (active forwarding a packet), the assigned output port is O_0, and the destination VC is VC_0 with 2 credits left.

Cycle 2) The packet's head does LT, while the tail ST. At the same time, the head LA requests the switch and the bypass of R_1. The gray packet at R_1 cannot advance through the output port O_1 because there are no credits, so the LA wins LA-Arb in R_1 and generates an LA for the next hop towards R_2. The LA can acquire O_0 because the destination VC has at least one credit left (WH). Also, the tail LA does LT towards R_1.

Cycle 3) The head flit takes the bypass in R_1 and performs ST while the tail performs LT towards R_1. Meanwhile, the head LA does LT towards R_2, and the tail LA requests the bypass of R_1 but fails because there are no more credits available for O_0 towards R_2.

Cycle 4) The head flit does LT, and the tail flit advances towards the buffer of I_0 in R_1. However, the gray packet is at the front of the buffer, while the control registers of the VC store the blue packet information. This situation leads to the packet-interleaving error in the network. In this case, the gray packet will overwrite the blue packet routing information from V_0 control registers of I_0 in R_1 once it moves towards O_1, miss-routing the blue packet's tail.

Cycle 5) The head of the blue packet is written in the buffer of I_0 in R_2.

In conclusion, the problem arises when the blue packet's head takes the bypass at R_1 when there is another packet in the buffer to bypass and only one credit left.

Figure 4.2 shows the previous example in a simplified diagram that only shows the buffer state after propagating flits. We use this type of diagram from now on to explain the mechanisms that avoid packet-interleaving. This type of diagram uses the following terminology. *Source* refers to the initial router or buffer that contains a packet to forward. *Bypass* refers to the router or buffer that an LA requests to bypass. *Destination* refers to the final router or buffer of a packet. Notice that the credit information is the same as in non-bypass routers, so the *source router* does not have credit information about the *destination router*.

4.1.2) Avoiding packet-interleaving: empty buffer bypass

Section 2.1.3.A already mentions the 3 conditions to use the bypass proposed in [Kumar2007] to avoid packet-interleaving. In summary, the most important part is that the input buffer that receives an LA has to be empty to allow the bypass (condition 1). The

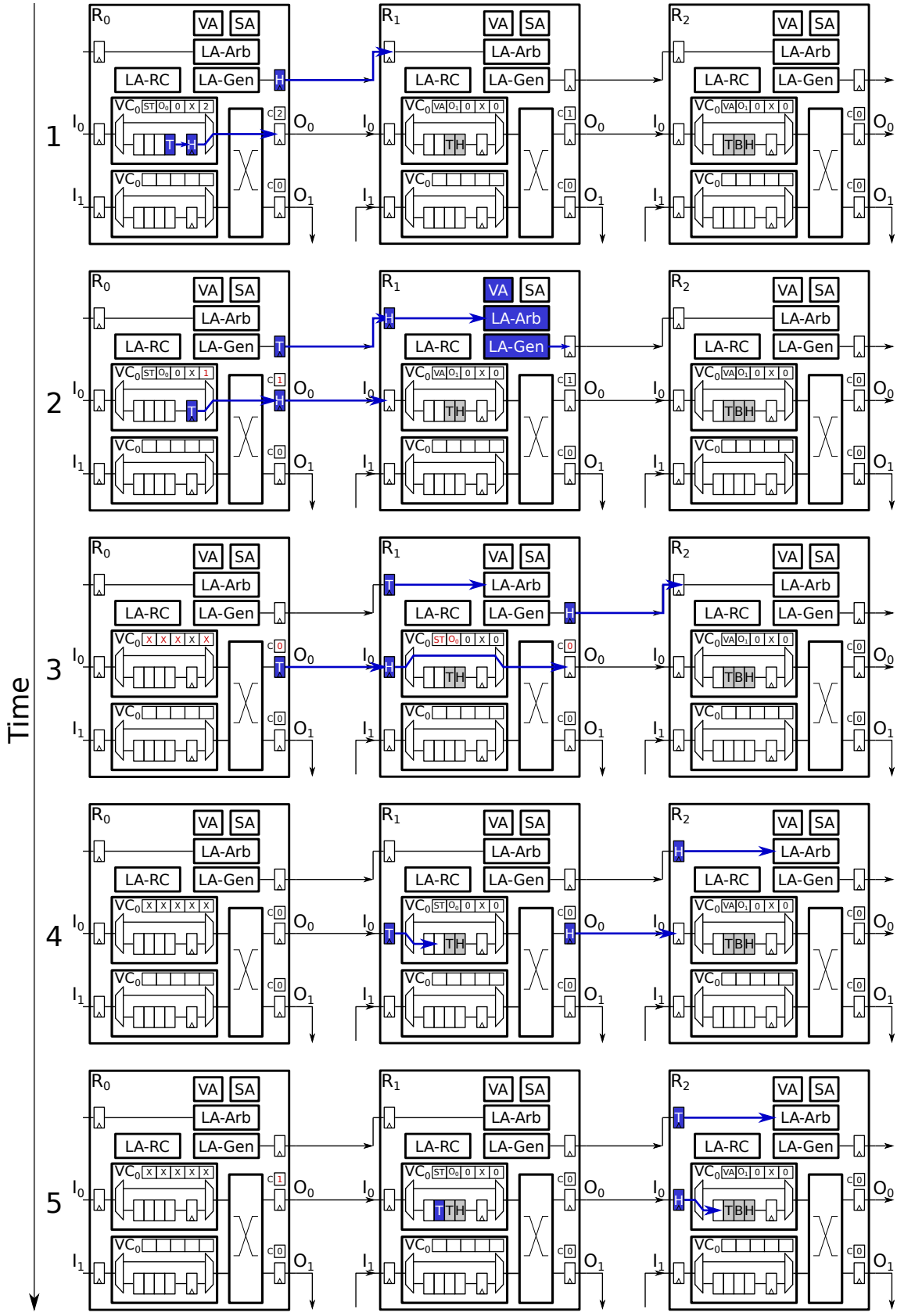
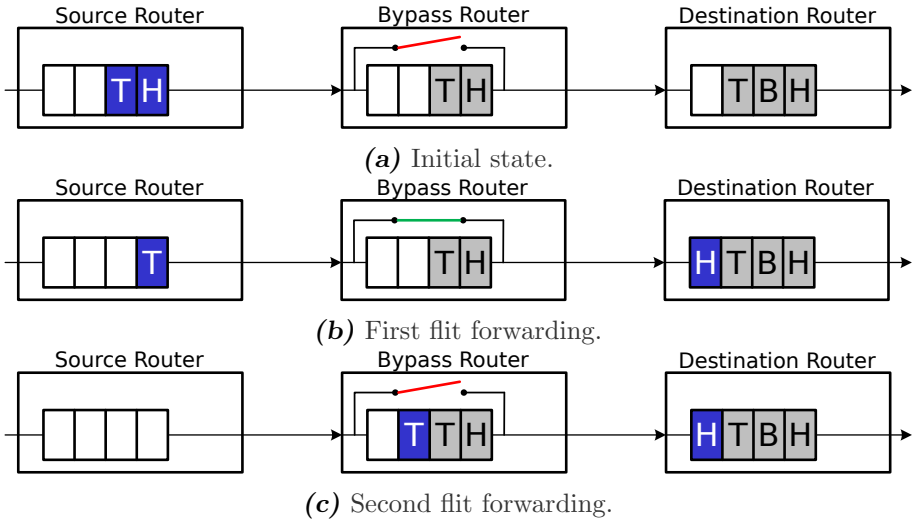


Figure 4.1: Incorrect packet-interleaving example caused by misconfigured bypass restrictions.



(a) Initial state.

(b) First flit forwarding.

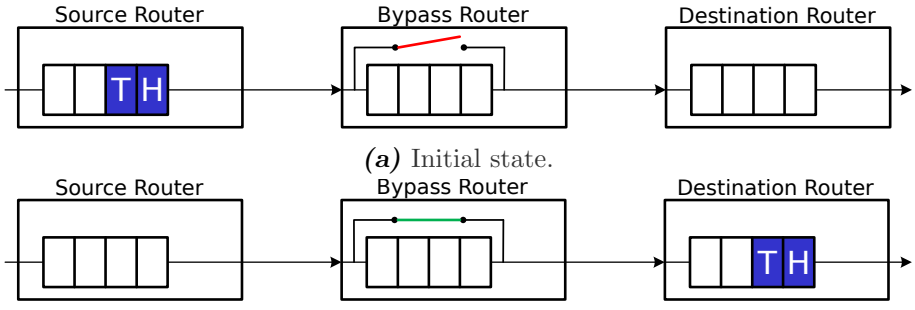
(c) Second flit forwarding.

Figure 4.2: Buffer state of Figure 4.1 showing a packet-interleaving example.

second and third conditions, which ignore LAs in case of a conflict with another LA or local flit, are optional when using LA-Arb. However, the authors of these papers use the term *free VC* when they refer to the state of the destination VC when forwarding packets. This term is ambiguous as it can mean that the destination VC has to be empty or just available, i.e., not in use by another packet and with at least one slot free. Both alternatives are valid to avoid packet-interleaving, but the large number of VCs and the implementation of dynamic shared buffers suggest the first meaning as this type of implementation mitigates the buffer under-utilization of requiring empty VCs. This Chapter explores the implications of both alternatives. Next, we describe both mechanisms.

4.1.2.A) Empty VC Forwarding

Empty VC Forwarding (EVCF) consists of allowing packet forwarding only when there is a destination VC available (not in use by another packet) and empty. This mechanism forces buffers to only store one packet at a time, avoiding the possibility of packet-interleaving. The example depicted in Figure 4.3 shows an example of empty VC forwarding. In this case, the blue packet takes the bypass of the *bypass router* because its buffer and the *destination* one are empty.



(a) Initial state.

(b) Packet forwarding.

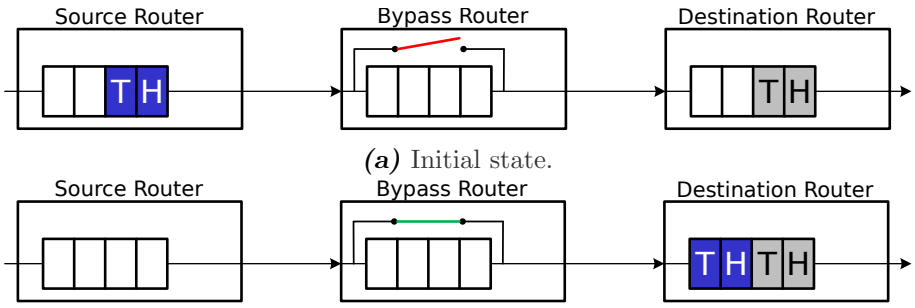
Figure 4.3: Empty VC Forwarding (EVCF) example.

This mechanism does not impose any space requirements in the buffers, in other words,

the minimum buffer space is 1 flit. Additionally, buffer sizes greater than the maximum packet size do not make sense.

4.1.2.B) Empty Buffer Bypass

Empty Buffer Bypass (EBB) only allows the bypass if the buffer to bypass is empty and in other state than active, i.e., the VC is not currently forwarding another packet. A buffer can be empty in active state when a packet is fragmented because of WH. Figure 4.4 shows examples of empty buffer bypass in different situations. In the example, the blue packet takes the bypass in the *bypass router* because its buffer is empty and there is at least room for one flit in the *destination buffer*. Like EVCF, this mechanism does not impose any space requirements in the buffers. However, this mechanism does not limit the maximum buffer size as buffers can have multiple packets simultaneously. In both mechanisms, it is not a problem if only one part of the packet takes the bypass since the second part will be stored at the front of the *bypass buffer*.



(a) Initial state.

(b) Packet forwarding.

Figure 4.4: Empty Buffer Bypass (EBB) example.

4.2 Non-Empty Buffer Bypass

Non-Empty Buffer Bypass (NEBB) is a mechanism that improves the efficiency of the bypass overcoming the limitations of *Empty VC Forwarding* (EVCF) and *Empty Buffer Bypass* (EBB). As the name itself indicates, NEBB allows taking the bypass path even when the buffer to bypass is not empty. EVCF and EBB need to use a considerable number of VCs to benefit from the network's bandwidth and the bypass paths to guarantee that free VCs are available for bypassing when other packets are blocked. NEBB mitigates such Head of Line Blocking without requiring many VCs, by bypassing routers with non-empty blocked input buffers.

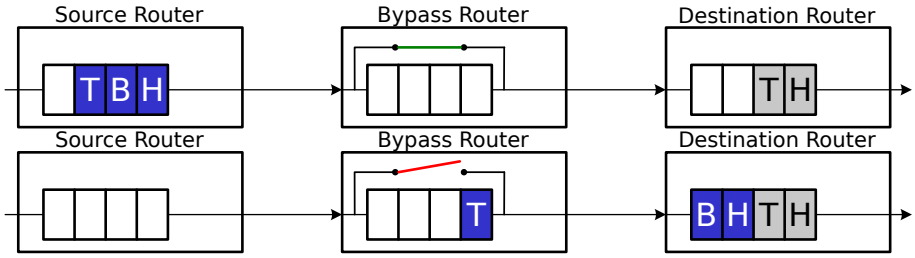
We propose three versions of NEBB inspired in the WH and VCT flow controls. These are: NEBB-WH, NEBB-VCT and NEBB-Hybrid. They are detailed and discussed in the next subsections. In all versions, the buffer to bypass has to be in a state other than active to allow the bypass.

4.2.1) NEBB-WH

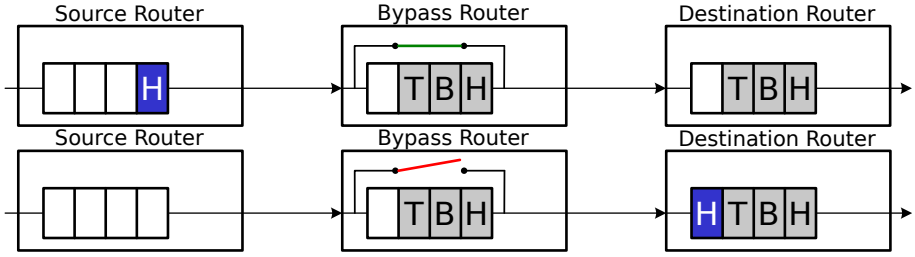
NEBB-WH is based on WormHole, hence the allocations of the switch and the bypass paths are done in a flit by flit basis. It sets two conditions to avoid packet-interleaving:

1. If the packet being considered is single-flit, then it can take the bypass as long as there is space in the destination VC.
2. If the packet is multi-flit, then it can take the bypass only if the buffer to bypass is empty (similarly to EBB).

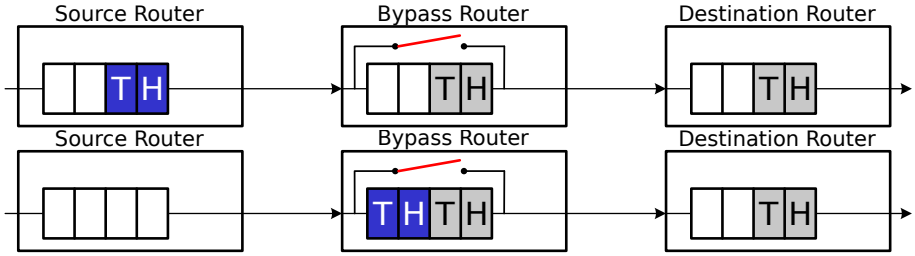
Figure 4.5 shows multiple examples of NEBB-WH. These examples are a representation of the most common situations in terms of buffer occupation.



(a) Multi-flit packet, partial room in destination VC.



(b) Single-flit packet example.



(c) Multi-flit packet, room for the whole packet in destination VC.

Figure 4.5: Examples of NEBB-WH.

The first example (Figure 4.5a) shows the case of a multi-flit packet trying to take the bypass in the *bypass router* when its buffer is empty. The second condition applies in this situation, so that the first two flits of the packet take the bypass, and the last flit is buffered in the *bypass router*. Despite the packet being fragmented in two routers, the second part is at the front of the buffer so it cannot lose the routing information from the VC control registers.

The second example (Figure 4.5b) depicts a single flit-packet, and the *bypass* and *destination* routers have their buffers almost full with just one slot left. The first condition applies in this case, so the packet takes the bypass because it has only one flit.

In the third example (Figure 4.5c), there is a two-flit packet, and the *bypass* and *destination* buffers have two slots left. In this case, the packet cannot take the bypass because the packet has two-flits and the *bypass buffer* is not empty.

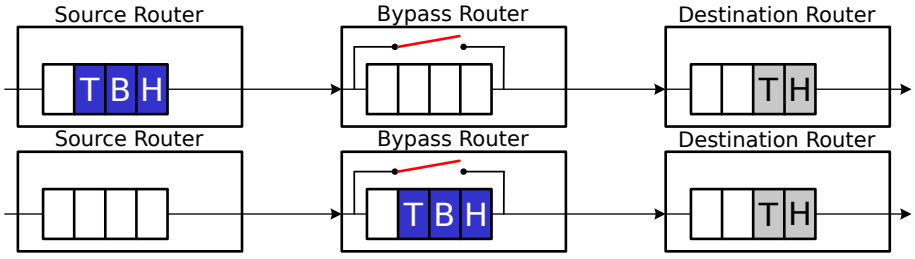
Summarizing, NEBB-WH is an incremental version of EBB that allows the bypass of single-flit packets independently of the occupation of the buffer to bypass. The implementation cost is negligible and only requires an additional multiplexer controlled by the packet size so that single-flit packets ignore the occupation of the *bypass buffer* (following the terminology used in the figures).

4.2.2) NEBB-VCT

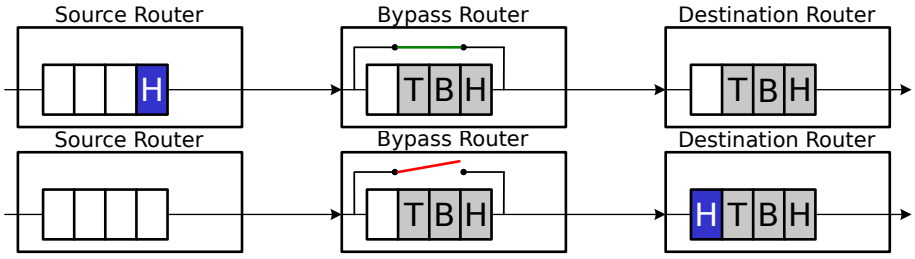
NEBB-VCT is based on Virtual Cut-Through, so the switch and the bypass acquisition are carried out on a packet by packet basis. The conditions that LAs must meet in the *bypass router* are the following:

1. The bypass path is not already granted to another packet.
2. The destination VC has enough space for the whole packet.

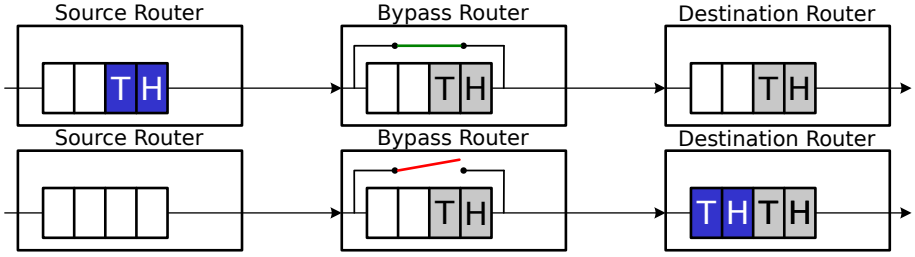
Figure 4.6 depicts the same examples used in Section 4.2.1, but in this case when applying NEBB-VCT.



(a) Multi-flit packet, partial room in destination VC.



(b) Single-flit packet example.



(c) Multi-flit packet, room for the whole packet in destination VC.

Figure 4.6: Examples of NEBB-VCT.

In the first example (Figure 4.6a), the blue packet has three flits while the *destination buffer* only has two free slots. Therefore, following the second condition, the packet cannot take the bypass in the *bypass router*.

In the second case (Figure 4.6b), the single-flit packet can take the bypass like in NEBB-WH, because there is one slot left in the *destination router*.

In the third example (Figure 4.6c), the blue packet has two flits and the *destination buffer* has two slots. Thus, the packet ignores the occupation of the *bypass buffer*, taking the bypass. According to VCT, to forward the packet from the *source* to the *bypass router*, the *bypass buffer* has to have at least two free slots.

NEBB-VCT can be implemented in networks with both WH and VCT in the standard pipeline. In VCT networks, the implementation is straightforward as the arbitration is carried out packet-by-packet. In WH networks, the router locks the bypass path when the head LA of a packet wins LA-Arb and releases the lock when the corresponding flit performs ST.

Figure 4.7 is an example that shows NEBB-VCT with WH in the standard pipeline. The initial state shows that there is only space for the head flit of the blue packet in the *bypass router*, but there is room for both flits in the *destination router*. In this situation, the packet can use the bypass. The bypass has to be locked until the tail blue flit traverses the *bypass router*, ignoring any other request to the same output port from LAs or local flits.

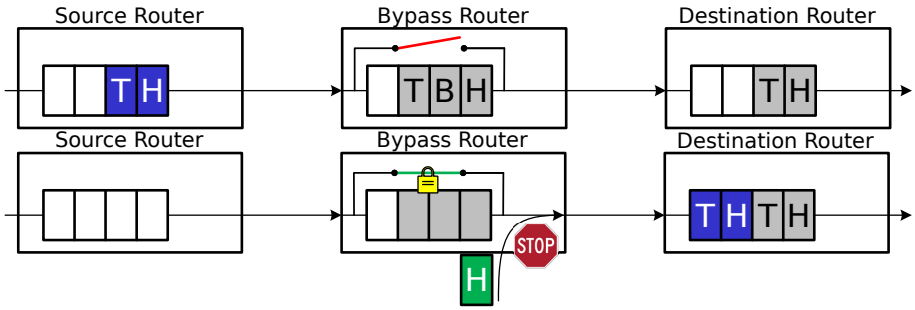


Figure 4.7: Bypass path lock or hold for the packet.

The implementation of NEBB-VCT is simple, specially when using VCT. It requires an extra lock register per input unit to hold the bypass path for the whole packet. This lock register stores the VC ID when an LA wins LA-Arb and frees it when the tail of the packet traverses the bypass. VCT already holds the switch for the packet so no additional logic is required. In LA-Arb, before requesting the bypass and the switch, the router checks the lock register status. Note that with VCT there are only LAs for head flits. In the case of applying NEBB-VCT over WH, the router has to check if the VC assigned to the LA of body flits matches the VC stored in the corresponding lock register. If there is a match, it means that the packet is holding the bypass and the switch. The router also needs a lock register per output port at LA-Arb to hold the switch for the input VC. This is to avoid the reallocation of the output port to a local flit located in the same input unit but in another VC.

4.2.3) NEBB-Hybrid

NEBB-Hybrid combines NEBB-WH and NEBB-VCT to maximize the bypass utilization. It uses the conditions of NEBB-WH or NEBB-VCT, depending on the occupation of the

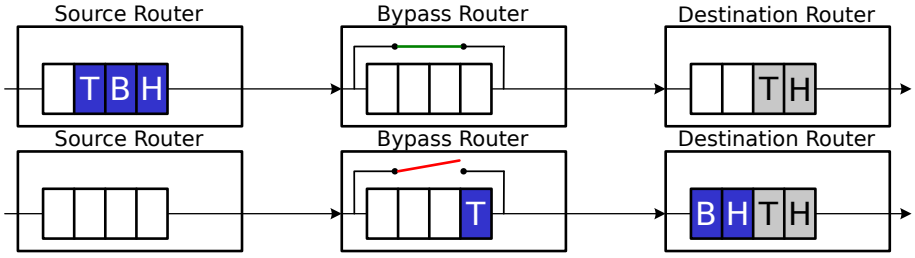
bypass and *destination* buffers. To combine both mechanisms, NEBB-Hybrid uses WH in the standard pipeline. Next, are listed the conditions in which each mechanism is used to take the bypass:

1. (NEBB-WH or NEBB-VCT) If the packet has one flit and there is a destination VC available with at least one free slot.

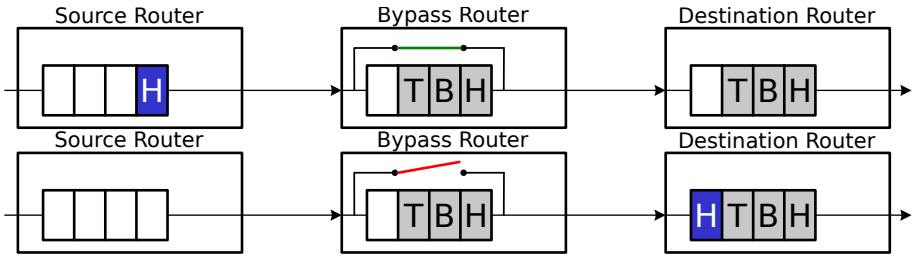
2. (NEBB-WH) If the packet has more than one flit, the *bypass buffer* is empty, the bypass path is not locked for another packet, and there is a *destination buffer* available with at least one free slot.

3. (NEBB-VCT) If the *bypass buffer* is not empty and there is a *destination buffer* available with enough space for the whole packet. In this case the bypass is locked for the packet as mentioned in Section 4.2.2.

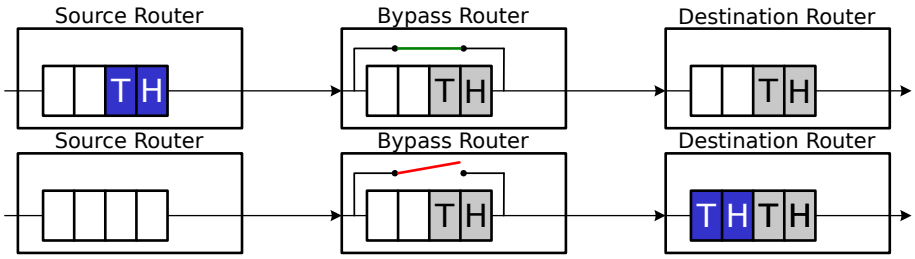
Figure 4.8 depicts the same example scenarios used previously for NEBB-WH and NEBB-Hybrid, but applied to NEBB-Hybrid.



(a) Multi-flit packet, partial room in destination VC.



(b) Single-flit packet example.



(c) Multi-flit packet, room for the whole packet in destination VC.

Figure 4.8: Examples of NEBB-Hybrid.

In the first scenario (Figure 4.8a), the three-flit packet takes the bypass in the *bypass router* as the situation meets the second condition. In the second scenario (Figure 4.8b), the packet has one flit and there is enough room at the *destination buffer*, so the packet takes the bypass following the first condition. The third example (Figure 4.8c) shows a

two-flit packet and the *bypass* and *destination* buffers have two empty slots. The packet could progress to the *bypass router* with just one empty slot, but in this case, it also takes the bypass because the *destination buffer* has room for the whole packet as stated in the third condition. The bypass path has to be locked for the packet.

The implementation of NEBB-Hybrid is slightly more complex than the other two alternatives because of the number of conditions to check. The decisions depend on four control values, instead of three like in NEBB-WH and NEBB-VCT: the packet size, the state of the *bypass* and *destination* buffers, and the lock register of the bypass and switch when using NEBB-VCT. To hold the switch like NEBB-VCT over WH, LA-Arb needs a lock register per output port to hold the switch for the whole packet when it is granted following the third condition. Besides, NEBB does not require VCs to achieve a high bypass utilization and throughput (see Section 4.5), so there is a wide margin to simplify the VC logic if required to meet timing requirements.

4.2.4) NEBB summary

Table 4.1 summarizes the buffer conditions under which each of the bypass mechanisms can be applied. It employs the same terminology as the previous figures: *bypass buffer* refers to the router buffer in which the packet or flit takes the bypass, and *destination buffer* is the one that receives it after taking the bypass.

Table 4.1: Bypass buffer conditions for different mechanisms. VCT and NEBB-VCT require buffers of size, at least, equal to the maximum packet size. NEBB-Hybrid can work with buffers even of just 1 flit, but packets with a greater size than the buffer can not take the bypass following NEBB-VCT.

Bypass buffer occupation	Empty			Not empty		
Packet size	Single-flit	Multi-flit		Single-flit	Multi-flit	
Destination buffer space	$\geq$ 1 flit	$\geq$ packet	$<$ packet	$\geq$ 1 flit	$\geq$ packet	$<$ packet
VCT (EVCF & EBB)	✓	✓				
WH (EVCF & EBB)	✓	✓	✓			
NEBB-WH	✓	✓	✓	✓		
NEBB-VCT	✓	✓		✓	✓	
NEBB-Hybrid	✓	✓	✓	✓	✓	

The first two rows are traditional implementations of bypass routers. *VCT* refers to a version with VCT in the standard pipeline. *VCT* only allows taking the bypass when the *bypass buffer* is empty, and the *destination buffer* is empty (EVCF) or has space for the whole packet (EBB). *WH* refers to a version with WH in the standard pipeline. To enable the bypass path, it requires that the *bypass buffer* is empty, and the *destination buffer* is empty (EVCF) or has space for at least one flit (EBB).

Regarding NEBB, NEBB-WH allows taking the bypass when the *bypass buffer* is empty or the packet size is one flit. NEBB-VCT allows using the bypass when the *destination buffer* has room for the whole packet. NEBB-Hybrid is the mechanism that covers the largest number of cases. It allows bypassing the *bypass buffer* when it is empty independently of the available space in the *destination buffer*, and when it is not empty and there is space for the whole packet in *destination buffer*.

4.3 NEBB in tori with bubble-based flow control

This section presents a study of the implementation of NEBB in tori with bubble-based flow control. To the best of our knowledge, traditional single-hop bypass routers have only been evaluated in mesh topologies. Tori are interesting because they present lower average distances and larger bisection bandwidths than meshes, and they are also node symmetric as discussed in Section 1.4.1. As mentioned in Section 1.4.2, Bubble-based flow controls are efficient deadlock avoidance mechanisms for tori that do not require VCs.

Implementing bubble-based flow controls in NEBB bypass NoCs is not straightforward. We focus on Flit Bubble Flow Control (FBFC) [Ma2015] as it works in a flit-by-flit basis like NEBB-Hybrid. In particular, we focus on FBFC-L (Localized). It requires a destination buffer with enough room for the whole packet plus one flit (the bubble) when injecting it or forwarding it to a different dimension. This guarantees that at least one flit slot is always empty in every ring of the torus.

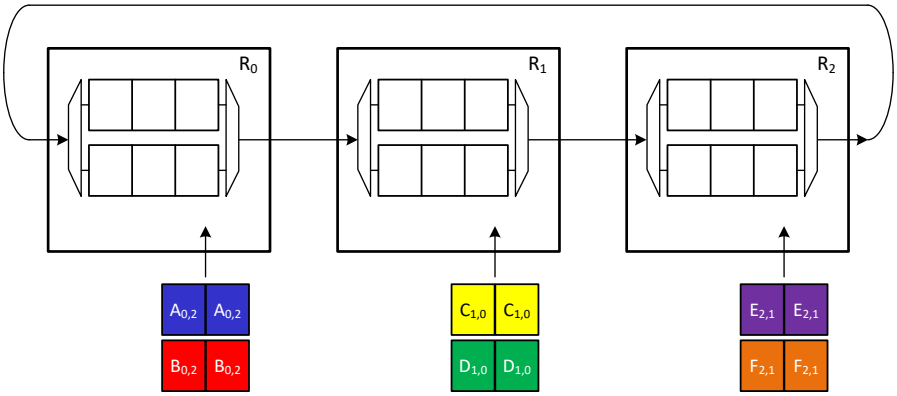
As mentioned in Section 2.1.3.D, the switch allocator in single-hop bypass often employs a two-stage implementation form by SA-I and SA-O [Kumar2008; Krishna2010]. Simple round-robin (RR) arbiters are often used. RR input arbiters cycle through all available VCs, selecting one at a time consecutively. If one of the output VCs is not available (for example, there are no credits in the destination VC) it cannot win SA-O, wasting one cycle. Also, such implementation inherently multiplexes packet flits, generating packet holes in WH, i.e., flits of the same packet are not forwarded in consecutive cycles. However, the benefit of such implementation is that it simplifies the router design, since output availability does not need to be propagated to the inputs.

Holes are undesirable when VCT is used in the bypass of *NEBB-Hybrid* as it locks the bypass for the whole packet, blocking other packets. To minimize them, we use a variable priority input arbiter to give priority to body flits, selecting the same VC until the packet tail flit is forwarded, similar to VCT. However, this might introduce performance and deadlock issues when WH and VCT are combined. First, a packet may be forwarded using WH without space at the destination buffer for the complete packet, introducing delays until the buffer becomes available. Second, a direct implementation introduces a dependence between input VCs, which might generate a deadlock. This may happen when a fragmented packet tries to access SA-O, but there is another packet blocked in SA-O waiting for credits. If at the same time there is a packet in the downstream router in a similar situation, it may prevent releasing space for the blocked packet in the upstream router. The deadlock is produced when this situation occurs in all the routers of a ring of the tori.

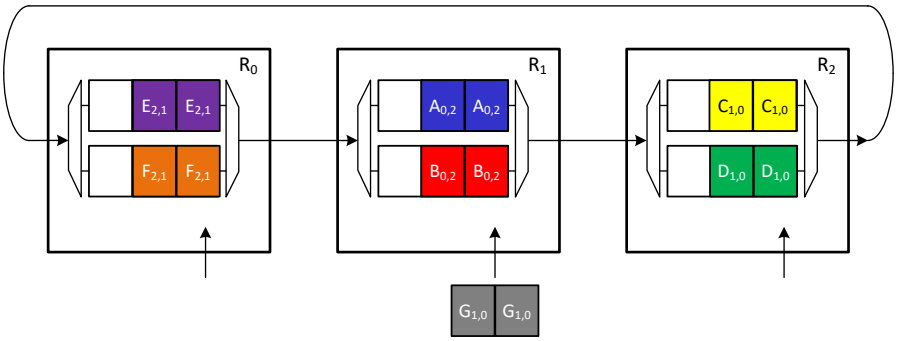
Figure 4.9 illustrates an example in of the potential deadlock when using NEBB in a torus. The configuration presented has three routers forming a ring (1D-Torus), unidirectional in this case for simplicity. Every packet in the example has two flits and requires two hops to reach the destination router. Each packet is represented with a different color and is identified by a letter, its source and destination: $X_{src,dst}$, where X is the identifier, *src* the source and *dst* the destination.

Figure 4.9a depicts the initial state, where two packets are injected in each router: A and B in R_0; C and D in R_1; and E and F in R_2. The FBFC-L condition is satisfied in every case because the destination VCs have room for three flits (two flits plus the bubble).

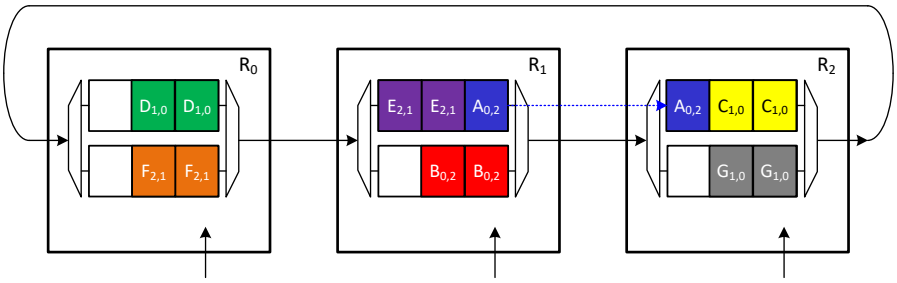
Figure 4.9b represents the state of the routers after injecting the packets into the ring.



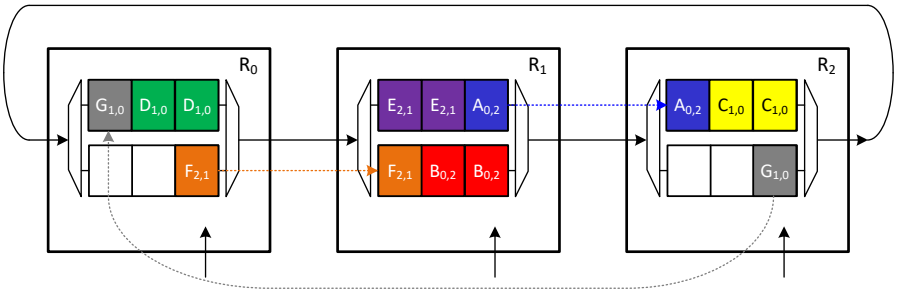
(a) Initially each router injects two packets.



(b) R_1 injects packet G after sending packet D to R_0.



(c) SA-I winners: E in R_0, A in R_1 and D in R_2.



(d) F wins in R_0 and F in R_2 producing a deadlock.

Figure 4.9: Switch allocator deadlock in Torus (ring) using FBFC. Packets have two flits. They are represented with different colors and letters ($X_{src,dst}$, where X the packet identifier, src the source router and dst the destination router).

In this situation, there is a conflict between VCs in SA-I of each router. We assume that the winners are: E in R_0, A in R_1 and D in R_2; and they choose the first VC of R_1, R_2 and R_0, respectively. Also, R_1 will inject a new packet, G, after D frees the second VC of R_2.

The new state of the routers is depicted in Figure 4.9c. In this case, F wins in R_0 and G in R_2. G chooses the first VC of R_0 and F the second VC of R_1.

Finally, Figure 4.9d shows the deadlock. In R_0, F locks SA-I but cannot progress because the second VC of R_1 is full. The same occurs with A in R_1. B cannot advance to R_2 despite there is space the second VC because SA-I is locked by A. In R_2, the lack of space in the first VC of R_0 blocks G, and G blocks C the access to SA-I.

The solution is simple, instead of holding SA-I for body flits, we give priority to body flits temporally to minimize packet holes as before. However, a flit loses the priority when it does not advance to SA-O. Thus, in the next cycle the RR arbiter can choose the next candidate in SA-I breaking the input VC dependency.

4.4 Implementation details

This section addresses implementation considerations when designing single-hop bypass routers.

4.4.1) Credit management using shared buffers

In Section 2.1.3.C we mention that the original single-hop bypass NoC proposals use VC Selection (VS) instead of VC Allocation (VA). However, we use VA based on credits to simplify the comparison with standard routers, since in fact, our simulation models rely on credits. This requires some adjustments in the credit management when using shared buffers to tolerate the round-trip delay of credits.

Shared buffers [Tamir1992; Nicopoulos2006] are very important when implementing EVCF to rise performance without wasting buffer space. The credit accounting with this type of buffer has to consider the flow control mechanism applied in the standard and bypass paths. When a router forwards a packet following VCT, it has to decrease the credit count by the packet size in advance. Otherwise, the forwarding of another packet to the same output port following WH may invade slots initially intended for the first packet. This only affects NEBB-Hybrid and NEBB-VCT over WH. The implementation needs an extra multiplexer in each output unit to select the value to decrement from the credit count, depending on the bypass conditions used. Credit forwarding and reception are the same as in a traditional WH router.

In NEBB-VCT over VCT, the forwarding and control of credits follow a packet-basis management. Credits need an extra field to indicate the packet size that is moving from the buffer so that the upstream router can increase the credit count by this value.

4.4.2) VC Selection in NEBB

Although the proposed router architecture uses VA, NEBB is compatible with VC Selection (VS). As a remainder, VS combines a pool of free VCs together with on/off signaling and a VC Selector to reduce the length of the input path that may determine the critical path.

To use this idea in NEBB, instead of using *free VC* signals, it employs *avail VCs* signals to indicate if there is an available VC, i.e., not in use by another packet, with enough slots to store a whole packet. In the case of NEBB-VCT and NEBB-Hybrid, they require one *avail VC* signal per packet size to perform VCT forwarding when bypassing a whole packet.

4.4.3) Bypass in torus using Flit Bubble Flow Control and shared buffers

Section 4.3 addresses the issue of combining FBFC and single-hop bypass due to the separation of SA in two stages. The example used in that section analyzes the problem with private buffers. However, implementing shared buffers introduces another deadlock source. The bubble condition is checked when the head of the packet is forwarded.

Consider a shared buffer with space for exactly two packets. Two packets are forwarded, interleaved in the same physical channel, and both obey the condition when their head is sent. However, when they are fully received, the bubble disappears.

This issue is solved by decrementing the whole packet size when the head of a packet changes the traveling direction or dimension. This is similar to the previous solution presented in 4.4.1 when combining WH and VCT. Therefore, when using *NEBB-Hybrid* with shared buffers in tori, credits are decremented by the whole packet in two cases: when forwarding multi-flit packets using VCT and for injection and dimension change using any flow control mechanism.

4.5 Evaluation

This section evaluates the three variants of NEBB, with special emphasis on NEBB-Hybrid. Section 4.5.1 describes the experimental setup, Section 4.5.2 evaluates the mechanisms with synthetic traffic, and Section 4.5.3 evaluates them with real traffic from full system simulations.

4.5.1) Experimental setup

To evaluate NEBB flow control mechanisms we use the BookSim version of BST presented in Chapter 3. We also use DSENT [Sun2012] to estimate the dynamic power consumption. We model 64-node networks, arranged as 8×8 mesh or torus topologies. The router employs a two-stage SA similar to [Kumar2008; Krishna2010] to balance pipeline stages. Priority is given to LAs over buffered flits. In the standard router path, priority is given to body flits as mentioned in Section 4.3. Simulation parameters are shown in Tables 4.2 and 4.3, unless otherwise noted in the text.

Six bypass flow control mechanisms are evaluated. EVCF, including LA-Arb like the router architecture described in Section 2.1.1. *Baseline w/o Arb* implements EBB but does not include LA-Arb, ignoring LAs when there are conflicts with other LAs or local flits, resembling the architecture of [Kumar2007]. *Baseline* implements EBB with LA-Arb. Additionally, we evaluate the three variants of NEBB introduced in Section 4.2: *NEBB-WH*, *NEBB-VCT* (employing VCT in the non-bypass pipeline) and *NEBB-Hybrid*.

Table 4.2: Default simulation parameters.

General parameters	
Topology	8×8 mesh or torus
Link latency	1 cycle
Router architecture	2/4-stage bypass router (accounting LT)
Router size	5 ports
Packet size	1 and 5 flits
Buffer implementation	Shared (DAMQ, [Tamir1992])
Buffer size	12 flits (1 private flit per VC)
Routing	DOR
SA input arbiters	8 Round Robin arbiters, #VCs:1
SA output arbiters	8 Matrix arbiters, 8:1
LA arbiters	8 Matrix arbiters, 8:1
VA policy	Highest number of credits
Channel width	128 bits

Experiments use synthetic and real traffic. For synthetic traffic we use both single-flit packets or bimodal traffic. Bimodal traffic resembles a coherence protocol using packets of one (control) and five (data) flits. A single-flit packet ratio of 80% is used [Ma2012; Ma2015]. The default traffic pattern injected is random-uniform, but we also evaluate bit-complement, tornado, transpose and hotspots in the corners of the layout (nodes 0, 7, 56 and 63). We focus on the most relevant performance metrics, which are average packet latency, dynamic power, and percentage of buffered flits. The latter divides the number of times a flit is buffered by the total number of times it is forwarded, averaged for all flits.

We use gem5 [Binkert2011] to generate the real traffic from FS simulations. We simulate a tiled system with 64 Out-Of-Order (O3) ARM CPUs with private L1s and a shared L2 distributed among the tiles. We use Virtual Networks (VN) to avoid protocol-deadlock. Each VN has one VC, except for the case of EVCF in the torus, which requires 2 VCs to avoid routing-deadlock using Dateline [Dally2003]. We run the PARSEC suite [Bienia2008] with the *simlarge* input set for every benchmark to have enough workload for the 64 cores. Because of the excessive simulation time of the whole benchmarks, we collect the results after executing the first 100 million cycles.

To estimate dynamic power with DSENT we have implemented an approximated model of a bypass router micro-architecture based on the default four-stage router model of DSENT. The first part of the approximation consists in multiplying the dynamic power of the buffers and allocators of the default router model by the ratio of buffered flits over all the received flits per router reported by BookSim. The second part approximates the consumption of the switch allocators and LA-Arb. LA-Arb is very similar to SA-O. Therefore, the power consumption of LA-Arb equals the power of SA-O. In this case we do not apply any correction factor because these arbiters are used for every LA, and one LA is received for each flit. We omit the consumption of extra control logic (such as the checks of the packet size, the occupancy of the buffer to bypass, etc) as it is negligible compared to the consumption of the buffers, crossbar or arbiters, and can be reduced by using VS plus *avail_vc* signaling.

Table 4.3: Parameters for each simulation type.

Synthetic traffic parameters	
Num. VCs	2
Simulation cycles	50.000 cycles
Full-system parameters	
CPUs	64x O3 ARM @ 2 GHz (DerivO3CPU)
L1 caches	32KB (L1-I) and 64KB (L1-D) per core
L2 caches	64x 256KB shared banks
Memory controllers	16 (first and last rows)
Num. VCs	3 (1 per virtual network)
NoC frequency	2 GHz
Simulation cycles	10^8 cycles
DSENT parameters	
Frequency	2 GHz
Technology	Tri-Gate 11nm LVT process

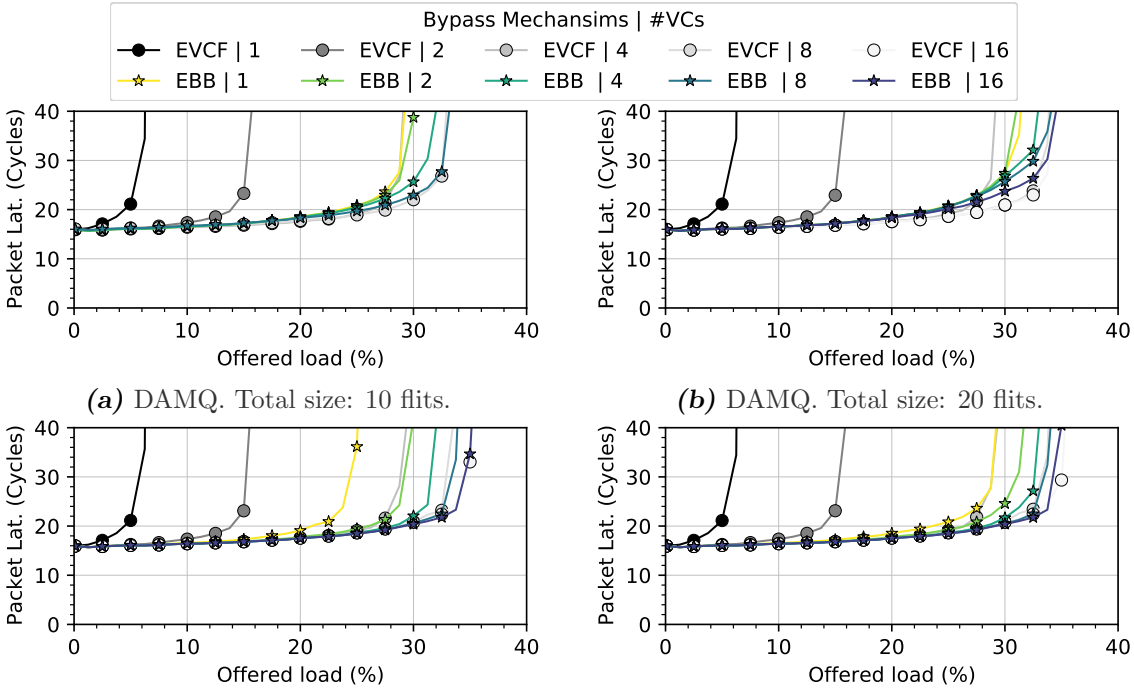
Packets are able to take the bypass path in their injection routers. This is possible by creating and sending LAs from nodes to injection routers before forwarding the corresponding flits. BookSim implements this by adding one extra cycle to flit injection links so that LAs arrive one cycle before flits. Link latency is one cycle except for injection links, which is 2 cycles for flits and 1 cycle for LAs. The rest of the links of the network have a latency of one cycle.

4.5.2) Synthetic traffic analysis

This evaluation focuses on comparing the different single-hop router bypass mechanisms with synthetic traffic. It starts comparing EVCF, which is the mechanism used in the most relevant works about bypass routers [Kumar2007; Kumar2008; Krishna2010; Krishna2013; Kwon2017] and the baseline for the rest of the section, EBB, which is a more efficient mechanism (see Section 4.5.2.A). Next, we evaluate the performance and efficiency of NEBB using single-flit packets and bimodal traffic in meshes and tori. The last part carries out two sensitivity analyses, to the buffer configuration and the bypass priority policy.

4.5.2.A) Empty VC Forwarding vs Empty Buffer Bypass

Figure 4.10 compares the two bypass flow control mechanisms described in Section 4.1.2: *Empty VC Forwarding* (EVCF) and *Empty Buffer Bypass* (EBB). As a reminder, the difference between them is that *EVCF* only forwards packets when destination buffers are empty. Both mechanisms have LA arbiters because it is the configuration with the best results, as shown in Section 4.5.2.B. Bimodal traffic formed of 1-flit and 5-flit packets is used in all cases.



(a) DAMQ. Total size: 10 flits.

(b) DAMQ. Total size: 20 flits.

(c) Private VC. VC size: 5 flits.

(d) Private VC. VC size: 10 flits.

Figure 4.10: Packet latency in an 8×8 mesh with *Empty VC Forwarding* (EVCF) and *Empty Buffer Bypass* (EBB), using bimodal traffic.

Figures 4.10a[1] and 4.10b employ shared buffers (DAMQ) with space for 10 and 20 flits, respectively. Clearly, *EBB* is better with 1, 2 and 4 VCs. It is particularly notorious the low throughput of *EVCF* with 1 and 2 VCs. *EBB* with a single VC is even better than *EVCF* with 4 VCs. With 8 or more VCs, the improvement of *EBB* runs out, providing similar results in both configurations. Figures 4.10c and 4.10d have a private buffer per VC with room for 5 and 10 flits, respectively. Again, *EBB* is better for 1 to 4 VCs, and similar with 8 and 16 VCs.

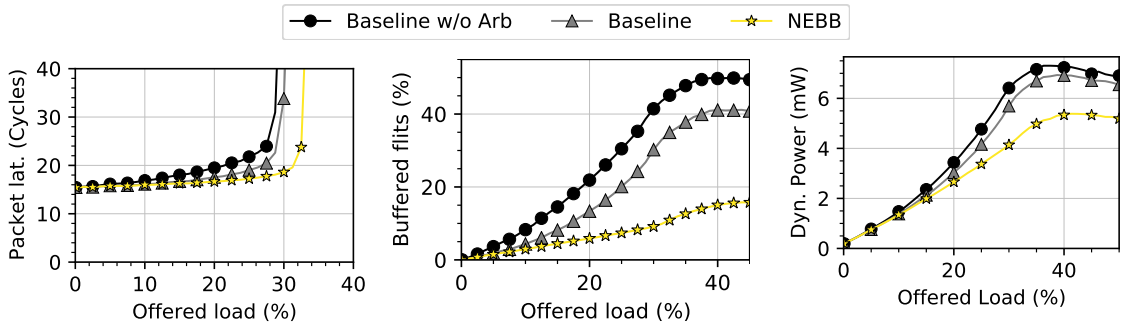
EBB is clearly better than *EVCF* when the number of VCs and their buffer size are low. Reducing the number of VCs and the buffer space is crucial for reducing power consumption and area of NoC designs. For this reason and to be conservative when comparing NEBB, we use EBB as our baseline in the next sections.

4.5.2.B) NEBB using Single-Flit Packets

Figure 4.11 compares packet latency, buffered flits and dynamic power of each bypass mechanism; lower values are better in all the cases. These first evaluations use single-flit traffic, therefore, all *NEBB* variants are equivalent. Due to the small packet size, the configurations with shared buffers have only 6 slots.

The percentage of buffered flits in 4.11b grows with the network load. *Baseline w/o Arb* stores packets when the buffers are non-empty, or there are LA conflicts (it discards LAs in such case) like in [Kumar2007]. The *Baseline* model is similar, but one LA proceeds in case of conflict, reducing the use of buffers. In *NEBB*, buffers are only used when conflicts occur, and not because of non-empty buffers, minimizing the buffer utilization.

[1]Figure 4.10a does not have results with 16 VCs because each VC needs a private slot and the DAMQ only has 10 flits.

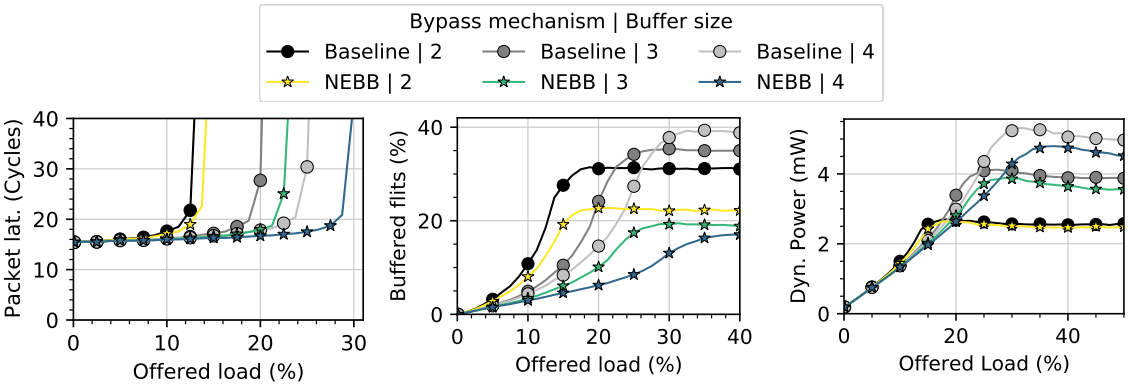


(a) Packet latency. ***(b)*** Buffered flits. ***(c)*** Router dynamic power.

Figure 4.11: 8 × 8 mesh performance and efficiency with single-flit random-uniform traffic, a DAMQ of 6 flits and 2 VCs.

This translates into latency and power savings, particularly at intermediate and high loads. For example, at 28% load, *NEBB* reduces *Baseline w/o Arb* latency by 24.5% and buffered flits by 75.5%. From these values, 14.6% and 31.2% respectively come from the LA arbiter, as observed in *Baseline* results. Regarding dynamic power, *NEBB* saves 23.6% over *Baseline w/o Arb* at 28% load.

Figure 4.12 shows the latency, throughput and buffered flits of *Baseline* and *NEBB* using routers with minimal buffering. These configurations do not have VCs (i.e., equivalent to 1 VC), and the input buffers have 2, 3, or 4 slots. *NEBB* reduces the number of buffered flits under these conditions at 28% load by 27.3%, 46.5% and 69.9% for 2, 3, and 4 slots respectively. Increasing the bypass utilization also reduces HoLB as fewer packets use the buffers and packets may take the bypass if an input VC is blocked. This has a positive effect in throughput. Thus, *NEBB* achieves 7.9%, 12.3%, 17.7% more throughput than *Baseline* for 2, 3, and 4 slots, respectively.

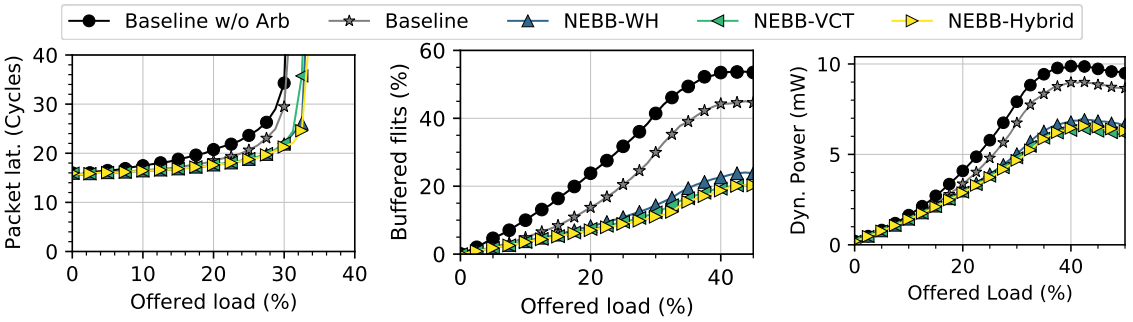


(a) Average packet latency. ***(b)*** Buffered flits. ***(c)*** Router dynamic power.

Figure 4.12: Performance of bypass routers in an 8 × 8 mesh with single-flit random-uniform traffic, and minimal buffering without VCs.

4.5.2.C) NEBB *Flow Control and* Hybrid

Figure 4.13 compares the *NEBB* alternatives using bimodal traffic. The three *NEBB* variants outperform the baselines, and *NEBB-Hybrid* presents the best results since it maximizes the cases in which the bypass is used.

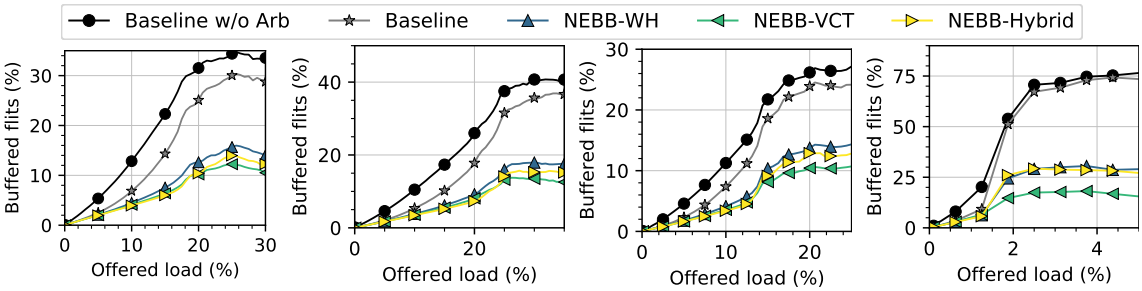


(a) Packet latency. **(b)** Buffered flits. **(c)** Router dynamic power.

Figure 4.13: 8×8 mesh performance and efficiency with bimodal uniform-random traffic, a DAMQ of 12 flits and 2 VCs.

Both *NEBB-WH* and *NEBB-VCT* present similar results. VCT has a slightly lower throughput, which translates into slightly lower power results after saturation as the volume of traffic is lower. *NEBB-Hybrid* has the best results in latency, buffered flits and dynamic power. Before saturation, at a load around 27%, the reductions over *Baseline* are 14.5%, 59.9%, and 34.3%, respectively.

Figure 4.14 depicts the buffer utilization for different traffic patterns. The results are similar to the previous ones with random-uniform traffic, with *NEBB* mechanisms improving the utilization of the bypass. *NEBB-VCT* has the lowest buffer utilization among the *NEBB* mechanisms. However, the reason is that it has slightly less throughput than the rest, stopping the growth of the buffer utilization earlier.



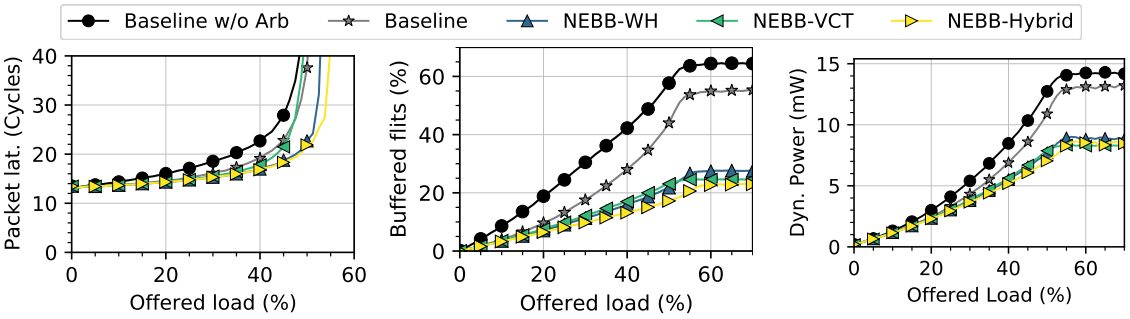
(a) Bit-complement. **(b)** Tornado. **(c)** Transpose. **(d)** Hotspot.

Figure 4.14: Buffered flits in an 8 × 8 mesh for different traffic patterns, using bimodal traffic, a DAMQ of 12 flits and 2 VCs.

4.5.2.D) NEBB *in Torus networks*

Figure 4.15 depicts results of the bypass flow control mechanisms in tori. All the mechanisms utilize FBFC except for *NEBB-VCT*, which relies on Bubble flow control [Carrion1997] since all the allocation is carried out at the packet level.

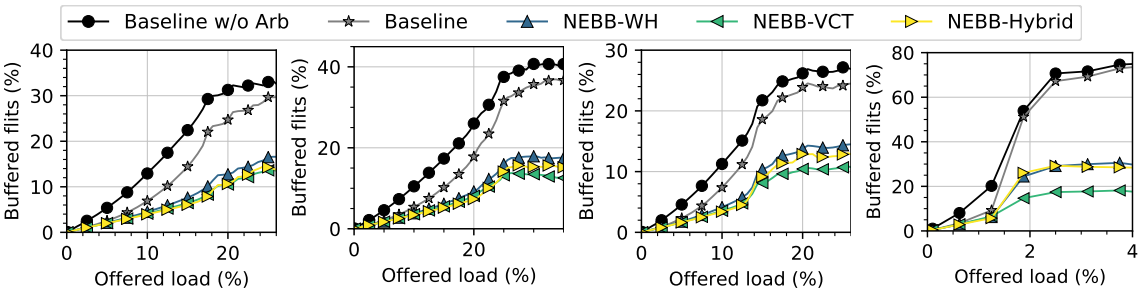
In general, base latency is lower than in the mesh, and throughput almost doubles, proving that FBFC and bypass routers with shared buffers operate correctly together. FBFC presents better results than Bubble flow control, with *NEBB-VCT* presenting similar latency than *Baseline*. Once more, *NEBB-Hybrid* has the best results, improving latency over *Baseline* by 54.1% and dynamic power by 23.8% at 40% load.



(a) Average packet latency. ***(b)*** Buffered flits. ***(c)*** Router dynamic power.

Figure 4.15: Performance of an 8×8 torus with bimodal uniform traffic, a DAMQ of 12 flits and 2 VCs.

Figure 4.16 shows buffer utilization results for various traffic patterns. *NEBB* always writes fewer flits in the buffers than the baselines. Like occurs in the mesh, *NEBB-VCT* has the lowest buffer utilization among the NEBB mechanisms. In this case, it is more appreciable that NEBB-VCT achieves less performance than the others in Figure 4.15a, given that it uses Bubble flow control instead of FBFC.



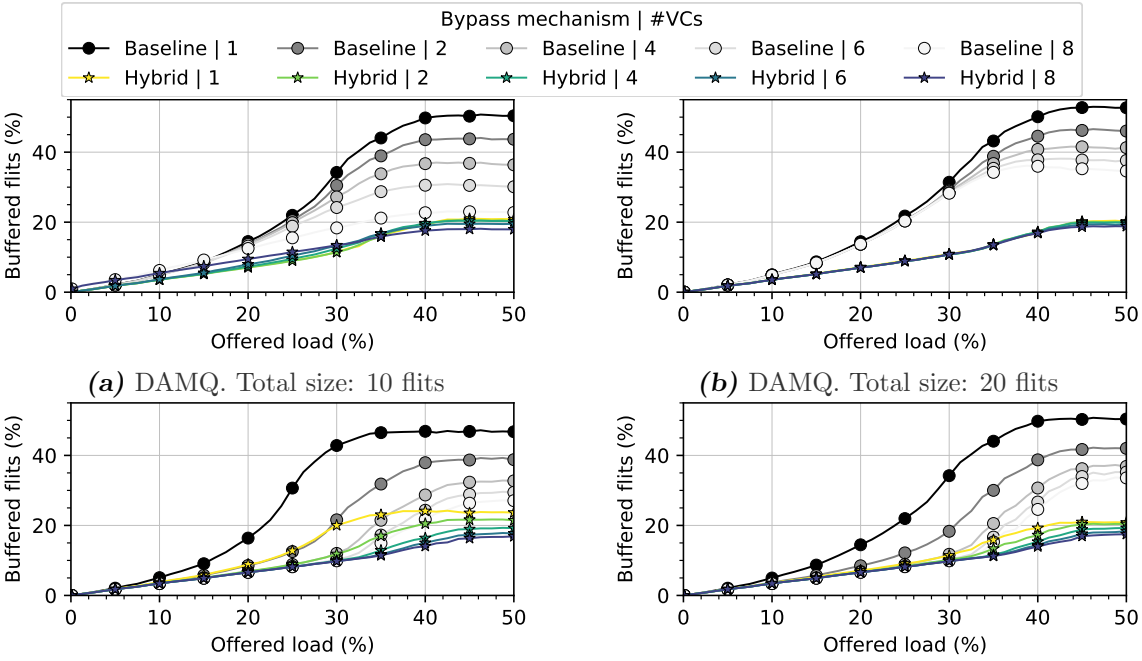
(a) Bit-complement. ***(b)*** Tornado. ***(c)*** Transpose. ***(d)*** Hotspot.

Figure 4.16: Buffered flits in an 8×8 torus for different traffic patterns, using bimodal traffic, a DAMQ of 12 flits and 2 VCs.

4.5.2.E) Sensitivity analysis: buffer depth and number of VCs

Figure 4.17 depicts the buffer utilization of *Baseline* and *NEBB-Hybrid* with different combinations of VCs and buffer sizes. Each curve represents the same configuration with a different number of VCs, either sharing the same buffer space (Figures 4.17a and 4.17b) or using private buffers per VC (Figures 4.17c and 4.17d). With shared buffers, *Hybrid* clearly outperforms *Baseline*, particularly when the shared buffer size is not very small. With 20 flits per port, there is not a single *Baseline* configuration that matches the result of *Hybrid*. Additionally, the number of VCs used in *Hybrid* has a small impact on buffered flits.

In the private buffers evaluations of Figures 4.17c and 4.17d, the total amount of storage increases with the VC count. If buffers are small (Figure 4.17c, buffer per VC equals the maximum packet size of 5 flits), *Hybrid* is better than *Baseline* for the same number of VCs, but the improvement is lower than with shared buffers. Indeed, this is the minimum buffer size for *Hybrid* to use VCT. With larger buffers (Figure 4.17d), the effect of using VCT in *Hybrid* grows and it gets approximately the same results of *Baseline* with



(a) DAMQ. Total size: 10 flits

(b) DAMQ. Total size: 20 flits

(c) Private VC. VC size: 5 flits.

(d) Private VC. VC size: 10 flits.

Figure 4.17: Buffer utilization for a mesh with different number of VCs and buffer sizes using bimodal traffic.

half the VCs before saturation, and gets better after this point.

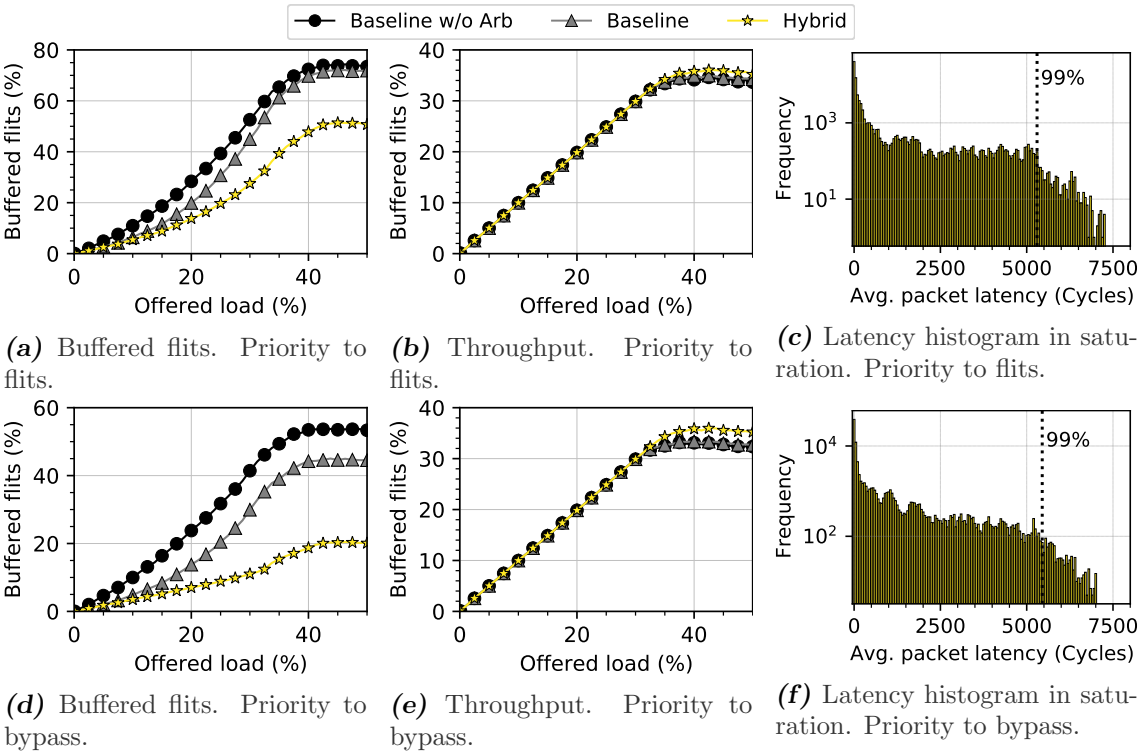
4.5.2.F) Sensitivity analysis: crossbar priority to buffered or bypassed flits

The original bypass proposal [Kumar2007] prioritizes flits in the non-bypass pipeline (second bypass condition mentioned in Section 2.1.3.F), but the opposite priority (to LAs) is used in this thesis. Figure 4.18 compares both alternatives, depicting buffered flits, throughput, and packet latency histograms. On the one hand, giving priority to LAs is positive to decrease the number of buffered flits in all the mechanisms, specially at medium and high load. On the other hand, the maximum throughput decreases slightly.

The packet latency histograms of *NEBB-Hybrid* show that the number of high latency packets slightly increases with priority to LAs. This issue is shared by all mechanisms when more than 1 VC is used. To reduce peak latency, priority may be given to buffered flits after a given number of cycles. The specific threshold used (e.g., 30 cycles in [Kumar2008]) presents a trade-off between the results with priority to LAs or to buffered flits.

4.5.3) Real traffic analysis

To conclude, we evaluate *NEBB-Hybrid* with real traffic through FS simulations in meshes and tori. We compare *NEBB-Hybrid* with *EVCF* and *Baseline*, both with LA arbiters. In this case we include also *EVCF* because the effect of *NEBB* is only appreciable at medium-high loads. All configurations have shared buffers of 12 flits with 3 VCs, 1 VC per VN to avoid protocol-deadlock, except for *Torus-EVCF* that has 6 VCs of 5 slots each. It does not support FBFC-L, so it has to implement Dateline to avoid routing-deadlock, requiring 2 VCs per VN.



(a) Buffered flits. Priority to flits.

(b) Throughput. Priority to flits.

(c) Latency histogram in saturation. Priority to flits.

(d) Buffered flits. Priority to bypass.

(e) Throughput. Priority to bypass.

(f) Latency histogram in saturation. Priority to bypass.

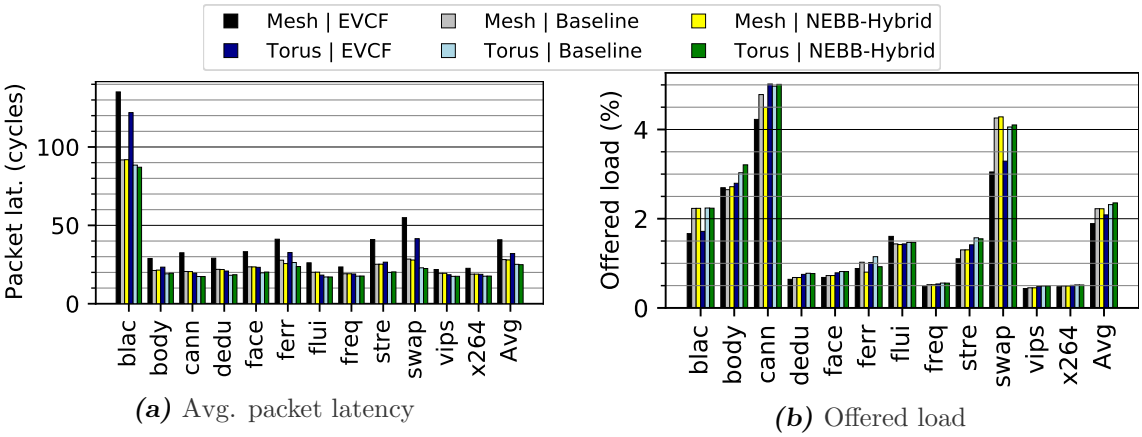
Figure 4.18: *NEBB-Hybrid* buffer utilization, throughput and network latency histograms prioritizing buffered flits or LAs in case of conflicts. Histograms show latency distribution at 50% of offered load.

Figure 4.19 shows packet latency and offered load. First, the offered load in these benchmarks is very low, around 5% in *Canneal*, the benchmark with the highest load. For these reasons, the differences between *Baseline* and *NEBB-Hybrid* are small. Second, the configurations that use a torus reduce latency with respect to their mesh counterpart due to its lower average distance and bandwidth. Third, the *EVCF* configurations have on average 44.9% and 27.3% more latency in the mesh and torus, respectively, compared with *Baseline*. As shown in Section 4.5.2.A, *EVCF* requires more VCs to offer the same performance, complicating the management of VCs and their allocation. In the case of the torus, it is remarkable that the FBFC-L configurations get lower latency with half the VCs and require less overall buffer space due to Dateline's inefficient VCs utilization.

4.6 Conclusions

Single-hop bypass routers reduce the router delay by reducing the pipeline length when possible, specially at low loads. Our proposal, Non-Empty Buffer Bypass, is based on a proper analysis to relax the original bypass conditions stated in [Kumar2007]. We present three variants of *NEBB* mechanism: one version following WH rules; a second one following VCT; and a third one denoted *NEBB-Hybrid* that combines the previous ones to maximize the utilization of the bypass.

We show the effectiveness of *NEBB* in a mesh and a torus using FBFC. Our proposals decrease packet latency by up to 24% and dynamic power up to 23% in comparison with



(a) Avg. packet latency

(b) Offered load

Figure 4.19: Real-traffic performance.

applying the original conditions. Moreover, the experimentation shows that *NEBB-Hybrid* outperforms prior proposals with shared buffers and requires half the VCs to obtain the same result with private buffers, simplifying VC allocation and management.

In summary, *NEBB*, and specially *NEBB-Hybrid*, is a competitive and cost-effective alternative to improve the design and performance of single-hop bypass mechanisms for NoCs.

Chapter 5

SMART++

In this chapter we describe SMART++, which overcomes the limitations of SMART to efficiently use the network resources. Similar to the traditional single-hop bypass mechanisms studied in Chapter 4, SMART is also inefficient due to the unnecessary conservative conditions to use the bypass and the VCs.

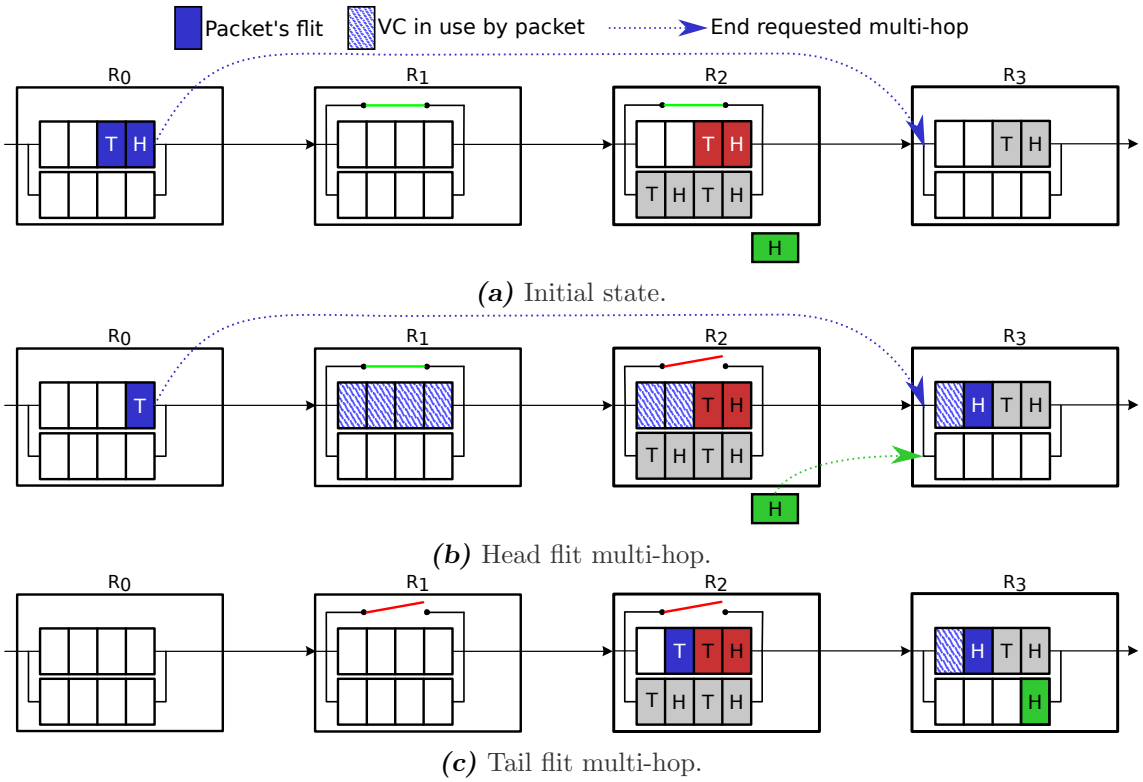
The organization of the chapter is as follows. Section 5.1 describes the packet-interleaving problem that might occur in multi-hop bypass NoCs if a careless implementation is used. Section 5.2 details how this issue is handled in SMART, requiring empty buffers for bypass. This solution implies a significant number of VCs to obtain good performance, but this drastically impacts area, power, and frequency results, resulting in suboptimal implementations. Section 5.3 introduces SMART++, which improves the bypass and buffer utilization to solve the inefficiency problems of SMART. Section 5.4 contains the experimental evaluation of SMART++ with real-hardware simulations from an HDL implementation and functional simulations. Section 5.5 closes the chapter with the most important conclusions.

5.1 Packet-interleaving in multi-hop bypass

This section illustrates the potential data corruption problem that might occur when the required conditions to allow the bypass are carelessly designed. Section 4.1 describes the problem of packet-interleaving in single-hop bypass routers. Multi-hop bypass is not immune to this issue, in fact, is it more susceptible. The larger HPC_{Max} is, the higher the chances of packet-interleaving. SMART avoids this problem with very restrictive conditions as described in Section 5.2. Therefore, this section analyzes the source of the problem to define less severe restrictions in SMART++, which is described in Section 5.3.

Figure 5.1 shows an example of packet-interleaving in a multi-hop bypass network when there are no restrictions to use the bypass and VCs are assigned like in traditional WH networks. There are three relevant packets in the network highlighted in blue, red and green. The blue packet, with two flits at R_0, tries to perform a multi-hop from R_0 to R_3. The red packet is stalled in the first input VC of R_2. One possible cause may be that it has to make a dimension change but there are not free VCs in the next router. The green packet in R_2 of one flit, is in another input unit and tries to advance to R_3. In the initial state (Figure 5.1a) the head of the blue packet wins SA-G in the first three routers (bypass enabled in R_1 and R_2). In this example, the lack of restrictions allows the bypass

of R_2 even though both buffers have packets. The blue head flit stops at the first input VC of R_3 (Figure 5.1b). The first VC of R_1, R_2 and R_3 is assigned to the blue packet, but its tail flit loses SA-G in R_2 against the green packet because of the local priority policy, which is described in Section 2.2.3.B. Note that the green packet can advance towards R_3 because the second VC is available, i.e., it has space for the flit and is not being used. In the last state shown in Figure 5.1c, the blue tail flit stops at the first VC of R_2, resulting in packet-interleaving between the blue and red packets.



(a) Initial state.

(b) Head flit multi-hop.

(c) Tail flit multi-hop.

Figure 5.1: Buffer state of multi-hop bypass packet-interleaving example.

In summary, this example shows the necessity of applying additional restrictions to traditional flow control mechanisms to prevent packet-interleaving.

5.2 SMART: Empty VC Forwarding

This section details the correctness aspects of SMART related to the packet-interleaving problem presented in Section 5.1. The solution relies on a large pool of input VCs per port, which results in inefficient designs in terms of area, power, and cycle time. SMART uses *free_vc* signaling to indicate that there are free VCs, i.e., empty buffers, in the next router. This similar to Empty VC Forwarding (EVCF) described for single-flit bypass in Section 4.1.2.

Figure 5.2 shows the example described in Section 5.1 but applying EVCF. The example has three relevant packets highlighted in blue, green and red (Figure 5.2a). The blue packet attempts to perform a multi-hop to R_3 but it cannot advance further than R_1 because R_2 does not have an empty VC (*free_vc* is off), so it loses SA-G in R_1. The same

occurs for the tail flit (Figure 5.2b). It loses SA-G in R_2 against the green packet because the latter is a local flit (Prio=Local). Note that, even in a different scenario in which the tail flit could win SA-G in R_1 and R_2, it has to stop at R_1 to preserve the flit order within the packet. In Figure 5.2c, the green packet reaches R_3 and the blue packet waits until any of the VCs of R_2 is empty. In the next state (Figure 5.2d), the red and green packets have already moved forward to their corresponding routers (not illustrated in the figure). Thus, the head of the blue packet can move towards R_2, so that it wins SA-G in R_1 and R_2, and reaches R_3 in a multi-hop. In this state, the head flit wins SA-G in R_1 and R_2 to do reach R_3 in a multi-hop. The same process follows the tail flit in Figure 5.2e, stopping at R_3 as shown in the last state (Figure 5.2f).

Besides demonstrating how EVCF fixes packet-interleaving, it also shows the limitations of this mechanism in terms of throughput. Despite having space for the whole blue packet in R_3 in the first multi-hop attempt, this packet has to wait until the buffers have been emptied to complete the multi-hop. This means that SMART needs many VCs to take advantage of the topology bandwidth. Increasing the number of VCs does not only increase the number of buffers in the input units but also it increases the control registers and the complexity of the VC allocator. This has a negative impact on the area, power, and the critical path delay of some pipeline stages, which eventually translates to higher latencies in certain configurations.

The following sections discuss important considerations about the implementation of Virtual Channel Selection (VS) in SMART.

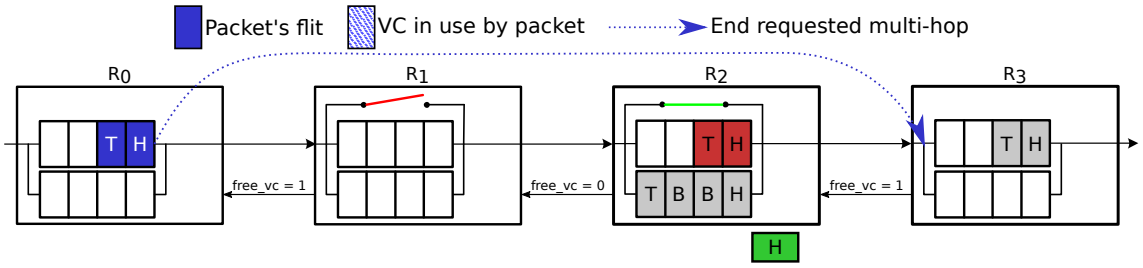
5.2.1) Virtual Channel Selection: flow control and buffer size

SMART implements VCT and Virtual Channel Selection (VS), which is described in Section 2.2.3.A. Using VCT guarantees that buffers have space for body flits once a packet acquires them. Since SMART only forwards packets when downstream routers have an empty VC, the minimum buffer size is the one of the largest packet in the network.

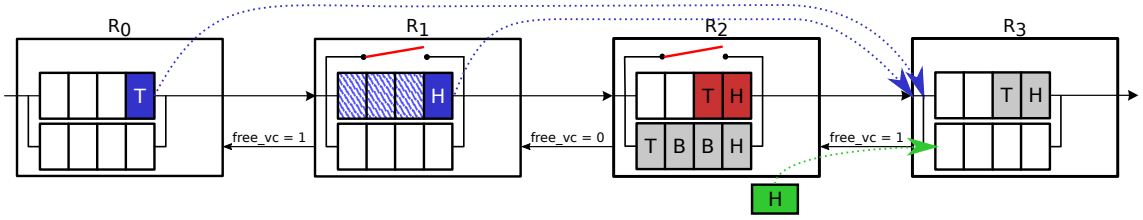
Unlike traditional VCT, SMART does SA in a flit-by-flit basis instead of packet-by-packet. Flit-by-flit arbitration introduces the possibility of body flits stopping prematurely with respect to their headers. This possibility avoids the retransmission of all the flits of a packet in consecutive cycles, which is characteristic of VCT. This may not have impact on the utilization of the links, but it conditions the restrictions to use the bypass. Section 5.3.3 describes how SMART++ implements packet-by-packet arbitration to improve the bypass utilization in combination with NEBB.

5.2.2) Virtual Channel Selection: management of multi-flit packets

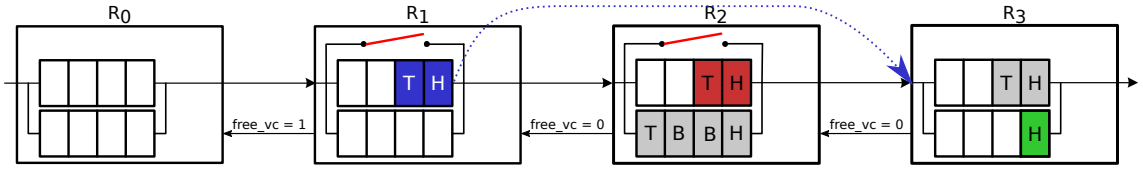
As a consequence of implementing VS with flit-by-flit arbitration, SMART requires a mechanism to identify the VC assigned to a packet when body flits arrive at the router. SMART has a hash table in every input unit to store the VC assigned to each packet. These tables have a size equal to the number of VCs with buffers of more than one slot, i.e., buffers that can hold multi-flit packets. The index field is the injection router ID of the packet. A router assigns this value to the VC entry obtained from VS when the head flit arrives, regardless of the flit taking the bypass or not, and it frees the entry when the tail arrives. To avoid that two or more packets arrive with the same injection router ID, SMART guarantees that all the local flits of a packet leave an output port of a router before another packet can use the same output port. This is done by holding the switch until the traversal of the tail flit, a characteristic of VCT.



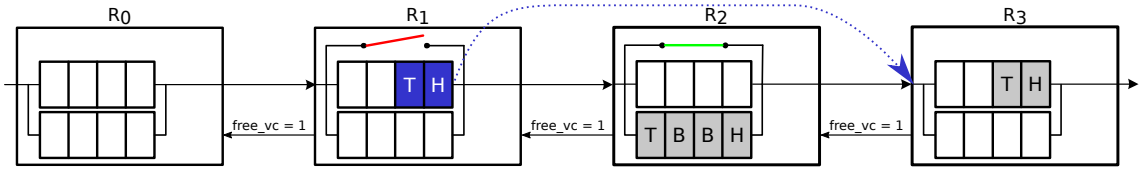
(a) Initial state.



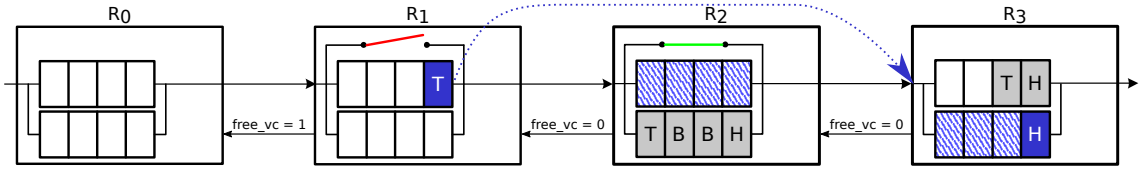
(b) Blue packet's head multi-hop attemp.



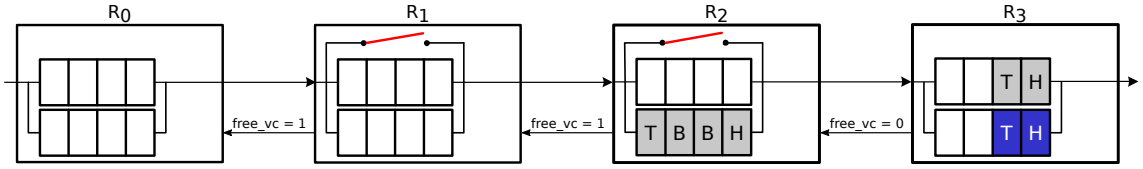
(c) Blue packet's tail multi-hop attemp.



(d) Read and green packets frees VCs of R_2 and R_3.



(e) Blue packet's head second multi-hop.



(f) Blue packet's tail second multi-hop.

Figure 5.2: Buffer state of multi-hop bypass using SMART's empty VC forwarding.

5.3 SMART++

SMART++ is a multi-hop bypass network that combines four mechanisms: SMART, multi-packet buffers, NEBB and packet by packet arbitration. The main advantage of SMART++ over SMART is that it does not require VCs to achieve good performance, reducing the hardware costs of multi-hop bypass drastically. Next we analyze the mechanisms, introducing them incrementally with respect to the base SMART. Section 5.3.4 compares SMART and SMART++, including partial implementations of SMART++. Section 5.3.3 describes a required change in the implementation of the input units to operate efficiently when no using VCs.

5.3.1) Multi-packet buffers

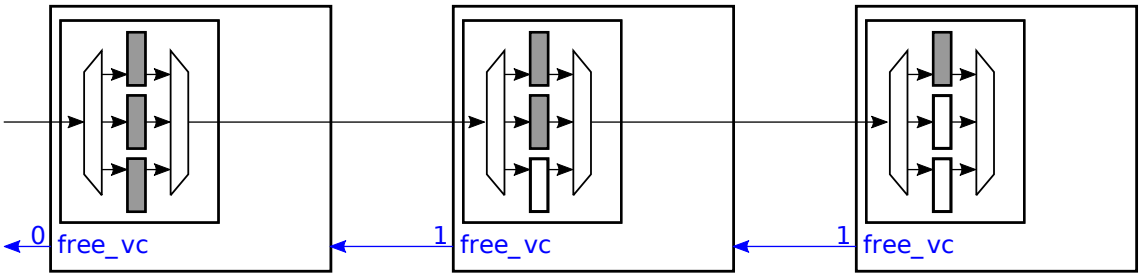
SMART requires buffers sized for the largest packet in the network, since it implements VCT flow control, but it only holds a single packet due to its VC reallocation policy. SMART++ allows holding multiple consecutive packets in router buffers and can exploit buffers larger than a single packet size. Such approach is similar to previous proposals for NoCs [Chen2011; Ma2012; Wang2013; Daya2014; Towles2014]. The implementation is similar to Whole Packet Forwarding (WPF [Ma2012]). WPF implements an aggressive VC reallocation mechanism, which allows reallocating a given VC if it has enough buffer slots to hold the whole packet and the tail of the previous packet has been already sent. According to [Ma2012], *WPF can be viewed as applying packet-based flow control in a wormhole network*. Note that in SMART flit-by-flit arbitration behaves similar to a WH network.

The use of multi-packet buffers allows employing a lower number of VCs with deeper buffers, leading to simpler memory organizations that exchange width (#VCs) by length (deeper FIFOs). Such VC reduction simplifies allocation and reduces overall chip area even though the total storage remains the same. Additionally, combining multiple packets in the same buffer increases its efficiency, particularly with different-size packets (bimodal traffic), which often occurs in NoCs. This is evaluated in Section 5.4.2.A.

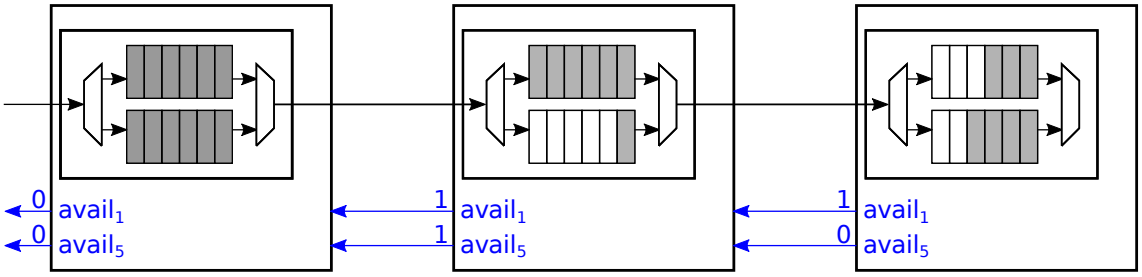
However, applying multi-packet buffers to SMART is not straightforward due to the use of VS. The necessary changes to implement multi-packet buffers with VS are described next.

First, the *free_vc* signaling has to be adapted. The *free_vc* signals of SMART, depicted in Figure 5.3a, only activate when one or more VCs of the input unit are empty. The *free_vc* is a 1-bit signal that only indicates that there is room for a packet so upstream routers do not know which VCs are free. When using multi-packet buffers, *avail_vc* replaces the *free_vc* signals. Instead of a 1-bit signal, *avail_vc* uses one independent 1-bit signal per packet size in the network, as depicted in Figure 5.3b. Each signal indicates the VC availability for each packet size, so the upstream router only attends the signal corresponding to the packet being evaluated. This supports a case with free space for a single-flit packet but not enough space for a multi-flit packet. These signals are denoted $avail_i$, where i indicates the number of flits of the packet size. In the example of Figure 5.3b, there are two *avail_vc* signals: $avail_1$ and $avail_5$ for packets of 1 and 5 flits respectively. If $k < j$ and $avail_j$ is set then $avail_k$ will be also set.

Second, changes in the SSR signals are required. The SSR signals have to include a new field which indicates the packet size. This requires $\lceil log_2 N \rceil$ bits where N is the



(a) *free_vc* signals in SMART



(b) *Availability* signals in SMART++ with packet sizes of 1 and 5 flits.

Figure 5.3: Buffer signaling mechanisms in SMART and SMART++.

number of packet sizes. For $N = 2$, which is very common in CMPs, this implies that only one additional bit is required. This information is needed when computing SA-G, so the router can select the correct *avail_vc* signal for the requested output port.

Third, additional control logic is required to check whether the bypass can be activated or not to avoid packet-interleaving. To grant access to the bypass path, the input unit of the router has to have an empty VC that is not in use by another packet. A buffer can be empty but being used due to packet fragmentation given the flit-by-flit arbitration of SMART, as mentioned in Section 5.2.2. With the use of VS instead of VA, the control of the input VC has to be done in the receptor router. To control if an input VC is being used, it is enough to retire the assigned VC from the pool of available VCs when the head of a packet arrives to the router and reintroduce it when the tail arrives.

5.3.2) NEBB

NEBB can be applied to SMART with multi-packet buffers. It does not make sense to apply NEBB to SMART directly because it can only forward packets when the VC buffer of the next router is empty. NEBB allows the bypass of single-flit packets in SMART++ as they can not be fragmented and therefore they are not a source of packet-interleaving.

For multi-flit packets, the flit-by-flit allocation mechanism in SMART implies that the bypass operation might be interrupted at any cycle, if a higher-priority SSR is received at an intermediate router. When this happens, the remainder of the packet is stored in the intermediate router buffer, and if it is not empty, packets would be interleaved and corrupted. For this reason, *NEBB* does not support multi-flit packet bypass with non-empty buffers when flit-by-flit allocation is employed.

Therefore, SSRs have to meet the following conditions before placing their request in SA-G to activate the bypass:

1. If the packet size is one flit, then the SSR can set-up the bypass and the switch independently of the input VC buffer occupation.

2. If the packet has more than one flit, then the input VC buffer has to be empty.

3. In both cases, the input VC has to be in a state other than active (already forwarding a packet).

The packet size field in SSRs is checked before performing SA-G to validate/check if the packet is single flit.

Summarizing, in SMART with multi-packet buffers and NEBB, only single-flit packets can make use of the bypass paths when intermediate buffers are not empty.

5.3.3) Packet-by-packet arbitration

To achieve that multi-flit packets can take the bypass with non-empty intermediate buffers, SMART++ employs packet-by-packet arbitration (PPA). PPA is implemented using a grant-hold circuit [Dally2003] coupled to the round-robin arbitration stages (SA-G and SA-L, discussed in Section 5.3.5).

Grant-hold circuits hold the arbiter outcome for a certain amount of time. When a multi-flit packet header wins arbitration, SMART++ logic locks the arbiter to the winning packet. However, winning SA-G does not guarantee that a flit will be transferred in the following cycle: the flit could suffer a premature stop in an upstream router in the multi-hop. To cover this case, SMART++ releases the grant in two cases: when the packet tail is received, or when no flit is received. Grant-holding is not required for single-flit packets.

Effectively, this makes SMART++ behave exactly as VCT, receiving all the flits of a packet in consecutive cycles. Only packet headers generate SSRs. When an intermediate router receives a high-priority SSR while a packet is traversing the bypass path, the router ignores the SSR and stores the corresponding packet. This behavior does not conflict with the *single priority enforced in the network* requirement of SMART because it does not introduce false positives (a non-expected flit arrives at a router), only premature stops. Additionally, these premature stops do not reduce performance: they always occur because another packet is actually traveling towards the desired output[1].

Therefore, the following conditions have to be met in a multi-hop intermediate router to grant the bypass to a packet:

1. There is an available destination VC in the next router with enough room for the packet.

2. There is a VC in other state than active in the input unit of the router that receives the SSR.

3. Another packet does not hold the switch and bypass path.

The implementation of the previous conditions requires the SSR's packet size field. This mechanism only generates SSRs for head flits because SA-G follows a packet-basis arbitration.

Packets cannot be fragmented when using PPA, i.e., the flits of a packet are forwarded in consecutive cycles, without gaps between them. Therefore, all the buffer slots used by a packet can be considered as free when forwarding the head flit, reducing the time required to enable the *avail_vc* flag for multi-flit packets. Figure 5.4 shows an example

[1]There are no cascading invalidations, as occurs with SSRs using Prio=bypass [Krishna2013].

comparing the activation time of the *avail_vc* signals when using flit-by-flit and packet-by-packet arbitration.[2] With flit-by-flit arbitration the blue packet cannot advance to R_1, until there are 4 slots available as it is uncertain if the flits of the green packet will progress in consecutive cycles. Thus, it takes 8 cycles to forward the blue packet. By contrast, with packet-by-packet arbitration, R_1 activates the $avail_4$ signal when the head of the green packet leaves the buffer, as the remainder flits will leave the router in consecutive cycles. In this way, the flits of the blue packet use the slots that the green packet is leaving. In this case, it takes 5 cycles to forward the blue packet.

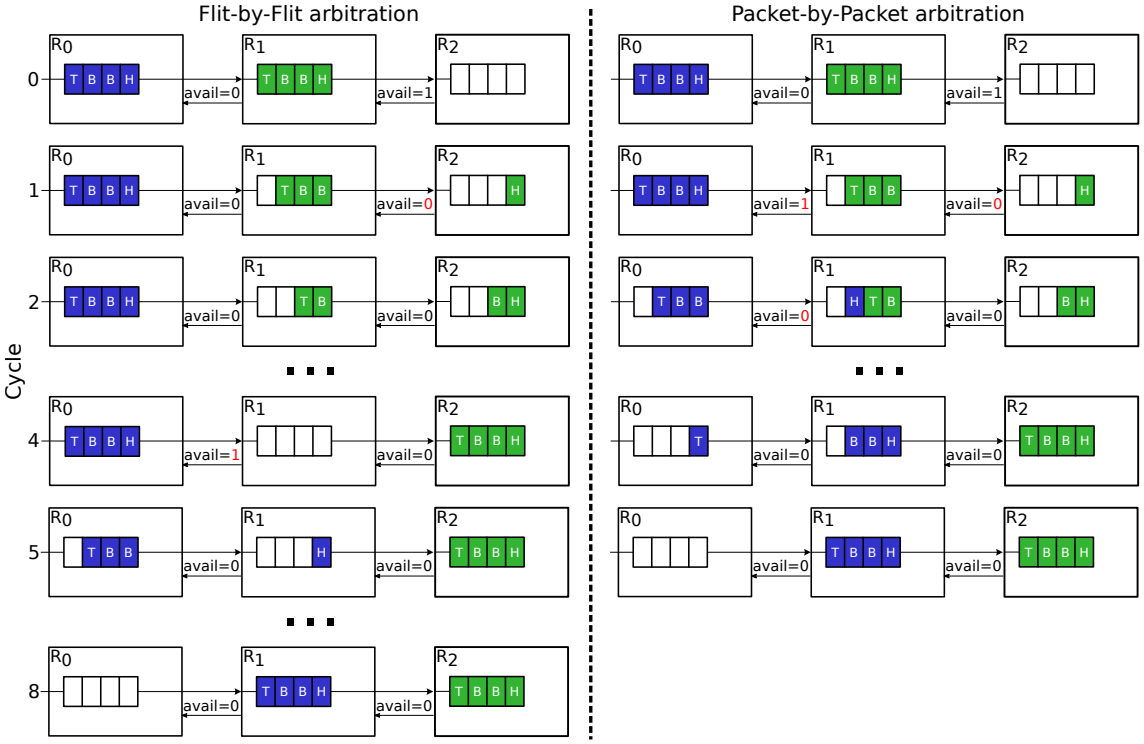


Figure 5.4: Activation of availability signals (avail) when using flit-by-flit and packet-by-packet arbitration.

5.3.4) Comparative analysis of the mechanisms

Table 5.1 summarizes the different cases in which bypass is supported in SMART and SMART++, detailing the specific contribution of each of the mechanisms SMART++ comprises. SMART routers can only forward packets when the next router has an empty buffer and set the bypass if they have an empty input VC. MPB supports forwarding packets to non-empty buffers, but packets can only take the bypass path in intermediate routers with an empty input VC. NEBB adds support for single-flit packet bypass of non-empty buffers. Finally, PPA completes SMART++, which supports bypassing non-empty buffers for any packet size.

Figure 5.5 depicts two examples that compare the conditions presented in Table 5.1, for both single- and multi-flit packets. In all cases the destination of the blue packet is R_4. When using SMART, the packets stop at R_1 because the buffers of R_2 are not empty

[2]This example does not take into account the pipeline of the routers so flits move between buffers instantaneously.

Table 5.1: Bypass activation depending on the buffer status. *Bypass* and *Dest. buf.* refers to the buffers in the bypass router and in the next router. When multiple routers are bypassed, intermediate buffers are both *bypass* and *dest.* buffers. They may need to be completely *empty*, or may accommodate at least a whole *packet*.

	Bypass buf.: empty		Bypass & Dest. buf.: packet	
Bypass mechanism	**Dest. buf.: empty**	**Dest. buf.: packet**	**1-flit packet**	**Multi-flit packet**
SMART (Baseline)	✓			
SMART+MPB	✓	✓		
SMART+MPB+NEBB	✓	✓	✓	
SMART++ (MPB+NEBB+PPA)	✓	✓	✓	✓

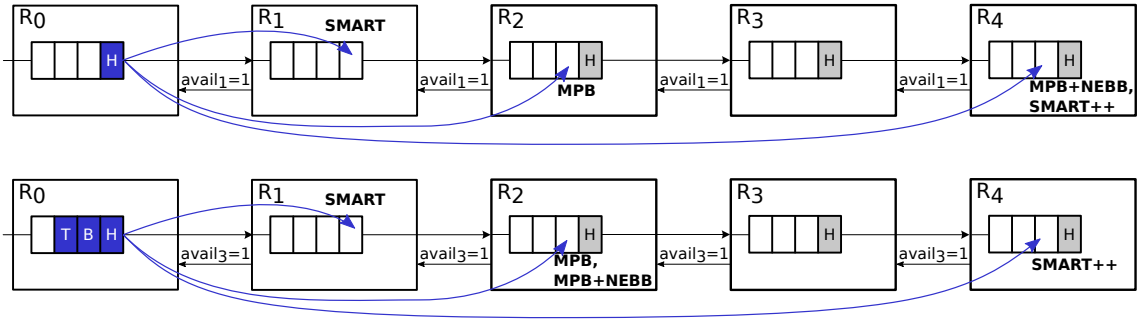


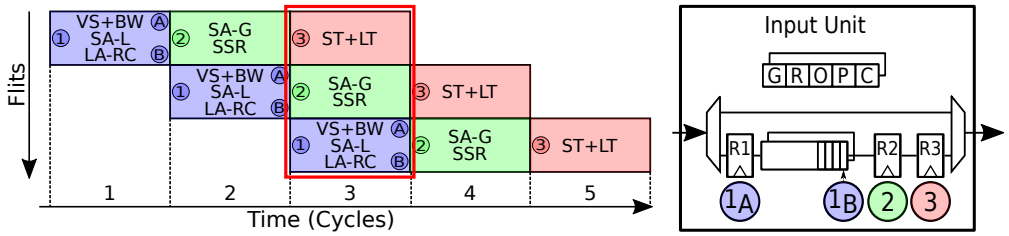
Figure 5.5: Stop router of each mechanism in SMART++ for single-flit (up) and multi-flit (bottom) packets. R_4 is the destination of the blue packet in R_0. Routers only have one buffer.

in both cases. In the case of MPB, the packets stop at R_2 in both cases because MPB only allows bypassing when the input buffer is empty. MPB+NEBB allows bypassing non-empty VCs for single-flit packets, so the single-flit packet reaches R_4 while the 3-flit packet stops at R_2. Finally, SMART++ allows bypassing non-empty VCs regardless of the packet size, so in both cases the packet reaches R_4.

5.3.5) SMART++ input unit architecture

SMART++ implements a three stage pipeline like SMART. However, it targets efficient designs with few input buffers, ideally one. Such configurations may introduce architectural dependencies between the pipeline operations of consecutive packets that reuse the same VC, which causes idle cycles, also known as bubbles, as described in [Dimitrakopoulos2015]. In the case of SMART++ (including partial implementations described previously in this section), three stages may need to access flit information in the same cycle. In the example presented in Figure 5.6a this occurs in cycle 3, when the first flit is doing ST+LT, the second one SA-G, and the third one SA-L. Using the input unit organization of SMART generates architectural dependencies that may lead to stalls because flits cannot exit from the head of the buffer until completing SA-G in stage 2. In a design without VCs, this would interrupt the transmission of packets sharing the same input buffer. This is because the next packet in the queue cannot place a request in SA-L in stage 1 while the front packet is performing SA-G. Pipeline bubbles increases latency and reduces throughput.

This issue is avoided in SMART++ by changing the pipeline registers of the input unit



(a) Router pipeline. ***(b)*** Input unit organization

Figure 5.6: SMART++ router: input unit organization and pipeline.

as depicted in Figure 5.6b. The input unit has three pipeline registers *R1-R3*. Flits are dequeued to R_2 when winning SA-L in the first stage, transferring the following flit to the front position of the buffer. In this organization, VS and BW in stage 1 read the flit from the first register (*1A*), whereas RC and SA-L also in stage 1 read their data from *1B*, i.e., the front of the input buffer (if it is not empty) or register *R1* (if the buffer is empty). In the latter case, the flit advances directly to *R2* avoiding the buffer, which typically acts as another pipeline register. SA-G reads the flit from register *R2*, while ST reads the flit from *R3*.

SMART++ implements VCT as defined in Section 5.3.3, so buffer credits may be handled per-packet. For single-flit packets, the credit is sent back to the upstream router when the flit advances to *R2*. However, flits may wait indefinitely in *R2*. For multi-flit packets, we notice that when the header advances to *R3*, it is certain that the packet flits will be transferred consecutively. For this reason, credit handling for multi-flit packets may be optimized as follows: when a packet header advances to *R2*, one credit is generated; when the first body flit advances to *R2*, the remaining credits are generated.

5.4 Evaluation

This section evaluates SMART++. Section 5.4.1 describes the simulation methodology and the experimentation parameters; Section 5.4.2 presents performance results; and Section 5.4.3 shows power, resource utilization and maximum frequency results;

5.4.1) Methodology

The evaluation methodology uses BST, introduced in Chapter 3, and consists of both functional simulations and estimations of power, area, and frequency based on FPGA synthesis. Next, we describe the simulation infrastructure and parameters.

5.4.1.A) Simulation Infrastructure

We have implemented cycle-accurate models of SMART and SMART++ in the BookSim version of BST, as described in Chapter 3, including the partial versions detailed in Table 5.1. These models support multi-flit packets and, for simplicity, they use credits. BookSim's back-pressure mechanism is based on credits, and each topology defines the credit channels between routers. Thus, the adaptation of SMART to use credits fits better

in BookSim's architecture than adding a new back-pressure mechanism, while performance results are the same.

The SMART++ model of OpenSMART is based on the SMART implementation with the modifications mentioned in Section 3.2 that fix some errors. The SMART++ implementation works with credits and is limited to single-flit packets, like the SMART implementation. All the models in both tools use *router bypass* (Section 2.2.3). The OpenSMART models are also used to validate the latency and throughput results of the BookSim models, through BSV functional simulations. This type of simulation is fast in terms of execution time but very resource-heavy and time-consuming for compiling, making it difficult to use for experimentation with a high degree of variable parameters. The estimations of power, resource utilization and frequency, are done using Quartus Prime 18.1 Lite Edition, synthesizing and measuring the metrics on an Arria II EP2AGX45DF29I5 FPGA. We feed the power analysis tool with VCD (value change dump) files generated from ModelSim functional simulations with a clock frequency of 50MHz.

Table 5.2 gathers the most relevant simulation parameters. Unless otherwise stated, these are the default parameters of experiments in next sections.

Table 5.2: Simulation parameters.

Parameter	BookSim	OpenSMART
Topology	4×4 and 8×8 meshes	4×4 mesh
Bypass mechanism	SMART, SMART++ and partial versions	SMART and SMART++
Bypass type	SMART_1D with router bypass	
Router size	5 ports	
VC number & backpressure	1, 2, 4 or 8 VCs using credits	
Buffer size	1, 2, 4, 5, 8, 10, 15 or 20	1, 2, 4 or 8
Packet size (flits)	1, 5 or bimodal (80% of 1 + 20% of 5)	1
Routing	DOR XY	
VC selection policy	Shortest queue	First available VC
SSR policy	Priority to local flits	
HPC_{MAX}	3 or 7	3
Flit size	128 bits	32 bits

All the evaluations of SMART and SMART++ only propagate SSRs in one dimension (SMART_1D), given that SMART_2D focuses on reducing the base latency and SMART++ on improving the efficiency of the buffers and bypass. Performance simulations on BookSim evaluate 8×8 meshes with $HPC_{Max} = 7$, which is the maximum possible value for this topology. Validation simulations are evaluated in 4×4 meshes with $HPC_{Max} = 3$ due to the large requirements of the BSV compiler. In both cases, local flits have priority over SSRs as it is the best policy so far [Krishna2013]. All the simulations use synthetic traffic. Five traffic patterns are evaluated: random-uniform, bit-complement, tornado, transpose and hotspots in the corners of the topologies. Moreover, we evaluate three packet sizes: single-flit packets, 5-flit packets, and bimodal traffic that combines single-flit and 5-flit packets following a distribution of 80% and 20%, respectively. It emulates the packet distribution observed in full-system simulations of the PARSEC

benchmarks [Ma2012]. We do not elaborate Full-System simulations in this case because the loads generated are very low as shown in the evaluation of Chapter 4. Under these conditions, the difference in terms of base latency between SMART and SMART++ is null, so that the mechanism practically does not affect the whole system performance (execution time). Therefore, we focus on the efficiency of SMART++, i.e., achieving similar performance with less VCs.

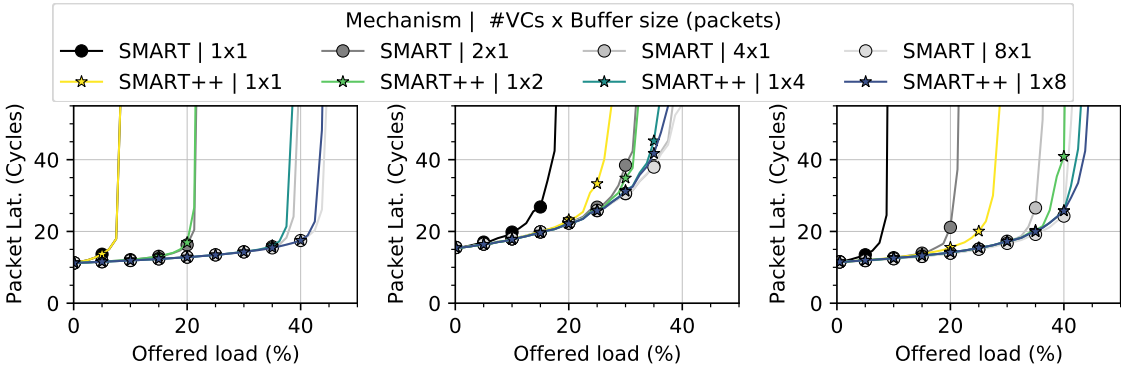
5.4.2) Cycle-level Performance Results

This section evaluates SMART++ without VCs, then with VCs, and finally the contribution of each of its mechanisms.

5.4.2.A) SMART++ without VCs

We compare SMART and SMART++, the first using multiple VCs and the second without VCs (1 VC), but with equivalent total buffer space. First, we evaluate the mechanisms with random-uniform traffic and then with other traffic patterns.

Figure 5.7 shows the average packet latency for 1-flit packets, 5-flit packets and a combination of 1-flit and 5-flit packets (bimodal traffic). With single-flit packets and for the same amount of buffering per input port, the performance of SMART++ is similar to SMART. In the case of 5-flit packets, SMART++ achieves more throughput when the buffer space is low, 5 and 10 slots, and similar for 20 and 40. With a 5 slots buffer, SMART++ outperforms SMART throughput by 48.7% with 1 VC. This is thanks to PPA's early activation of the *avail_vc* signals when the head of a multi-flit packet starts its traversal to the next routers (Sections 5.3.3 and 5.3.5). With bimodal traffic, the performance improvement of SMART++ over SMART is even larger. It requires half of the buffer space of SMART to practically obtain the same performance, which is a consequence of supporting multi-packet buffers. In SMART, single-flit packets are overusing the whole buffer size, which is required to hold 5-flit packets.

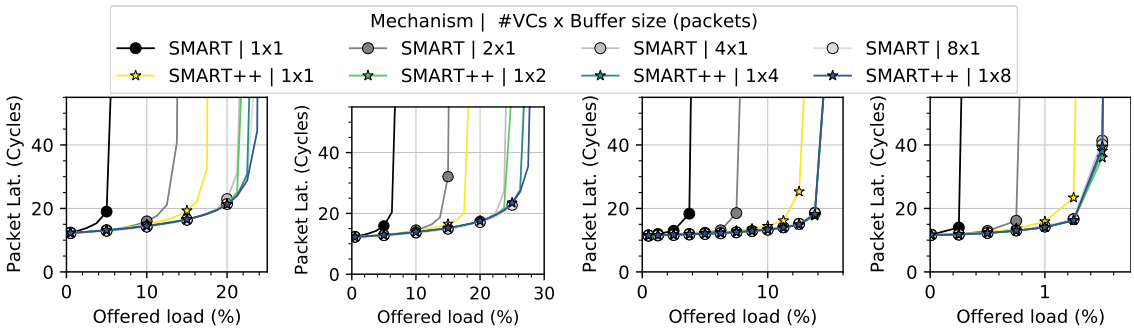


(a) Single-flit packet latency. **(b)** 5-flit packet latency. **(c)** Bimodal traffic packet lat.

Figure 5.7: Packet latency for different packet sizes in SMART and SMART++. SMART++ only employs 1 buffer (no VCs). The size of buffers is relative to the maximum packet size.

Figure 5.8 depicts the packet latency using bit-complement, tornado, transpose and hotspot (in the corners of the mesh) traffic patterns for bimodal traffic. We only show results of bimodal traffic because the conclusions remain the same as with random-uniform

traffic. Results with both traffic patterns are very similar, requiring buffer sizes of only 2 packets (10 flit slots) to reach the maximum throughput, while SMART requires 4 VCs (20 flit slots in total). On top of that, SMART++, with just one buffer for one packet, improves the throughput of SMART with 2 VCs by 14.1%, 20.7%, 41.1% and 36.4% for bit-complement, tornado, transpose and hotspot.

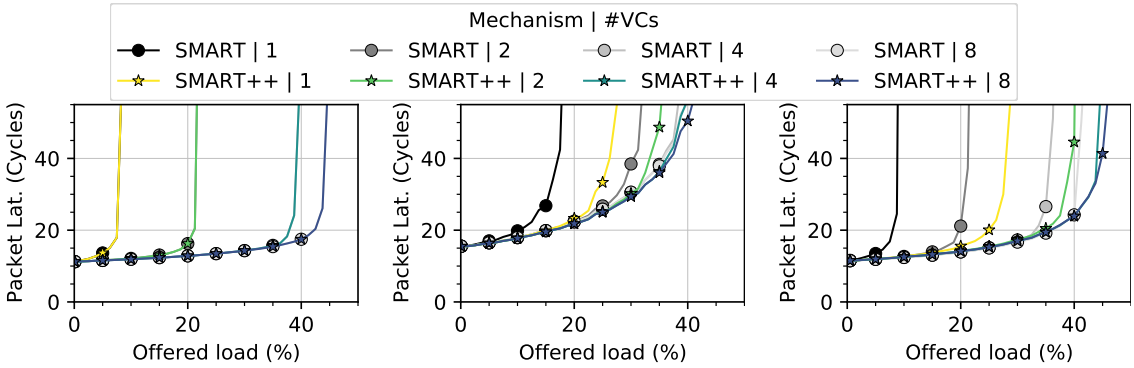


(a) Bit-complement. **(b)** Tornado. **(c)** Transpose. **(d)** Hostpot.

Figure 5.8: Latency of SMART and SMART++ without VCs for bit-complement, transpose, tornado and hotspot (in the corners of the meshes) traffic, with bimodal traffic. The size of buffers is relative to the maximum packet size which is 5 flits.

5.4.2.B) SMART++ with multiple VCs

Next, we compare SMART++ against SMART using the same configuration, i.e., same number of VCs and buffer size. Figure 5.9 shows the latency of both mechanisms. The performance of SMART++ is clearly better than SMART. These results are very similar to the results shown in the previous evaluation without VCs. Even so, SMART++ has slightly more performance when using multiple VC since they mitigate the Head of line Blocking (HoLB) effect. Notice that the results for single-flit packets are practically the same for both mechanisms because this packet size does not exploit the early activation of the *avail_vc* signals, unlike multi-flit packet (Sections 5.3.3 and 5.3.5).

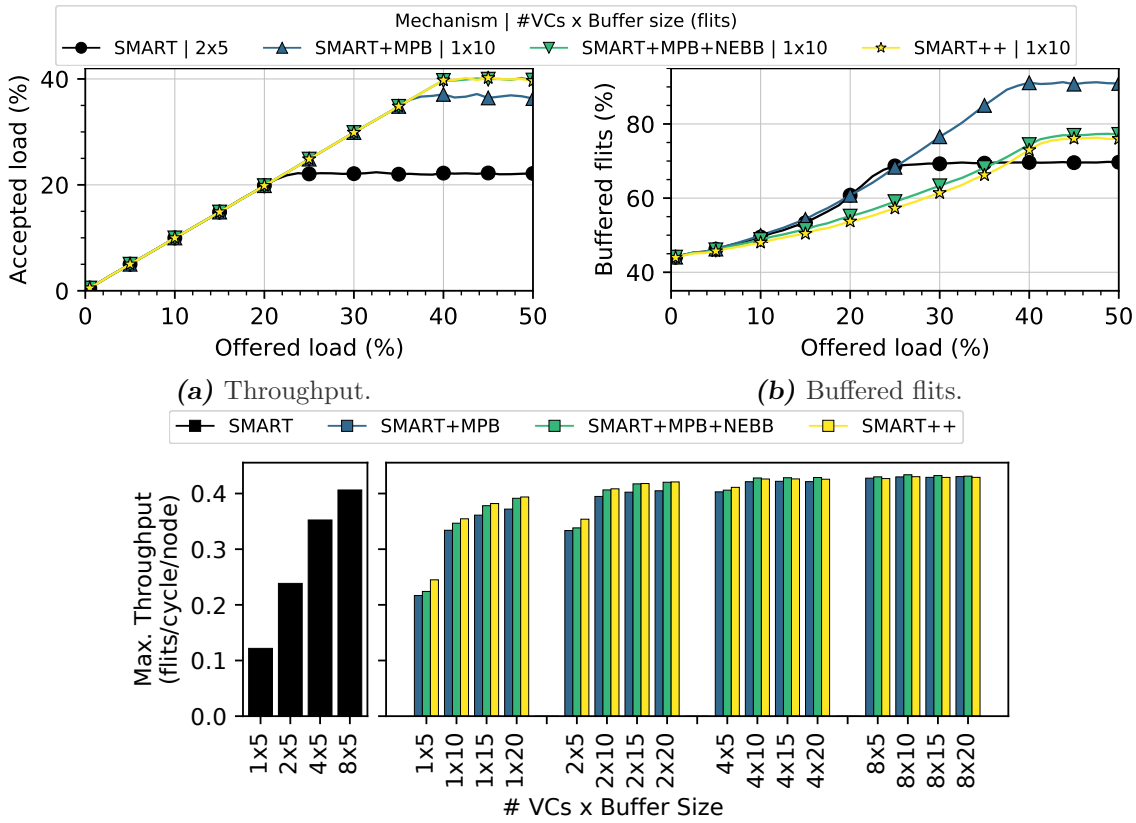


(a) 1-flit packet latency. **(b)** 5-flit packet latency. **(c)** Bimodal traffic latency.

Figure 5.9: SMART vs SMART++ packet latency with multiple VCs and minimal buffer size per VC.

This evaluation breaks down the performance of SMART++ by incrementally incorporating the four mechanisms that compose it: SMART, SMART+MPB, SMART+MPB+NEBB and SMART+MPB+NEBB+PPA (SMART++).

Figure 5.10 shows throughput and buffer utilization results when injecting bimodal traffic. First, Figures 5.10a and 5.10c show the evolution of the throughput with the injected load, for a fixed buffer configuration with space for two packets of 5 flits in total, and the maximum throughput for different configurations, respectively.



(a) Throughput. ***(b)*** Buffered flits.

(c) Maximum throughput.

Figure 5.10: Performance of the partial implementations of SMART++ using bimodal traffic.

SMART clearly depends on the number of VCs to extract the maximum throughput possible in the NoC. The theoretical bound of an 8×8 mesh is 0.5 flits/node/cycle (50%). Implementing MPB support, the throughput increases drastically since VC buffers are used efficiently. In the case of the configuration with buffers of 10 flits (SMART with 2 VCs of 5 slots vs SMART+MPB with 1 VC of 10 slots) SMART+MPB increases the throughput of SMART by 39.7%. The other two mechanism increase even further the throughput, thanks to the efficient utilization of the bypass: SMART+MPB+NEBB by 45.1% and SMART++ by 48.5%, both with respect to SMART. Additionally, the maximum throughput chart (Figure 5.10c) show the low dependency of SMART++ (and the rest of the partial implementations) on the number of VCs. The total amount of buffer space is more important. So, in this particular case, SMART++ without requiring VCs and with half the resources is very close to the maximum throughput achieved by SMART.

Particularly, SMART++ with 1 VC of 20 slots achieves 3.0% less throughput than SMART with 8 VCs of 5 slots. Beyond 20 slots in total, the impact of increasing the buffer size is negligible. In summary, SMART++ does not require VCs to exploit the mesh bandwidth, while reducing the optimal overall buffer space.

Second, Figure 5.10b shows the buffer utilization of the mechanisms with 10 flits overall. In general, a lower buffer utilization means that the bypass utilization is higher. Starting with SMART+MPB, the buffer utilization is slightly lower just before saturation, which is positive. The buffer utilization after saturation is higher because it has a higher maximum throughput. Following with SMART+MPB+NEBB, the utilization is notably lower than with the previous mechanism. This is the result of applying NEBB (only for single-flit packets) which increases the opportunities of taking the bypass. Lastly, when including PPA, the effect of bypassing 5-flit packets when the buffers are not empty reduces the utilization of the buffers. The use of bimodal traffic practically hides the effect of PPA in this evaluation. Moreover, the implementation of PPA is simpler than flit-by-flit arbitration. Note that increasing the utilization of the bypass does not only reduce latency at medium to high loads but also the dynamic energy consumption, as shown in the evaluations of Chapter 4.

5.4.3) Synthesis results

This section shows the evaluations on the Arria II FPGA. The first part validates the BookSim models used in the previous section with the real OpenSMART implementations of SMART and SMART++. The second and third parts show the logic resources, power consumption, and maximum frequency on the FPGA. The final part scales the latency results of Section 5.4.2.A using the maximum frequency estimated in this section.

5.4.3.A) Model Validation

To validate the models implemented in BookSim we compare packet latency results using BSV simulations of the OpenSMART implementations. We simulate a 4×4 mesh with $HPC_{max} = 3$, random-uniform traffic and single-flit packets.

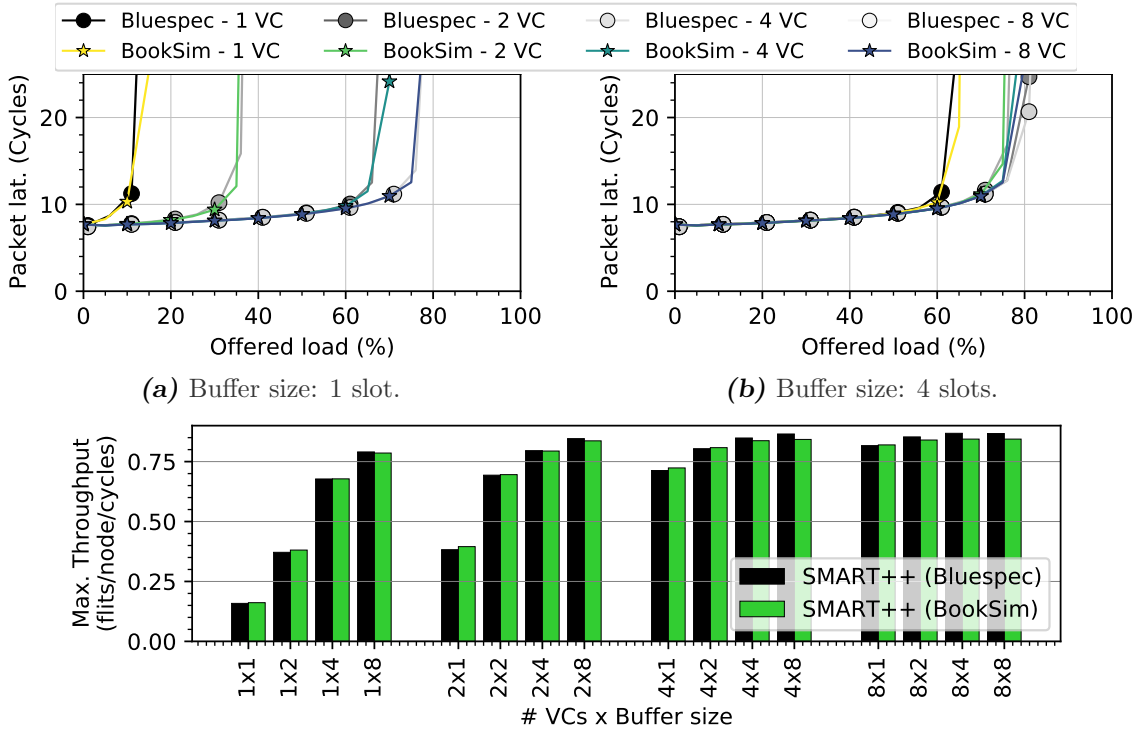
Figure 5.11 depicts the packet latency and maximum throughput of both implementations for various buffer configurations. The packet latencies obtained form both models (Figures 5.11a and 5.11b) are equal until reaching the saturation region where there is a negligible difference. In terms of maximum throughput the difference are insignificant. The highest relative error between models is 3.53%, when using 2 VCs of 1 slot.

5.4.3.B) Resource Analysis

This section analyzes the resource utilization of SMART and SMART++. We focus on just one router because of the large duration of the synthesis process (including place and routing, etc), the large number of configurations evaluated and the limited number of resources in the FPGA.

Figure 5.12 shows the number of Adaptive Look-Up Tables (ALUTs), Adaptive Logic Modules (ALMs), dedicated registers and internal block memory bits of the FPGA used. These results show the high impact of the number of VCs on resource demands.

The number of VCs directly impacts the size and complexity (control logic and registers) of the input unit, credit units (credit handling logic) and VA. When duplicating the



(a) Buffer size: 1 slot.

(b) Buffer size: 4 slots.

(c) Maximum throughput.

Figure 5.11: Comparison of packet latency and throughput of the SMART++ models implemented in BSV and BookSim.

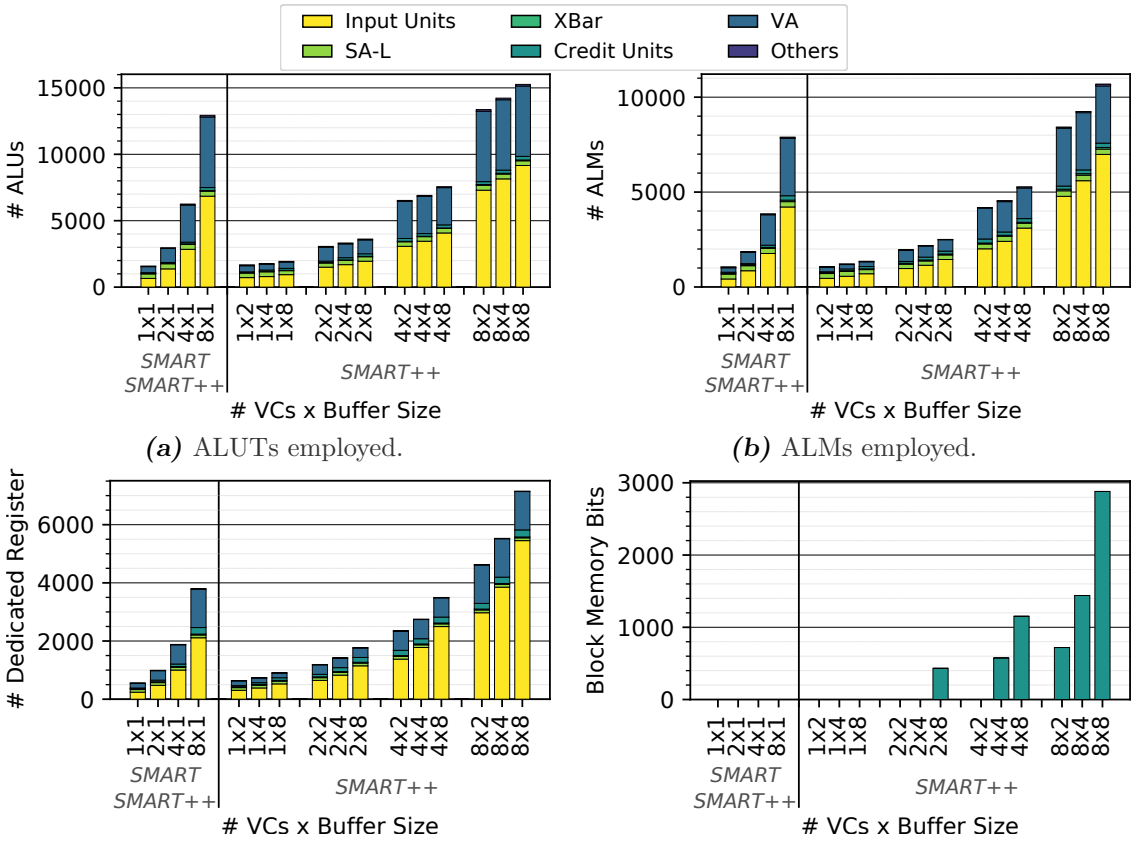
number of VCs, the number of resources is almost doubled. For example, the configuration with 2 VCs of 1 slot increases the number of ALUTs by 86.9%, ALMs by 82.3% and registers by 77.63% with respect to 1 VC of 1 slot. Alternatively, the resource utilization grows in a much lighter way when using deeper buffers. For example, using 1 VC of 8 slots requires 22.3%, 31.4% and 63.14% more ALUTs, ALMs and registers, respectively, than using 1 VC of 1 slot.

The FPGA synthesizer employs block memory (internal FPGA RAM) when buffers larger than 10 slots are instantiated. This occurs in the credit unit for configurations with more than 8 buffer slots. This is a consequence of the implementation of OpenSMART's credit units. Each unit has two FIFO structures to store credits pending to be transmitted, one for credits from the standard pipeline and the other for the bypass path. Each of the FIFOs has to have as many entries as buffer slots has an input port to avoid overflow. In implementations with *free_vc* or *avail_vc*, these structures do not exist as they use pools of free or available VCs instead to manage the back-pressure.

5.4.3.C) Timing and Power Analysis

Figure 5.13 depicts the maximum operation frequency and the dynamic power consumption. The dynamic power has been obtained for a clock frequency of 50MHz, which is achievable by all the configurations (see Figure 5.13a). Static power has been omitted because it is almost constant for every configuration in the FPGA.

Both in terms of frequency and power, the results reveal that the number of VCs is a critical design factor. Doubling the number of VCs decreases frequency in a range



(a) ALUTs employed.

(b) ALMs employed.

(c) Registers employed.

(d) Memory employed.

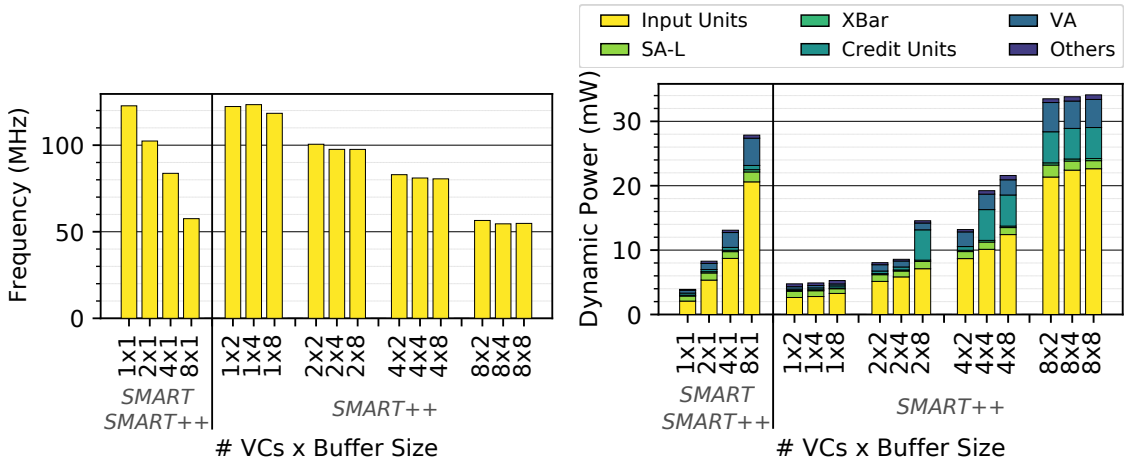
Figure 5.12: FPGA resources employed by each configuration.

between 18% and 29% in each step. The dynamic power almost doubles when duplicating the number of VCs. For example, 2 VCs of 1 slot multiplies by 1.99 the power of 1 VC of the same size. However, increasing the buffer depth has a negligible impact on frequency. Abrupt increases on dynamic power are caused by the use of block memory bits (Figure 5.12d) as mentioned before, being 75.84% the worst case (2×8 compared to 2×4 slots). Comparing the 8×1 (VCs $\times$ buffer size) SMART and the 1×8 SMART++ configurations, resources are reduced by 5.49× on average and dynamic power by 4.99×.

5.4.3.D) Scaled SMART++ performance results

Section 5.4.2 presents performance results in discrete time units, i.e., cycles without considering the frequency of each configuration. In this evaluation, we use the maximum operation frequencies depicted in Figure 5.13a to scale the performance results in order to observe the effect of reducing the number of VCs.

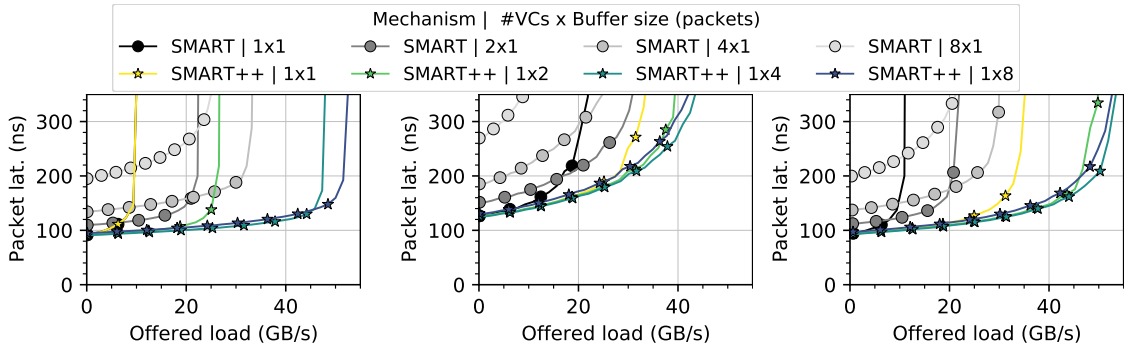
The maximum frequency is typically determined by the first two router stages which comprise routing computation, VC management and switch allocation. The delay of the third stage increases as the HPC_{max} value grows and high values might lower the router frequency. In such case, the frequency is given by the length of the maximum multi-hop and not by the complexity of the router, so SMART and SMART++ should have similar frequencies and their performance would be proportional to results of Section 5.4.2. In the case of moderate HPC_{max}, the performance will be determined by the router maximum frequency.



(a) Max. operation frequency. ***(b)*** Dynamic power.

Figure 5.13: FPGA frequency and dynamic power results.

Figure 5.14 presents frequency-scaled latency results of SMART and SMART++. A simple SMART++ configuration with 1 VC with a buffering space for 4 packets (4 slots for single-flit packets and 20 for 5-flit packets and bimodal traffic) clearly outperforms any SMART implementation. Comparing the same configuration to SMART with 4 VCs, which is a competitive configuration in terms of throughput, SMART++ has 32.1% less zero-load latency and 42.2% more throughput, for single-flit packets which is the most conservative comparison.



(a) Single-flit packet latency. ***(b)*** 5-flit packet latency. ***(c)*** Bimodal traffic packet lat.

Figure 5.14: Frequency-scaled latency of SMART and SMART++ using different packet sizes.

5.5 Conclusions

SMART reduces latency in topologies such as the mesh with the introduction of multi-hop bypass to reduce the effective number of hops. However, power and area efficiency are essential NoC features to meet the budgets of the CMPs design. The main issue of SMART in this regard is that it requires a large number of VCs to exploit the advantages of the mechanism due to conservative restrictions in the use of the bypass and buffers to

avoid packet-interleaving.

In this chapter we have described the fundamentals and implementation considerations of SMART++, and presented a detailed evaluation. SMART++ is a multi-hop bypass mechanism that does not require VCs. Its goal is to improve the efficiency of SMART, increasing the utilization of the available space in buffers and the bypass. SMART++ allows multiple packets to share the same buffer and bypassing routers when their buffers are not empty. Thus, it exhibits high performance without VCs, reducing drastically power, area and critical path delay due to the simplification of the input unit organization and VC management. Moreover, SMART++ presents a more efficient utilization of buffers and bypass, which translates in latency reductions and dynamic power savings at medium-high loads. Contrary to conventional wisdom, the use of multiple VCs in SMART++ just provides a marginal improvement.

For the same frequency and similar performance, SMART++ reduces area and power by 5.5× and 5.0×. SMART++ improves the performance in presence of congestion, but it does not reduce the router pipeline. For this reason, the base latency in cycles is the same as in SMART. However, selecting the maximum frequency, SMART++ may reduce base latency by up to 31.9% and increase throughput by up to 42.2%. In sum, the efficient design of SMART++ simultaneously provides near-optimal performance with a small footprint and reduced implementation cost.

Chapter 6

Speculative-SMART++

In the background of this thesis (Chapter 2) we have mentioned that single-hop bypass routers reduce the router delay while multi-hop bypass routers reduce the effective average number of hops in common 2D grid-based topologies. Multi-hop bypass routers are not an exact extension of single-hop bypass routers. One of the most important characteristics of single-hop bypass is that it can chain multiple bypass hops in consecutive cycles, completely avoiding the standard pipeline of the routers. In other words, a packet is able to complete its route without being buffered. In SMART and SMART++, each multi-hop needs to employ the complete pipeline of the router to prepare the next multi-hop, incurring a latency of 3 cycles per multi-hop.

In this chapter we describe Speculative-SMART++, a mechanism that combines the best of single-hop and multi-hop bypass to cost-efficiently reduce the latency of NoCs. S-SMART++ employs a bypass pipeline for consecutive multi-hops, reducing base latency and the relevance of HPC_{Max}.

The chapter is organized as follows. Section 6.1 describes the fundamentals and implementation considerations of S-SMART++. Section 6.2 evaluates the performance of the mechanism with synthetic and realistic traffic, the power consumption, the area required, and the maximum frequency. Section 6.3 summarizes the conclusions of the chapter.

6.1 S-SMART++: speculative SSR broadcast

Speculative-SMART++ (S-SMART++) relies on speculative SSRs to chain consecutive multi-hops with a single cycle per multi-hop. This idea is based on single-hop bypass, where routers can generate a subsequent LA when a previous LA wins LA-Arb. This idea works with both SMART (S-SMART) and SMART ++ (S-SMART ++). False negatives (activation of the bypass without receiving the corresponding packet) make it impossible to know whether a packet will arrive or not, so the creation of the next SSR for the next multi-hop has to be speculative. We start this section with an overview of how the mechanism works when applying it to SMART (Section 6.1.1). Then we describe the router architecture (Section 6.1.2), followed by a detailed analysis of its behavior (Section 6.1.3). The section concludes with some implementation considerations when applying the mechanism to SMART and SMART++ in Section 6.1.4, and its extension to torus networks in Section 6.1.5.

6.1.1) S-SMART overview

Single-hop bypass networks [Kumar2007; Kumar2008; Krishna2010] pre-allocate bypass paths to skip the buffering and allocation stages. The idea of SMART of conforming multi-hop paths by pre-allocating bypass paths is very similar to them, but their principles are slightly different. SMART seeks to minimize latency by conforming the largest multi-hop paths, whereas single-hop bypass chains multiple hops in consecutive cycles. SMART is more effective in terms of latency, specially in large networks. However, single-hop bypass optimizes per-hop latency and allows skipping all the buffers in the whole route, optimizing dynamic power. In SMART this is not possible, as the packet is buffered after each multi-hop.

Speculative-SMART combines both approaches to support chaining several multi-hops in consecutive cycles, skipping the first and second pipeline stages in subsequent multi-hops. In S-SMART, SSRs are generated speculatively in the last router of each multi-hop, where the packet would be buffered. Therefore, while a packet is traversing the routers, a speculative SSR (spec-SSR) is requesting the subsequent bypass paths.

In SMART, global arbitration is speculative, since a packet may not reach the maximum desired hop length because of a conflict. However, after sending an SSR, SMART always sends data on the multi-hop. S-SMART exploits speculation even further, since the SSR itself is also speculative (spec-SSR), given that it is generated before even knowing if the associated packet will reach the router in time for the multi-hop. Therefore, a spec-SSR may not be followed by any data, when it conflicts in an intermediate router in the previous multi-hop.

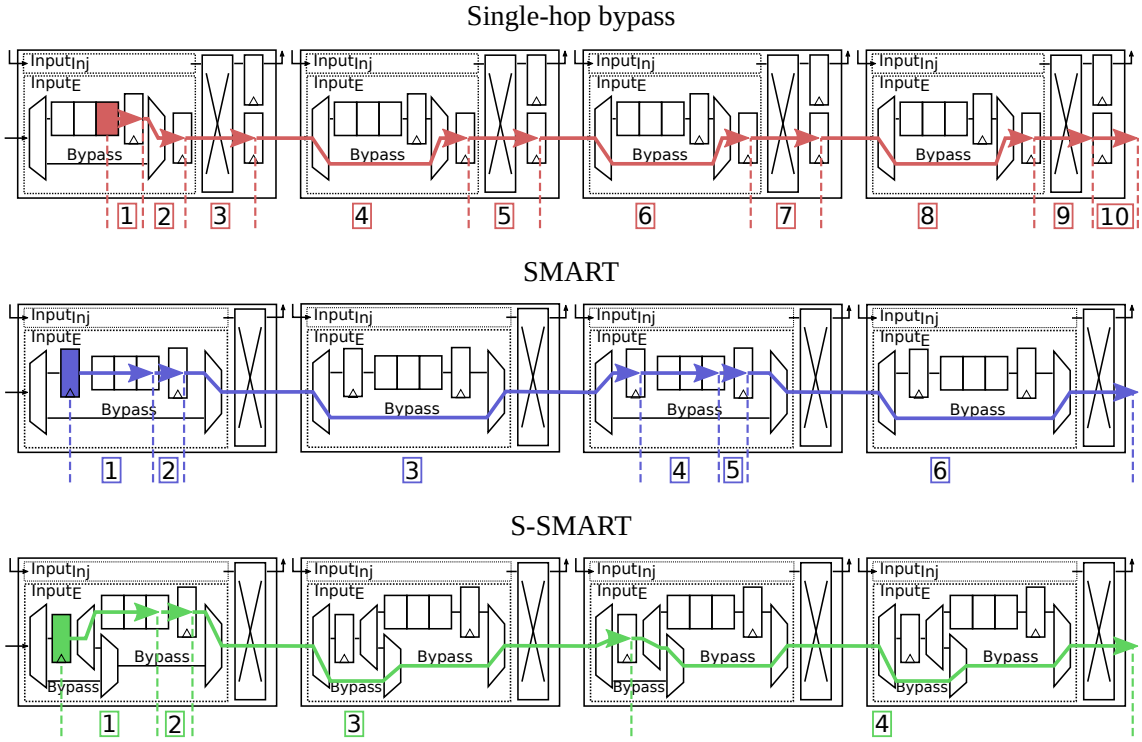
Figure 6.1a presents a comparative example of a flit crossing the network when using single-hop bypass, SMART, and S-SMART, the latter two with $HPC_{Max} = 2$. Note that the behavior of S-SMART with $HPC_{Max} = 1$ would be equivalent to single-hop bypass. Figure 6.1b depicts the pipeline stages of the router and their temporal behavior. They are reviewed next.

- **Single-hop bypass**: The packet in R_0 (in red) wins SA-I, SA-O and VA (VC Allocation) in the first 2 cycles. In the next two cycles, it traverses the crossbar and the link, while the LookAhead travels through the link and acquires access to the crossbar and the bypass in LA-Arb of R_1. In cycle 4, the LA succeeds in LA-CC and originates the creation of another LA from R_1 to R_2. In this mechanism, the arrival of the packet to R_1 is guaranteed in cycle 5 so the second LA is not speculative. In the next two routers, the bypass and crossbar acquisitions and the packet traversal follow the same procedure. Overall, The packet spends 10 cycles to cross the four routers.

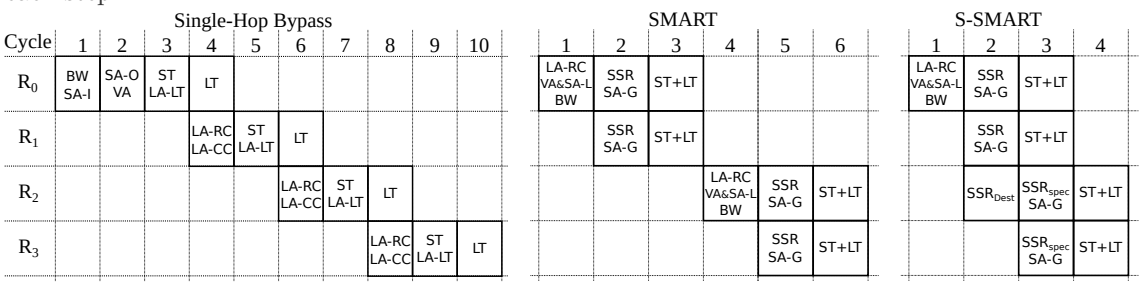
- **SMART**: In R_0, the packet (in blue) performs LookAhead Routing Computation (LA-RC), VA, SA-L and Buffer Write (BW) in the first cycle. In the second cycle, it propagates the SSR to R_1 ($HPC_{Max} - 1 = 2 - 1 = 1$) to acquire the bypass through SA-G, and in the third cycle, it performs the multi-hop to reach R_2. The SSR does not reach R_2, since this final router cannot set up the bypass to meet the temporal constraints. In R_2, the packet repeats the same process. Overall, the packet spends 6 cycles to cross the routers.

- **S-SMART**: Like in SMART, the packet (in green) spends 3 cycles in R_0. However, in cycle 2 the SSR broadcast is extended to reach R_2. The extended SSR does not request SA-G in R_2, but produces the broadcast of a speculative SSR (spec-SSR) in

cycle 3, which acquires the bypass of R_2 and R_3. Thus, in cycle 4, the packet takes the bypass path (instead of performing BW like in SMART). Overall, the packet requires 4 cycles to traverse the four routers.



(a) Flit forwarding in different bypass-router architectures. Boxed numbers represent the cycle of each step.



(b) Pipelines of different bypass-router architectures.

Figure 6.1: Comparison of single-hop bypass, SMART and S-SMART, the last two with $HPC_{Max} = 2$. The boxes placed together with the arrows indicate the cycle when the flit advances through the router paths.

6.1.2) Router architecture

S-SMART introduces modifications in three elements of the design of SMART: the generation of SSRs, the SSR priority scheme, and the bypass control. Figure 6.2 shows the router architecture of S-SMART and Figure 6.3 a partial implementation of SA-G for the path from the west input to the east output. The highlighted elements are the changes over the SMART design.

In S-SMART, the length of SSRs is extended by one unit so that they can reach the final router of the multi-hop. SSRs carry the final destination of the packet to determine

the next multi-hop direction and length. The *'final'* router of a multi-hop is determined using the multi-hop length requested in the SSR, information already carried in SMART. The multi-hop length is compared with the distance to the multi-hop requester, which is known from the ID of the SSR input port. The *'final'* router uses the final destination of the packet to generate the spec-SSR in the next cycle, instead of computing SA-G. This is depicted in Figure 6.3, with the other modules of SA-G: input arbitration (SSR Priority Arbitration), output arbitration, and bypass setup logic. Additionally, the router that generates a spec-SSR also propagates the request to its own SSR Priority arbitration logic in the SA-G unit ($Spec - SSR_{dist=0}$ in Figure 6.3) to activate its bypass.

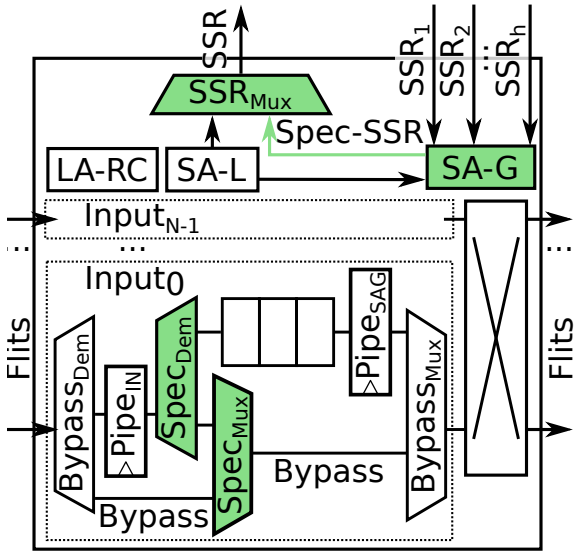


Figure 6.2: S-SMART router implementation. The additional elements included with respect to SMART are highlighted in green.

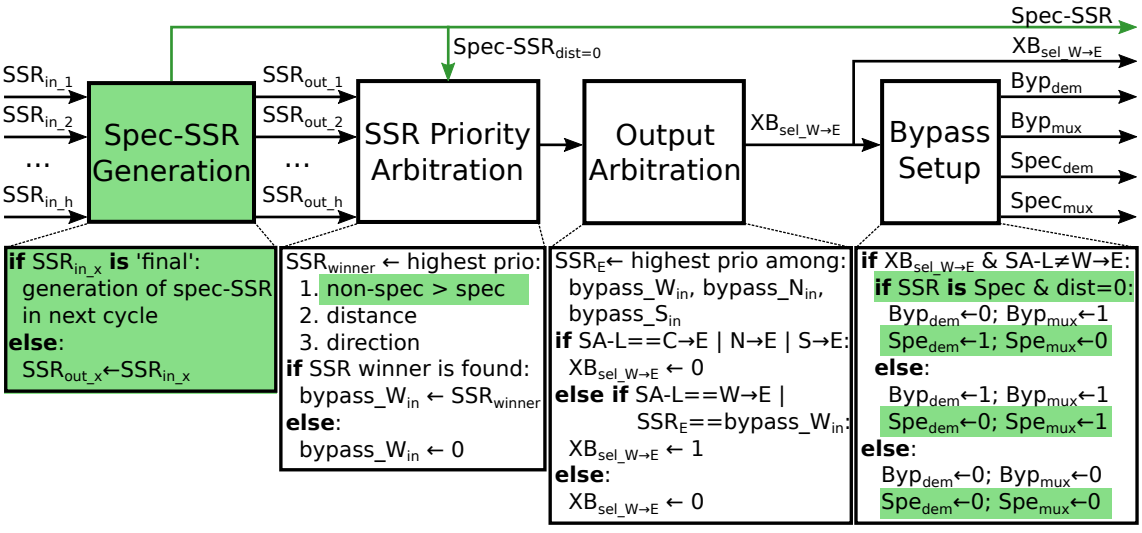


Figure 6.3: Implementation of SA-G for W_{in} to E_{out}. S-SMART additional elements are highlighted in green. $XB_{sel_{W->E}}$ is the selection signal of XBar to enable the path between input West and output East; Byp_{Mux}, Byp_{Dem}, Spe_{Mux}, and Spe_{Dem} are the selection signals of $Bypass_{Mux}$, $Bypass_{Dem}$, $Spec_{Mux}$, and $Spec_{Dem}$, respectively.

6.1.2.A) SSR priority scheme

The SSR priority scheme is modified to resolve conflicts between spec-SSRs and standard SA-G requests (SSRs or local flits). We give absolute priority to standard requests over spec-SSRs, to minimize unnecessary premature stops. The reason comes from the *local priority* policy used in SMART, i.e., priority to the SSR originated in the nearest router, which is the best policy evaluated in [Krishna2013]. With this shortest-distance policy, packets may not complete their whole multi-hop. For this reason a spec-SSR may win SA-G in a router but leave the bypass path unused in the following cycle, because the associated packet was stopped in an intermediate router of the previous multi-hop. Giving low priority to spec-SSRs prevents other packets from stopping prematurely due to a speculative bypass acquisition that is not used.

Conflicts between spec-SSRs and standard requests can occur in the input phase of SA-G or the SSR output phase. The first kind of conflict, in the input phase of SA-G, occurs when standard SSRs and/or spec-SSRs share the same input port. When the conflict is between a standard SSR and a spec-SSR, the spec-SSR is ignored as mentioned before. This requires one extra bit to identify spec-SSRs. When the conflict is between two or more spec-SSRs the arbitration follows the same policy used for standard SSRs, i.e., the SSR with the shortest distance has priority. The second kind of conflict, in the SSR output phase, can occur between a standard SSR (generated by a local flit) and a spec-SSR, or between two or more spec-SSRs. In the first case, the conflict is solved with a multiplexer (SSR_{Mux} in Figure 6.2) at each SSR output port, which discards the spec-SSR. The second conflict occurs because SSRs coming from different input directions at the same cycle can generate spec-SSRs for the same output port. In such case, only one of them is chosen. We choose the one with the longest multi-hop to maximize the utilization of the bypass paths.

6.1.2.B) Bypass control

The standard bypass of SMART is implemented by a demultiplexer ($Bypass_{Dem}$) and a multiplexer ($Bypass_{Mux}$). S-SMART implements an additional path to bypass the input buffer from the input pipeline register ($Pipe_{In}$) when spec-SSRs win SA-G. This path is formed by a demultiplexer ($Spec_{Dem}$) and a multiplexer ($Spec_{Mux}$), depicted in Figure 6.2. These mux/demux pairs are controlled together. Overall, there are three paths. The standard bypass path from $Bypass_{Dem}$ to $Bypass_{Mux}$ is used when an SSR wins SA-G, except for speculative SSRs generated in the local router (generated distance 0). The second bypass path from $Pipe_{In}$ to $Bypass_{Mux}$ is used when a spec-SSR with distance equal to 0 (local) wins SA-G. Finally, the traditional path from $Pipe_{In}$ to the buffer is used when local flits win SA-G.

6.1.3) Speculative bypass walk-through

This section describes the process of using the bypass paths speculatively through an example. Figure 6.4 shows the state of the network during the four cycles that the highlighted packet needs to make two multi-hops. The process is divided into two phases. The first phase extends from cycles 1 to 3, when the packet prepares and performs the multi-hop bypass like in the original design of SMART. The second phase shows the bypass via speculative arbitration, in cycles 3 and 4.

Cycle 1) The packet in R_0 requests a local output port in SA-L while the route for the

next multi-hop is computed in LA-RC. The packet wins SA-L (there are no competitors) and moves to the next pipeline stage.

Cycle 2) The SSR is propagated to the next two routers ($HPC_{Max} = 2$ in this example) to prepare the multi-hop. The SSR received in R_1 performs SA-G, while R_2 ('final' router) uses the destination information to compute LA-RC required to generate the spec-SSR in the next cycle. The packet wins SA-G in R_0 and R_1, advancing to the next stage and preparing the control signals of the crossbars and paths for the next cycle.

Cycle 3) The packet traverses R_0 and R_1, reaching R_2, where it is saved in the input port latch ($Pipe_{In}$ in Figure 6.2). Simultaneously, R_2 generates the spec-SSR from the destination information saved in the previous cycle, and sends it to its local SA-G and R_3 (and R_4 in case it exists). Both spec-SSRs win SA-G because there are no competing requests.

Cycle 4) The packet in $Pipe_{In}$ of R_2 travels through $Spec_{Dem}$ and $Spec_{Mux}$ towards the bypass path of R_2 and R_3.

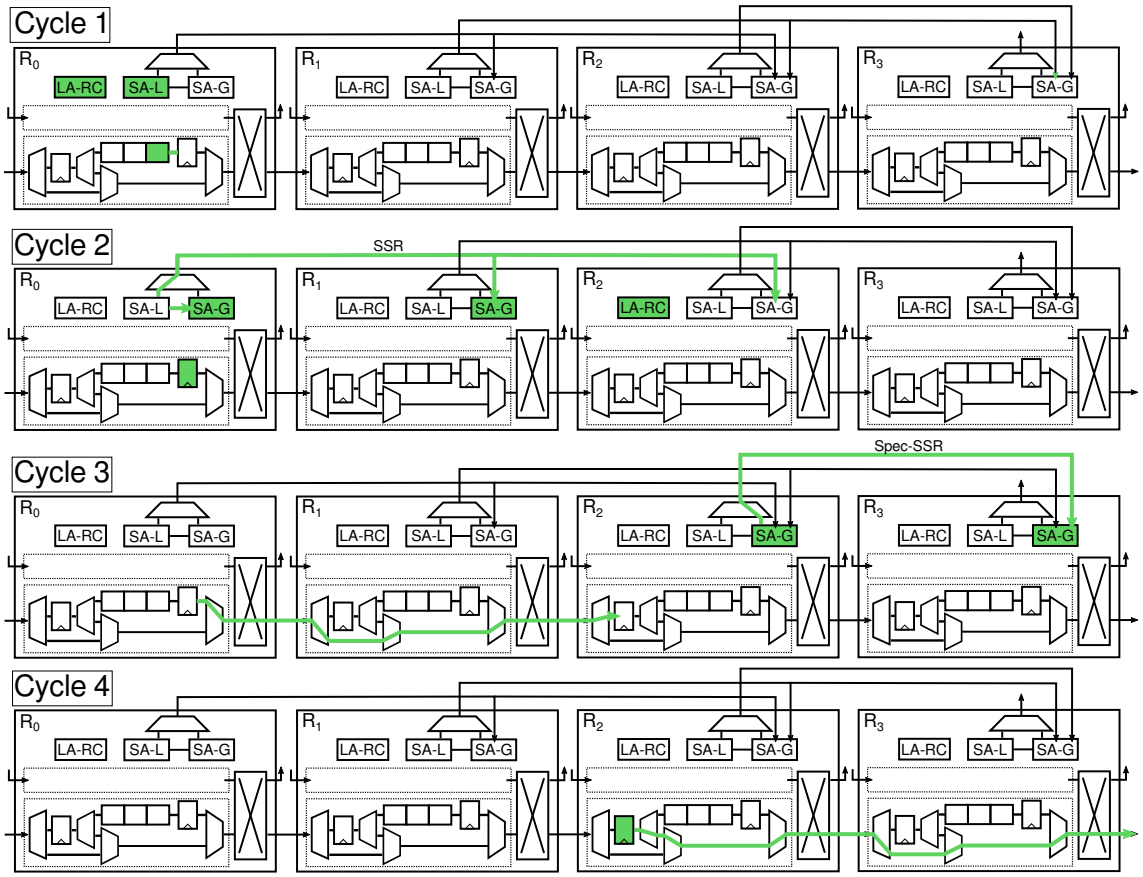


Figure 6.4: Example of speculative SA-G arbitration in S-SMART. HPC_{Max} is 2.

6.1.4) Speculative bypass in SMART and SMART++

As mentioned in previous sections, speculative bypass acquisition can be applied to both SMART and SMART++. Apart from the differences between SMART and SMART++

already stated in Chapter 5, the only distinction between S-SMART and S-SMART++ is that the former generates SSRs per flit and S-SMART++ per packet. Thus, S-SMART++ preserves the packet-by-packet arbiter of SMART++ so SSRs and Spec-SSRs are created only from head flits. Therefore, in S-SMART++ the bypass paths, taken speculatively or not, are locked for the whole packet once the head flit traverses them, and released after forwarding the tail flits.

6.1.5) Speculative-SMART++ in torus NoCs

Like happened with single-hop bypass routers, SMART is not compatible with bubble-like flow controls to avoid deadlock and has to use an alternative mechanism, such as Dateline. The reason is the utilization of *Empty VC Forwarding* (EVCF), which does not allow holding bubbles in the buffers for other packets in transit within a dimension. This does not occur in SMART++ and S-SMART++ since they implement *multi-packet buffers*, allowing multiple packets and bubbles waiting in the same buffer. In this case, SMART++ and S-SMART++ are not compatible with Flit-Bubble Flow Control (FBFC), because they use VCT flow control so the bubble has to have a size of the maximum packet size.

6.2 Evaluation

This section evaluates S-SMART++. Section 6.2.1 describes the simulation infrastructure; Section 6.2.2 presents cycle-accurate performance results with synthetic traffic and Full-System (FS) simulations; Section 6.2.3 shows synthesis estimations of power, resource utilization and maximum frequency.

6.2.1) Simulation Infrastructure

We have implemented models of S-SMART and S-SMART++ in the BookSim version of BST (Chapter 3). We also have developed a real-design implementation of S-SMART++ written in BSV based on the SMART++ one of OpenSMART from BST. Like OpenSMART, this model is limited to single-flit packets and works with credits. The implementation uses *router bypass* instead of *buffer bypass*, despite the fact that *buffer bypass* is preferable for S-SMART++. The reason is that packets can change the traveling dimension when taking the bypass speculatively, like occurs in SMART_2D as explained in Section 2.2.3.E. This produces conservative power, area, and frequency results for S-SMART++. The BSV implementation is also used to validate the latency and throughput results of the BookSim models, through BSV functional simulations as in Section 5.4.3.A. The BSV compiler is used to generate Verilog code that is synthesized with Quartus Prime 18.1 Lite Edition to measure power, area and maximum frequency on an Arria II EP2AGX45DF29I5 FPGA.

We employ three types of simulations: functional with synthetic traffic, FS simulations and validations with the BSV simulator. Tables 6.1 and 6.2 gather the most relevant network and gem5 simulation parameters, respectively.

We focus on comparing S-SMART++ against SMART, because the performance of SMART++ is equivalent to SMART in terms of cycles but with lower hardware cost

Table 6.1: Network simulation parameters.

Parameter	BookSim	gem5	BSV
Mesh size	4×4, 8×8 & 16×16	4×4 & 8×8	4×4
Torus size	8×8 & 16×16	-	-
Bypass mechanism	SMART and S-SMART++		
Bypass type	buffer bypass		router bypass
Router size	5 ports		
Back-pressure	Credits		
SMART VCs	8	12	1, 2, 4 & 8
S-SMART++ VCs	1	3	1, 2, 4 & 8
SMART buf. size (packets)	1 packet		
S-SMART++ buf. size (packets)	8	4	1, 2, 4, 8
Packet size (flits)	1 & 5		1
Routing	DOR XY		
VC selection policy	Shortest queue	Shortest queue	First available VC
SSR policy	Priority to local flits		
HPC_{Max}	1, 2, 3, 4, 7 & 15	3 & 7	3
Flit size	128 bits		32 bits

as shown in Chapter 5. Most of the functional simulations evaluate 8×8 meshes with $HPC_{Max} = 7$. There are also evaluations of 8×8 tori and 16×16 meshes and tori, with different values for HPC_{Max}. Some experiments also evaluate 8×8 SMART_2D meshes to weight the benefit of the pipeline bypass in S-SMART++. Validation simulations are evaluated in 4×4 meshes with $HPC_{Max} = 3$ due to the large requirements of the BSV compiler and the large number of simulation points obtained. The size of the network is relatively small due to the large requirements of the BSV compiler. We do not use the optimization to bypass the destination router as in the evaluation of SMART++ in Chapter 5. In all cases local flits have priority over bypass.

Synthetic traffic simulations evaluate five traffic patterns [Dally2003]: random-uniform, bit-reversal, transpose, tornado and hotspot (with traffic distributed evenly among 4 hotspots at the corners of the grid). We evaluate single-flit and 5-flit packets, which are equivalent to control and data packets. Full-system simulations evaluate SMART and S-SMART++ under real workloads in gem5. We simulate the execution of PARSEC [Bienia2011] benchmarks[1] under Linux 4.15.0 in an ARMv8 processor with 16 or 64 cores operating at 2 GHz. Simulations employ the detailed Out-of-Order processor (O3) and interconnection (Ruby) models in gem5. The cache coherence messages are distributed in 3 Virtual Networks (VNs) to break cyclic protocol dependencies. The 16-core model employs a 4×4 mesh with $HPC_{Max} = 3$ and runs the complete Region Of Interest (ROI) of each benchmark with the *simsmall* input sets. The 64-core model employs an 8×8 mesh with $HPC_{Max} = 7$ and runs the *simlarge* input sets, but only for 500 million cycles

[1]Raytrace is missing because of incompatibilities found with our simulation toolset.

Table 6.2: gem5 configuration parameters

Parameter	Value
CPU model	16x & 64x ARMv8 Out-of-Order (DerivO3CPU) @2GHz
Memory model	Ruby @2GHz
Coherence protocol	MESI with two levels of cache (MESI_Two_Levels)
Cache line size	64 Bytes
L1-I cache	private, 32KB, 2-way associativity
L1-D cache	private, 64KB, 2-way associativity
L2 cache	shared and distributed among cores, 16x & 64x 256KB, 8-way associativity
Memory controllers	8x & 16x DDR3-1600 11-11-11 (Location: first and last mesh rows)

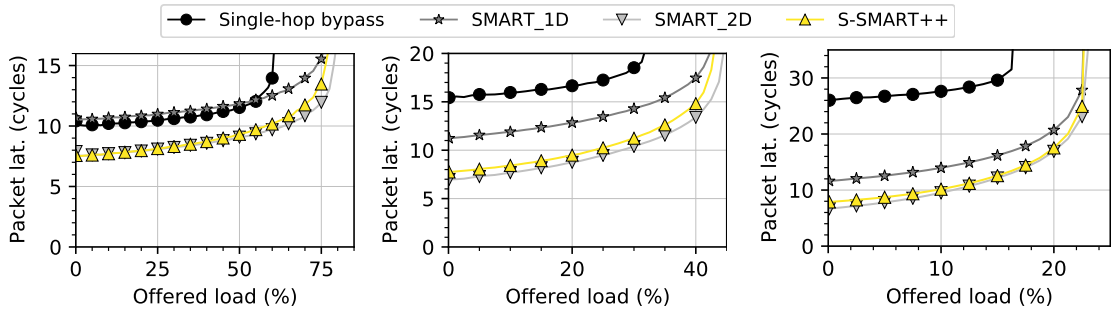
because of its high simulation time.

6.2.2) Cycle-level Performance Results

This section evaluates the performance of S-SMART++ based on the packet latency in terms of cycles, abstracting differences in the maximum operation frequency of the NoCs.

6.2.2.A) Bypass mechanisms comparison

This section compares single-hop bypass (based on NEBB-Hybrid), SMART_1D and SMART_2D with S-SMART++. Figure 6.5 shows the packet latency of the mechanisms in 4×4, 8×8, and 16×16 meshes with HPC_{Max} equal to 3, 7, and 15, respectively. These values cover a dimension of the meshes in one multi-hop. Injected traffic follows a random-uniform distribution and packets have one flit. NEBB-Hybrid and S-SMART++, which allow multi-packet buffering, have a single buffer of 8 slots. SMART_1D and SMART_2D have 8 VCs of 1 slot each.



(a) 4×4 mesh, $HPC_{Max}=3$ ***(b)*** 8×8 mesh, $HPC_{Max}=7$ ***(c)*** 16×16 mesh, $HPC_{Max}=15$

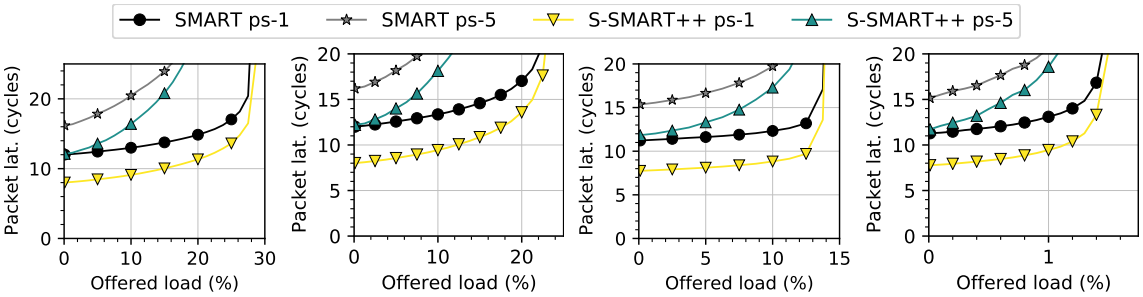
Figure 6.5: Latencies of single-hop bypass, SMART_1D, SMART_2D and S-SMART++ for different mesh sizes.

The results show that single-hop bypass is competitive in small networks because of its shorter per-hop latency. However, its performance quickly degrades with the network size

because the average hop count grows. In contrast, the zero-load latency with multi-hop bypass is almost constant with the network size, given that the effective number of hops is practically constant. S-SMART++ outperforms SMART_1D and almost reaches the performance of SMART_2D. The gain over SMART_1D is practically constant with the size of the meshes due to the values of HPC_{Max}, which are the maximum possible. For example the base latency reduction in the 4×4 mesh is 29.2% while in the 16×16 mesh is 32.1%. In all cases, the base latency remains almost constant since HPC_{Max} always covers a full dimension, whereas throughput halves when the size of the grid (mesh) side duplicates due to halving the ratio between the bisection bandwidth and the number of nodes.

6.2.2.B) S-SMART++ with different traffic patterns

This section compares SMART and S-SMART++ for various synthetic traffic patterns and packets of 1 or 5 flits. Figure 6.6 depicts the results for tornado, bit-complement, transpose, and hotspot traffic patterns, in an 8×8 mesh.



(a) Tornado. ***(b)*** Bit-complement. ***(c)*** Transpose. ***(d)*** Hotspot.

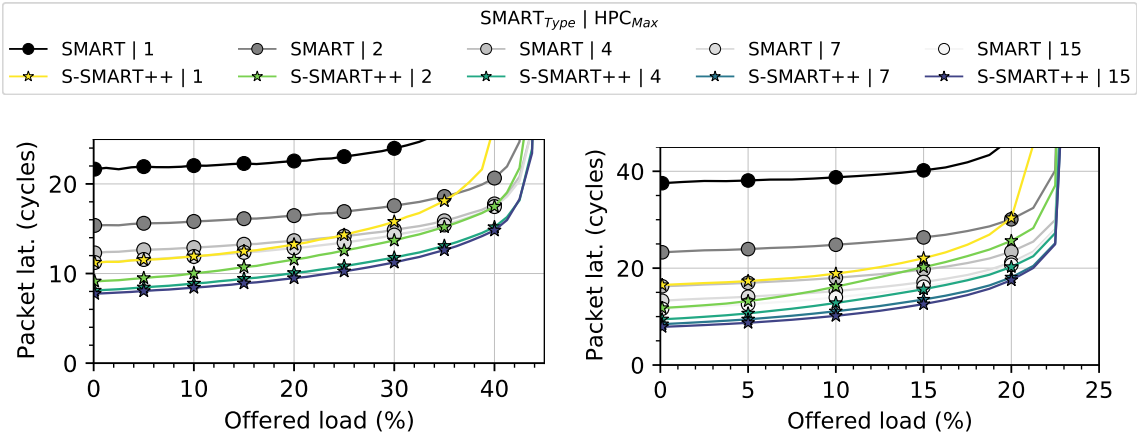
Figure 6.6: Latency for various traffic patterns in an 8×8 mesh with $HPC_{Max} = 7$ and packet sizes of 1 (*ps-1*) and 5 flits (*ps-5*).

The results for these four traffic patterns are similar to the case of random-uniform traffic in Section 6.2.2.A. S-SMART++ has lower latency than SMART in every case, with similar throughput in spite of not using VCs. Besides, using multi-flit packets do not have any effect other than increasing the latency due the flit serialization of packets.

6.2.2.C) HPC_{Max} analysis

This experiment studies the impact of HPC_{Max} on the latency of SMART and S-SMART++. We model 8×8 and 16×16 meshes with different values of HPC_{Max}. Figure 6.7 shows the packet latency of both configurations. We focus on single-flit packets and random-uniform traffic to simplify the analysis, given that the trend is similar with other packet sizes and traffic patterns.

As expected, latency improves with larger HPC_{Max} in all cases. However, S-SMART++ outperforms SMART even with low HPC_{Max} thanks to its speculation mechanism. Interestingly, S-SMART++ without multi-hop bypass ($HPC_{Max} = 1$) in the 8×8 mesh has the same zero-load latency than SMART with the maximum possible multi-hop bypass ($HPC_{Max} = 7$). The multi-hop length influence on the packet latency is also reduced, specially at low offered loads. In the 8×8 mesh, between $HPC_{Max} = 1$ and $HPC_{Max} = 7$ the reduction of zero-load latency is 10.42 cycles in SMART, and only 3.47



(a) 8 × 8 mesh ***(b)*** 16 × 16 mesh

Figure 6.7: Packet latency varying HPC_{Max}.

in S-SMART++; in the 16 × 16 mesh, these differences are 24.27 in SMART and 8.08 in S-SMART++.

6.2.2.D) Evaluation with real traffic

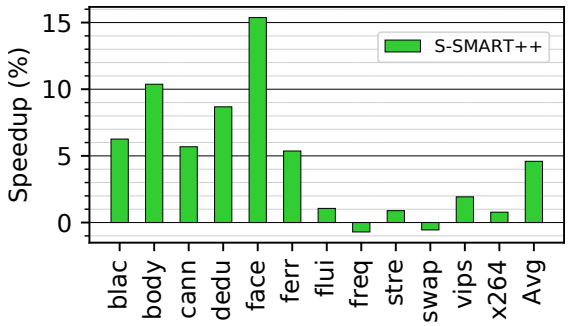
This section analyzes NoC performance using FS simulations running the PARSEC benchmarks. Figure 6.8a presents speedup results of S-SMART++ over SMART with a 16-core system. The average speedup of S-SMART++ is 4.59%. Half of the applications present similar performance in both models. It has to be considered that the distribution of micro-architectural and operating system events over time introduces some performance variability. The other applications present notable speedups for S-SMART++, with a maximum of 15.4%.

Figure 6.8b presents average packet latency results for the same executions. We measured an average network load of only 4.7%, so average latency results are very close to the base latency in most cases, with additional constant delays that correspond to the injection and ejection from the NoC in the memory sub-system. The speculative mechanism of S-SMART++ systematically improves latency in all cases, with an average reduction of 2.77 cycles over SMART, or a 17.4%.

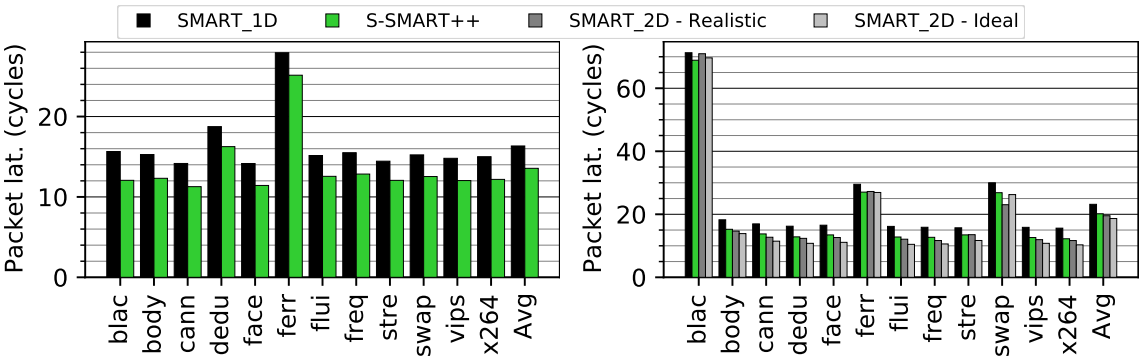
Figure 6.8c presents latency with 64 cores in an 8 × 8 mesh, but only for the initial 500M cycles of the ROI. In this case the initialization is more relevant, and some applications (especially Blackscholes) present larger latency than when observing the full ROI. However, the systematic improvement of S-SMART++ observed with 16 cores is preserved, with an average reduction of 3.01 cycles. This graph also includes results for two variants of SMART_2D, one of them with a very costly but *realistic* HPC_{Max} value ($HPC_{Max} = 7$) and an *ideal* implementation that reaches the destination in a single multi-hop ($HPC_{Max} = 15$). S-SMART++ is very close to the realistic implementation of SMART_2D, and only 1.50 cycles higher than the ideal model on average.

6.2.2.E) SMART and S-SMART++ in tori

This evaluation analyzes SMART and S-SMART++ in torus topologies. The first part of this evaluation compares single-hop bypass, SMART and S-SMART++ in tori with different sizes, like it was done at the beginning of the section with meshes. In this case



(a) S-SMART++ speedup over SMART with 16 cores running the complete ROI.



(b) Average packet latency with 16 cores running the complete ROI.

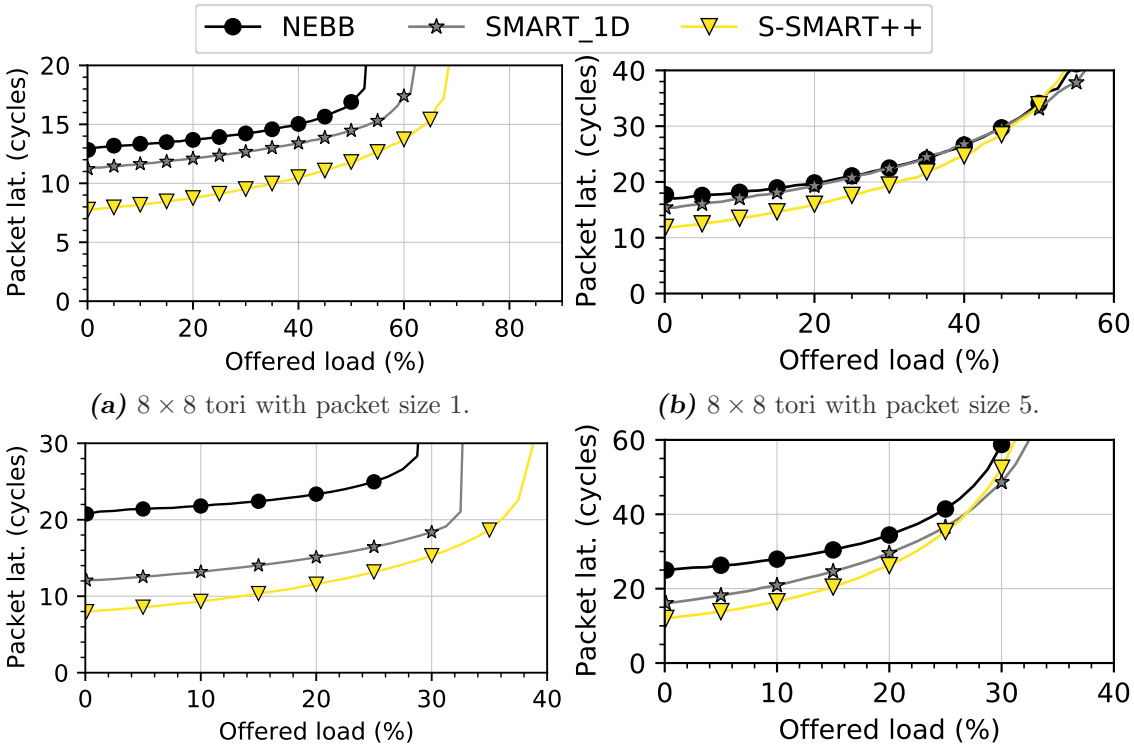
(c) Average packet latency with 64 cores, 500M cycles.

Figure 6.8: S-SMART++ performance on full-system simulations.

we focus on 8×8 and 16×16 torus with HPC_{Max} equal to 4 and 8. These HPC_{Max} are enough to cover the maximum distance between nodes within a dimension. This is one of the advantages of using a torus instead of a mesh as we show later.

Figure 6.9 shows packet latency results for 1-flit and 5-flit packets. Single-hop bypass uses NEBB-Hybrid and FBFC-L with 1 VC for 8 packets (8 slots for 1-flit packets and 40 slots for 5-flit packets), SMART uses dateline with 8 VCs for 1 packet each, and S-SMART++ uses Bubble flow control with 1 VC for 8 packets. The results are similar to meshes, i.e., single-hop bypass suffers high latency degradation when increasing the network size, and S-SMART++ has lower latency than SMART as expected. The latencies with 5-flit packets show similar behaviors. We note that S-SMART++ presents slightly lower throughput than SMART, which may be attributed to the lack of VCs to reduce HoLB.

Figure 6.10 compares the latency of 8×8 and 16×16 meshes and tori using S-SMART++ and varying HPC_{Max}. The first observation is that the torus configurations almost double the throughput of the mesh ones, as expected from the duplication of the bisection width (BW) mentioned in Section 1.4.1. The second observation is that the torus configurations reduce the latency with respect to their mesh counterparts, due to the reduction of the average distance. This is most noticeable with low values for HPC_{Max} as a result of having higher effective number of hops. The most important effect of employing a NoC topology with lower average distance is the reduction of the maximum HPC_{Max} as mentioned before. For example in the case of the 8×8 networks, the maximum HPC_{Max} is 7 in the mesh and 4 in the torus. In both cases the zero-load latency is the same because the effective number of hops is the same, i.e., 1 multi-hop if the destination of a packet is in the same dimension or 2 if packets have to perform a dimension change. For this reason



(a) 8 × 8 tori with packet size 1.

(b) 8 × 8 tori with packet size 5.

(c) 16 × 16 tori with packet size 1.

(d) 16 × 16 tori with packet size 5.

Figure 6.9: Packet latency of 8 × 8 and 16 × 16 tori with bypass. Single-hop bypass uses NEBB-Hybrid with FBFC-L and 1 VC; SMART employs dateline with 8 VCs; and S-SMART++ relies on Bubble flow control with 1 VC.

and given the low core-count of the FS simulations depicted in the previous evaluation, we do not include real-traffic results as it is expected to observe similar performance.

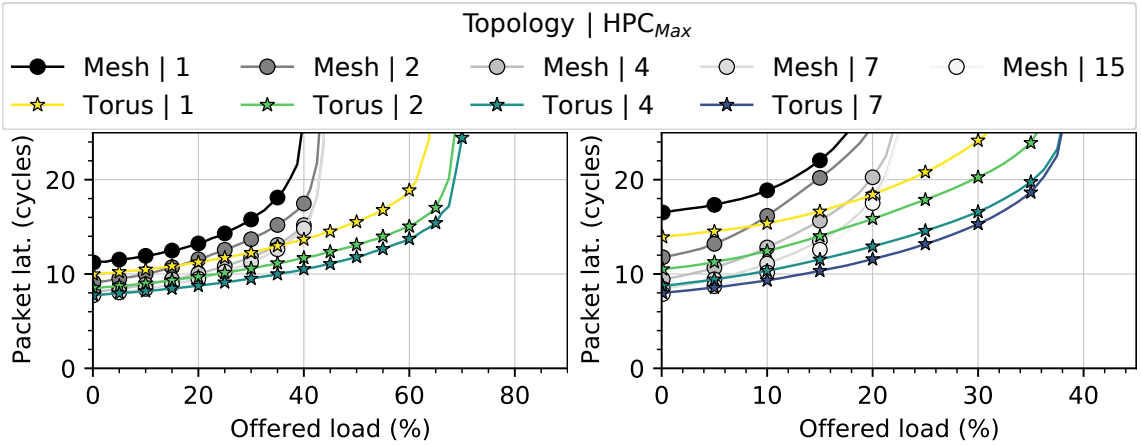
6.2.3) Synthesis results

This section evaluates the S-SMART++ model implemented in OpenSMART with the Bluespec System Verilog tools and Quartus.

6.2.3.A) Model Validation

This section validates the models implemented in Booksim and BSV by comparing their results. The network evaluated is a 4 × 4 mesh with $HPC_{Max} = 3$, single-flit packets and random-uniform traffic. Figure 6.11 shows the results of S-SMART++ in both platforms.

The base latency of both implementations is the same until reaching saturation, where most of the differences are negligible. The highest relative error is 9.77% when using 2 VCs of 1 slot. From these results we consider that the functional models implemented in BookSim, cycle accurately simulate the router architecture and the pipeline, according to the HDL implementation.



(a) S-SMART++ in 8×8 mesh and torus. ***(b)*** S-SMART++ in 16×16 mesh and torus.

Figure 6.10: Packet latency of 8×8 and 16×16 meshes and tori varying HPC_{Max}.

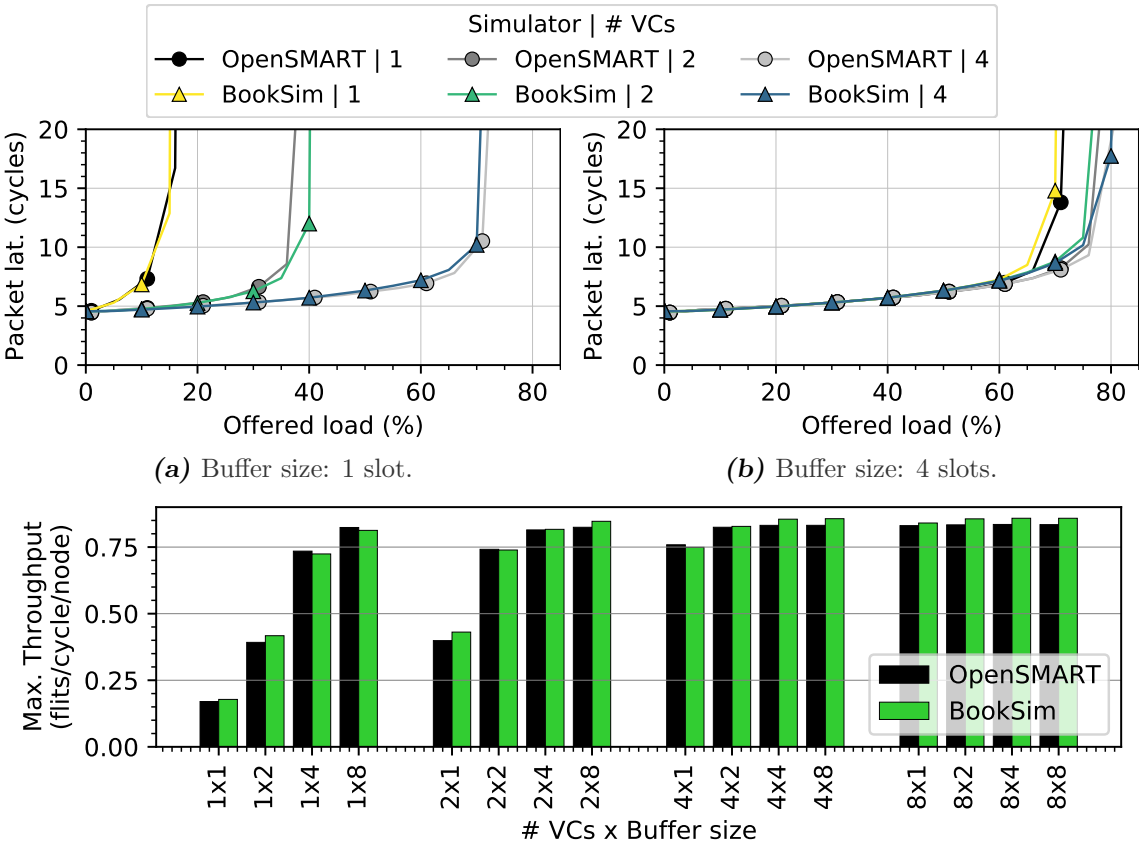
6.2.3.B) Resource Analysis

This section analyzes the resource requirements of a single SMART and S-SMART++ router, for different values of buffer count and depth using packets of 1 flit. Note that SMART is limited to only 1 packet. S-SMART++ targets designs with few deep buffers (1×8 instead of 8×1 in SMART) but configurations with multiple buffers are also considered. Figure 6.12 shows FPGA resources used by the synthesized routers, represented by the number of Adaptive Look-Up Tables (ALUTs), Adaptive Logic Modules (ALMs), dedicated registers and internal block memory bits.

First, the results show the high impact of VCs on resource demands as they are part of the input units, credit units (credit handling logic) and VA. When duplicating the number of VCs, the number of resources is almost doubled. For example, the configuration of SMART with 2 VCs of 1 slot increases the number of ALUTs by 74.3%, ALMs by 62.1% and registers by 80,9% with respect to 1 VC of 1 slot. By contrast, when increasing buffer depth, the resource utilization grows at a much slower rate. For example, in S-SMART++ with 1 VC of 8 slots the number of ALUTs increases by 5.1%, ALMS by 4.6% and registers by 25.9% compared to 1 VC of 1 slot. In some configurations of S-SMART++, like 1×8 slots, the synthesis employs block memory (internal FPGA RAM) to build the buffers of the input and/or credit unit because the FPGA employed does not have enough dedicated registers. This causes a significant power increment for this configuration as shown in Section 6.2.3.C.

The overhead of S-SMART++ over SMART is similar between common configurations. For example with 1 VC of 1 slot, S-SMART++ uses 31.0% more ALUTs, 24.9% ALMs and 1.3% registers than SMART. With 4 VCs of 1 slot, these values are 26.92%, 26.18% and 0.06%, respectively. The logic increase is localized in the VA of OpenSMART, which integrates a large part of SA-G.

However, since S-SMART++ targets few deep buffers, effective configurations are more efficient than in SMART. For example, S-SMART++ with 1 VC of 8 slots employs 80.88% less ALUTs, 82.06% less ALMs and 81.71% fewer registers than SMART with 8 VCs of 1 slot.



(a) Buffer size: 1 slot.

(b) Buffer size: 4 slots.

(c) Maximum throughput.

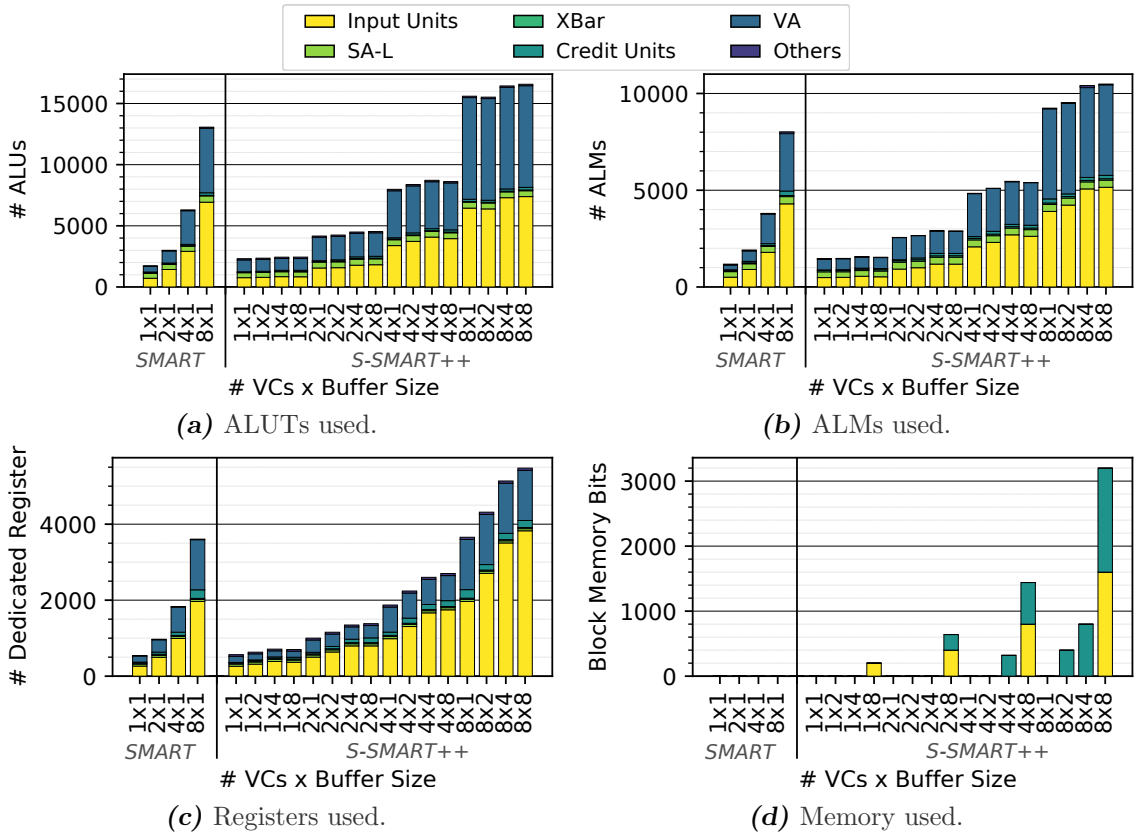
Figure 6.11: Comparison of packet latency and throughput of the S-SMART++ models implemented in BSV and BookSim.

6.2.3.C) Timing and Power Analysis

Figure 6.13 depicts the maximum operation frequency and the dynamic power consumption for multiple SMART and S-SMART++ router configurations. Again, note that S-SMART++ targets deep buffer arrangements such as 1×8, compared to 8×1 in SMART.

To obtain dynamic power results, we feed the power analysis tool with VCD (value change dump) files generated from ModelSim functional simulations with a clock frequency of 25 MHz, which is under the minimum operation frequency of the configurations depicted. The results reveal that the number of VCs is a critical design factor. Focusing on SMART, doubling the number of VCs reduces frequency by between 12.04 and 21.44 MHz in each step. The overhead of S-SMART++ reduces the maximum frequency of SMART by 23.98 to 29.76 MHz for equivalent configurations. However, increasing the buffer depth has a negligible impact on frequency, and S-SMART 1×8 obtains a frequency 46.1% faster than SMART 8×1.

Dynamic power, depicted in Figure 6.13b, almost doubles when duplicating the number of VCs. For example, SMART with 2 VCs of 1 slot multiplies by 1.73 the power of 1 VC with 1 slot. Moreover, increasing the buffer depth in S-SMART++ has a negligible impact on frequency and moderately increases dynamic power. Section 6.2.3.B mentions how the abrupt increment in dynamic power for some configurations (1×8, 2×8, 4×8, 8×2, 8×4, and 8×8) is caused by the use of memory instead of dedicated registers to allocate



(a) ALUTs used. ***(b)*** ALMs used.

(c) Registers used. ***(d)*** Memory used.

Figure 6.12: FPGA resources employed by SMART and S-SMART++.

part of the buffers of the input unit. Despite this, S-SMART++ 1×8 reduces dynamic power by 44.1% with respect to SMART 8×1.

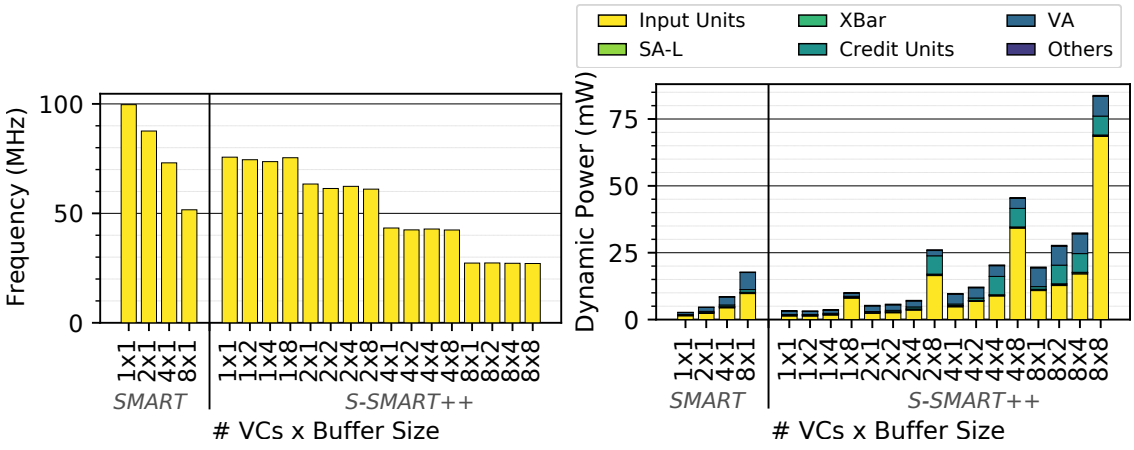
6.2.3.D) Scaled performance results

Similarly to Section 5.4.3.D, this section scales the performance results of Section 6.2.2 that are given in terms of cycles, with the maximum frequency of each configuration depicted in Figure 6.13a. Figure 6.14 presents the frequency-scaled latency of SMART and S-SMART++ with single-flit packets and random-uniform traffic. The figure shows SMART for different VC configurations (2, 4 and 8 VCs) given that more VCs increase throughput but reduce the maximum frequency. For S-SMART++ we only show a configuration with a single VC of 8 slots, equivalent in space and throughput to SMART with 8 VCs of 1 slot, because the frequency variations with the buffer depth are negligible.

From the results, it is clear that S-SMART++ outperforms SMART with lower costs. In terms of latency, S-SMART++ reduces the zero-load latency of SMART with 2 VCs by 45.3%. In terms of throughput, S-SMART++ increases the maximum throughput of SMART with 4 VCs by 13.9%.

The previous evaluation compares SMART and S-SMART++. However, SMART++ improves the efficiency of SMART without incurring in the implementation overhead of Speculative-SMART. Thus, Figures 5.13a and 6.13a show that SMART++ achieves higher maximum frequencies that S-SMART++ for the same buffer configuration.

Figure 6.15 shows the latency of SMART++ and S-SMART++ with the same buffer configuration and varying HPC_{Max}. The buffer configuration is 1 VC of 8 slots. The



(a) Max. frequency. ***(b)*** Dynamic power.

Figure 6.13: FPGA frequency and dynamic power results of SMART and S-SMART++.

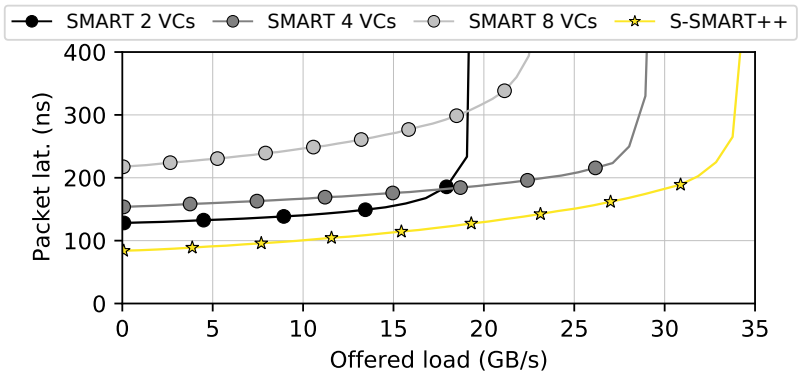


Figure 6.14: Frequency-scaled latency of SMART with different VC configurations and S-SMART++ without VCs.

zero-load latency of S-SMART++ is lower despite having a lower maximum frequency. The difference between both mechanisms is higher for small values of HPC_{Max}. However, the lower frequency of S-SMART++ notably decreases its maximum bandwidth with respect to SMART++. For example, the S-SMART++ configuration with $HPC_{Max} = 2$ reduces the bandwidth of SMART++ by 18.72%. In general, the lower dependency of S-SMART++ with HPC_{Max} makes it a more versatile version, capable of adjusting better to stringent design requirements. It is also important to remark that these results consider that the router architecture determines the maximum frequency instead of HPC_{Max}, which favors SMART++. Nevertheless, SMART++ is a suitable option when bandwidth is a determining factor.

6.3 Conclusions

In this chapter we have described S-SMART++, a multi-hop bypass NoC architecture that makes use of speculative allocation to reduce latency. The target of S-SMART++ is the relatively high router delay of SMART after each multi-hop, which reduces its effectiveness in small meshes and/or with small HPC_{Max} values. Applying S-SMART++ reduces the

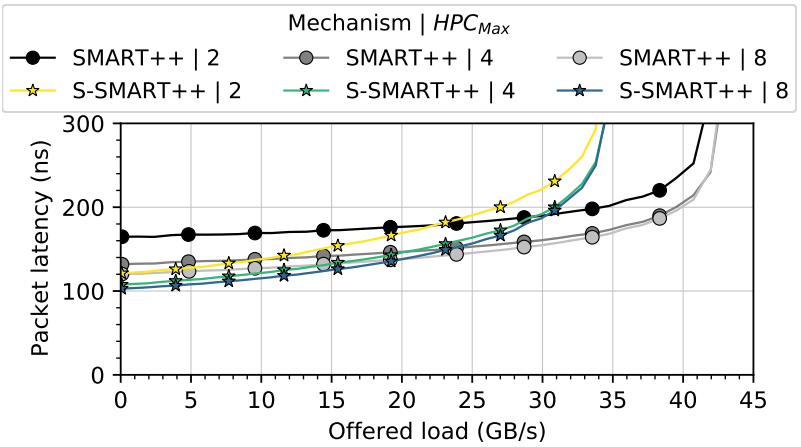


Figure 6.15: Frequency-scaled latency of SMART++ and S-SMART++ varying HPC_{Max}.

minimum router delay from three cycles to one, just like in a single-hop bypass router.

The evaluation results show that S-SMART++ reduces the zero-load latency of SMART_1D and achieves almost the same latency as SMART_2D, which implies an unaffordable cost given that its SSR interconnection grows quadratically with HPC_{Max}. Moreover, S-SMART++ reduces drastically the dependency of its base-latency with HPC_{Max}. The results of area, power, and maximum frequency show that the overhead introduced by the additional logic is not negligible. In short, the combination of SMART++ and Speculative-SMART expands the possibilities of integrating multi-hop bypass NoCs in real CMPs by adjusting critical design parameters such as the topology, the number of VCs and HPC_{Max}.

Chapter 7

Related Work

In this chapter we present and discuss previous work that is related with the contributions of this thesis, either to compare them or to propose their combination in possible future work. The organization of the chapter follows the structure of the thesis: Section 7.1 focuses on BST; Section 7.2 focuses on NEBB in single-hop bypass NoCs; Section 7.3 focuses on SMART++; and Section 7.4 focuses on S-SMART++.

7.1 BST

In Chapter 3 we discuss the NoC simulators BookSim and Garnet, and the HDL implementation OpenSMART. There exist many other open-source alternatives, such as [Papamichael2012; Abad2012; Catania2015; Chen2016a; Norollah2018]. While these tools provide very diverse and relevant functionalities, as far as we know none of them supports detailed bypass router models, at least in their public version.

In some cases, such NoC simulators are integrated with other tools in order to evaluate shared-memory systems. Most simulation platforms at this level tend to be cycle-accurate to faithfully model the processing cores and the memory hierarchy. However, most of such proposals make use of simple NoC architectures [Carlson2011; Sanchez2013] or have employed router architectures without bypass features [Binkert2011; Hsieh2012; Bakhoda2009; Lowe-Power2020].

The next subsections discuss alternative tools or methodologies to evaluate the effect of NoCs in real systems, which, in our opinion, is the biggest obstacle for their evaluation.

7.1.1) Simulation time of Full-System simulations

A fundamental aspect of evaluating shared-memory systems, especially CMPs, is their simulation time that in many cases is unaffordable because the simulators are commonly single-threaded. When parallelizing a time-driven simulator, typically, it is done following a coarse-grain partitioning, which is only applicable when the simulated system is large [Mohammad2017] or the scalability limit is acceptable. For example, TOPAZ [Abad2012], based on SICOSYS [Puente2002], has support for parallel executions and is integrated in gem5. However, the execution of gem5 is sequential and only the execution of TOPAZ is parallel.

Based on our own experience, this kind of simulation may take months to complete the execution of PARSEC benchmarks [Bienia2011] using large input-sets when run in 64-core CMPs with detailed models. Nowadays, CMPs already exceed such core counts and it is critical to evaluate NoCs with several hundreds or thousands of nodes for near-future CMPs. The concern is that simulation times will continue increasing with the number of cores, not only due to the system size but also because of the size of the input workload required by the benchmarks. These workloads also have to grow if the goal is to resemble the execution of the benchmarks in a real system. In this sense, even the largest state-of-the-art industrial FPGA platforms cannot simulate large multi-cores with tens of high performance cores, accelerators, a complex NoC and high bandwidth memory controllers. A possible solution to this issue might be the use of FPGAs to accelerate the simulation by implementing certain components of the system in them [Angepat2014], but this depends on the exploitable parallelism of the simulator.

The utilization of execution traces is a common solution to mitigate the previous problem. For example, elastic traces in gem5 [Jagtap2016] capture loads and stores of detailed out-of-order CPUs to then replay them, abstracting the core details and achieving speed-ups up to 8x over execution-driven simulations while providing low errors, around 3% in relative terms. As an alternative, SynFull [Badr2014] proposes to abstract the core details and the memory hierarchy to simulate or emulate only the NoC with automatic generated finite state machines that represents the traffic produced by the cores to the NoC. Mocktails [Badr2020] follows the idea of SynFull to generate memory traffic models in heterogeneous systems based on the temporal and spatial distribution of memory accesses. It focuses on generating black-box models of intellectual property (IP) blocks, of which the industry does not provide memory traces. Thus, companies may use Mocktails to provide synthetic models, which hide relevant information of their IP blocks, to support academic research. Nevertheless, in both cases, they first require a full system simulation or real execution to generate the trace; the size of the traces is usually in the order of some GBs; and they lack flexibility, i.e., they are typically accurate when modeling the same system configuration used during the capture of the trace.

BarrierPoint [Carlson2014] is a promising methodology. The methodology is based on SimPoint [Hamerly2005], which draws from the premise that applications can be divided in phases that are repeated multiple times during an execution. Thus, simulating just once each of the phases, it is possible to reconstruct the whole execution. BarrierPoint extends SimPoint to parallel applications, which introduce a new dimension to the problem due to the variability that each concurrent thread can exhibit. This is addressed in BarrierPoint by leveraging synchronization barriers commonly used in parallel programming to separate the application phases. In general, this methodology solves the lack of flexibility of traces. Moreover, this methodology has been successfully applied when using a real system to characterize the application and obtain the BarrierPoints, and use that information directly in the simulated system [Ferreron2017; TairumCruz2018]. The information from the real system comes from hardware counters and dynamic binary instrumentation, being accurate even when employing different ISAs between the real and the simulated system. However, it is uncertain that one can use this methodology in a real system with fewer cores than the target one, as the use of thread oversubscription might alter the temporal characteristics of the metrics.

7.1.2) Simulation of large-scale parallel applications

For larger deployments, subsystem simulators are common tools that allow obtaining performance predictions and assist computer architects in designing specific parts of HPC systems. Mubarak et al.[Mubarak2017], for example, propose CODES, a fast and flexible simulation framework that models large-scale state-of-the-art torus and dragonfly networks. Compared to this, our work focuses on a detailed NoC model that should be included in a state-of-the-art simulator like gem5, oriented to evaluate the nodes (servers) composing such big systems.

In this direction, prior work proposed simulation methodologies to evaluate the performance of large-scale parallel applications on distributed systems [Zheng2010; Denzel2008; Grobelny2007]. Most proposals use analytical models to estimate node performance (no cycle-accurate models are used) or system software interactions.

SST [Rodrigues2011] is a multi-scale simulator often used in combination with other simulators to model distributed applications. In BE-SST [Ramaswamy2018], authors combine SST with coarse-grained behavioral emulation models abstracting from micro-architectural details in favor of simulation speed. Other implementations integrate SST with a highly accurate simulator but require too costly full system simulations to produce a wide set of experiments [Hsieh2012; Ramaswamy2018]. Since SST is compatible with gem5, it could benefit from the contributions of BST.

The MUlti-level SimulAtion methodology (MUSA) enables fast and accurate performance estimations in scenarios with several thousands of cores [Grass2016; Gomez2019]. MUSA takes into account inter-node communication, node-level architecture, and system software interactions. MUSA combines sampling techniques with different simulators based on analytical models and cycle-accurate traces. The NoC model in MUSA is a simple multi-bus without any notion of the router architecture. Integrating BookSim in MUSA is an interesting future work.

7.1.3) Analytical models

With the same objective, application specific analytical models [Kerbyson2001; Nowatzki2013; Marjanovic2014] use a small set of parameters to predict performance for a single application on large systems. Once those models are created and validated, they are able to accurately predict performance with negligible compute and time cost. For example, Vaish et al. [Vaish2016] address three NoC design problems, which are memory controller placement, resource allocation in heterogeneous networks, and their combination. The main downside of these models is that they have little flexibility; any significant change in the application or hardware architecture requires the model to be updated, refined, and validated again. Our methodology focuses on hardware micro-architectural exploration and iterative fast co-design; new features can be tested on all applications at the moment they are included in the simulator.

7.2 NEBB

Chapter 4 targets improving the implementation of single-hop bypass NoCs defined initially in [Kumar2007]. In this section we review other works that also improve the original architecture and alternative solutions as well.

Lookahead routing [Galles1997], originally introduced to reduce the latency of routers in system area networks, is key to set-up the bypass in NoCs with bypass routers. To our knowledge, the first publication that presents a NoC architecture with bypass paths, to reduce latency without increasing the radix of the routers, is Express Virtual Channels (EVC, [Kumar2007a]). EVC is based on dedicating a set of VCs to use what the authors call the express pipeline of routers, which is equivalent to the bypass path of the routers described in Chapter 2. The dedicated VCs, or express VCs, form paths k-hops away (with $k > 1$) so when a packet acquires one of these VCs it can take the bypass in the intermediate routers. However, this proposal presents some issues. First, it requires many VCs, specially the dynamic version which have the best latency, to conform the express VCs. Second, the architecture complicates the buffer management as credits has to be sent to the upstream routers k-hops away. And third, bypass is only supported when traveling along a dimension, to avoid conflicts between packets acquiring overlapping express VCs at the same time.

In the same year, the router architecture presented in Section 2.1 was published [Kumar2007]. This architecture solves all the problems of EVC by using only information about the occupation of adjacent routers and without partitioning the VC set. This is possible due to the conditions defined to use the bypass. However, as we have already mentioned multiple times, they are unnecessarily conservative, requiring multiple VCs to exploit the throughput of the topology, which complicates the design and increases area and power.

Token Flow Control (TFC, [Kumar2008]) is another work from the same authors that, like EVC, communicates information, denoted as tokens, about the availability of resources among nodes in a neighborhood. The objective of the mechanism is the improvement of the bypass utilization by choosing low congested paths, exploiting path diversity with adaptive routing. In that work, the authors introduce the LookAhead Conflict Check unit (LA-CC, equivalent to LA-Arb) which modifies two of the bypass conditions originally presented that discard packet bypass when an LA conflicts with a flit or another LA. NEBB only takes advantage of the LA-CC idea because our objective was to make the conditions as simple as possible while providing compatibility with other mechanisms designed for traditional routers, such as FBFC in tori (Section 4.3). However, adapting NEBB to Token Flow Control could be interesting to exploit its adaptive routing, specially for adverse traffic.

Based on TFC, SWIFT [Krishna2010] presents a router architecture that concentrates on using low-swing links to implement the crossbars and links, improving energy consumption. The proposed conditions to use the bypass in NEBB are totally independent of the signaling technology, therefore it can be combined with NEBB.

ShortPath [Psarras2016] proposes an alternative pipeline organization for bypass routers. It focuses on enhancing performance by removing the speculative allocation of this type of router. With non-speculative allocation, flits may bypass the first allocation stage in case of losing the second one. This minimizes the time spent by each flit in a router by bypassing part of the allocation when the bypass of the entire router is not possible. As far as we know, ShortPath uses WH because it performs SA for body flits, and allows the storage of multiple packets in the same input VC. Combining ShortPath with NEBB is an interesting idea for future work to take the most of both.

7.2.2) Ordered message NoCs

As mentioned in Section 2.1.3.B, giving priority to LAs over local flits can break the message sequence order. Giving priority to local flits do not guarantee the message order as it can be already broken if the VC allocation policy is not designed accordingly, even in traditional routers. The cache coherence protocol of distributed directory-based implementations usually does not require message order. Therefore, giving priority to LAs is typically preferable, at least while it does not cause starvation.

SCORPIO [Daya2014] guarantees the message order if the protocol requires it, which is very common in snoopy protocols. SCORPIO basically consists of two subnetworks: the main network that transports the messages in any order, like any common NoC; and a notification network in charge of informing that there are messages in-flight towards the Network Interface Controllers (NICs). The notification network has a delimited maximum latency due to the micro-architecture design, guaranteeing that all the destination nodes of a message will receive the ordering information in time. Then, the NIC is in charge of reordering the messages at the destination using the previous information. SCORPIO is compatible with ARM's Advanced Microcontroller Bus Architecture (AMBA). The router architecture of the main network is based on the single-hop bypass architecture described in Section 2.1.1 with the addition of broadcast support. The combination of NEBB with a reordering mechanism such as SCORPIO remains as future work.

7.2.3) Hybrid flow controls

Finally, the *Hybrid* version of NEBB combines two flow controls, WH and VCT. Whole Packet Forwarding (WPF, [Ma2012]) applies packet-based flow control in a WH network. In this case, they use it to relax VC re-allocation requirements in deadlock-free fully adaptive routing NoCs, without considering bypass. Another work that combines two different types of flow control is [Jafri2010], which combines bufferless back-pressure with the typical buffered one, to minimize buffer power consumption at low loads and maximize performance at medium-high loads, but again, without considering bypass.

7.3 SMART++

Chapters 2 and 3 already described the original SMART proposal [Krishna2013] and its HDL implementation OpenSMART [Kwon2017]. In this section we briefly mention other related works and alternative topologies that has been proposed in the context of NoCs to reduce the number of hops done by packets.

7.3.1) SMART related works

As explained in 2.2.3.C, the key issue of SMART_2D is the complexity that introduces the links to propagate SSRs, which grows quadratically with HPC_{Max}. In [Chen2016], the authors propose a dedicated network to propagate SSRs, replacing the SSR broadcast wires and the complex allocators required by SMART. This is a good solution to reduce the wire overhead and energy consumption but it introduces an extra pipeline stage to arbitrate between SSRs in the SSR network. SHARP (Smart Hop Arbitration Request

Propagation) [Asgarieh2019] is an alternative solution that does not add an extra pipeline stage, besides eliminating the quadratic SSR arbitration. It also eliminates the possibility of false negative SA-G allocations by only propagating SSRs from the previous routers that win SA-G.

Besides the previous two works that could be adapted directly to SMART++, there are a variety of interesting works that uses SMART. For example, WiSMART (wireless-enabled SMART [Duraisamy2017]) is a hybrid NoC that combines SMART and a wireless NoC (WiNoC). This combination allows operating at high frequencies independently of HPC_{max}, using wireless communication for long distances. Another example is [Yang2017], which uses task mapping techniques to reduce conflicts between packets in SMART. The techniques focus on communication contention, rather than communication distance, as contention degrades bypass utilization. Finally, an analytical model of SMART is presented in [Bhattacharya2017] to accelerate simulations, achieving reductions of two orders of magnitude with respect to cycle-accurate simulators.

7.3.2) Low-diameter topologies

Designing low-diameter topologies is a common solution to reduce the average distance between nodes in networks. They are implemented using high-radix routers. Some of the most traditional examples of these topologies are concentrated mesh, flattened butterfly [Kim2007] or express cube topologies [Grot2009a]. A recent topology proposal is the Slim NoC (SN) [Besta2018], a low diameter NoC design that minimizes router radix for a given node count based on Moore graphs and non-prime finite fields. The main disadvantage of these four topologies is that they increase the router radix, which does not only increment the number of ports including their respective components (buffers, VC registers, control logic, etc), but also increases the complexity of the allocators and crossbars. The case of using concentration in a mesh, or any other topology, may be a double-edge sword. It increases the number of injection/ejection ports because there are more nodes connected to the routers, but it reduces the total number of routers with respect to an equivalent topology without concentration. However, the decisive factor may be the bandwidth reduction, which is divided by the concentration factor. Either way, SMART++ should be compatible with these topologies without further adaptation but the combination of them does not make sense, except for concentrated meshes with moderate/large distances. Another option is the combination of SMART++ with topologies in between meshes and flattened butterflies, i.e., meshes with *ruche* channels [Ou2020]. Instead of interconnecting adjacent routers of a mesh, *ruche* channels interconnect routers that are at a distance determined by the *ruche factor*. The case of the flattened butterfly is an extreme case of a mesh with *ruche* channels that combines all the possible *ruche factors*.

7.4 S-SMART++

In the previous section we already mentioned the related work of SMART++ and therefore of S-SMART++. However, returning to SHARP [Asgarieh2019], it introduces the concept of SSR propagation which eliminates false negatives in SA-G. Instead of broadcasting the SSR, SHARP routers process all the received SSRs and only forward one winning SSR per output port. Therefore, using SHARP allows intermediate multi-hop routers know

in advance when they will be the destination of a premature stop of a multi-hop. In combination with Speculative-SMART++, this information can be leveraged to generate spec-SSRs in intermediate routers where SA-G fails. If the spec-SSR succeeds in the next cycle, the packet will only lose 1 cycle in intermediate routers instead of 3 like in the current implementation of S-SMART++, which only works when packets traverse the whole multi-hop.

Chapter 8

Conclusions

NoCs play a key role in the performance and area/power budgets of many-core processors. They are part of the memory sub-system and add a new source of non-uniform latency in memory operations. Therefore, designing cost-efficient networks with minimal latency is critical. However, the task is becoming more and more difficult with the continuous increase in the number of cores found in CMPs, as it is the number of nodes to interconnect. Mesh-based NoCs are in use in many current multi- and many-core processors. However, the latency of meshes is significant for current core counts, and their scalability is limited. Alternative low-latency topologies using high-radix routers have been proposed but their implementation is costly.

NoCs in CMPs usually have to deal with low load, so under these conditions there are two main factors that define the latency: the delay of routers, and the distance between nodes. Bypass routers were proposed to minimize the latency in mesh-like topologies while taking advantage of its relative low cost. There are two types of bypass routers: single-hop bypass routers focus on reducing the router delay; and multi-hop bypass routers focus on reducing the effective distance between nodes.

Summary of contributions

This thesis focuses on the improvement of bypass routers in terms of efficiency and performance.

In Chapter 3 we describe BST (Bypass Simulation Toolset), a set of tools to develop and design NoC with bypass support. BST published in [Perez2020], has been made available to the community. The main goal of BST is to fill the gap found in open-source NoC simulators, that implement out-dated traditional router models or do not model bypass routers with cycle accuracy. BST comprises four parts: an updated version of BookSim with models of single-hop and multi-hop bypass routers, including NEBB, SMART++ and S-SMART++; an updated version of OpenSMART with SMART++ and S-SMART++; an API to easily integrate BookSim in Full-System simulators or any other kind of system model; and a set of scripts to launch bundles of simulations, generate plots, analyze and debug the network models. BST is the main tool used to evaluate the other contributions of the thesis: NEBB, SMART++ and S-SMART++.

In Chapter 4 we analyze the packet-interleaving problem that appears when blindly implementing bypass in traditional routers, focusing on single-hop bypass. We show that previous proposals set strict conditions to use the bypass, leading to low efficiency. In addition, they require multiple VCs to take advantage of the bypass paths and the band-

width of the network. Instead, we propose an implementation called NEBB (Non-Empty Buffer Bypass), that supports bypassing blocked buffers that are not empty. NEBB minimizes the cases in which the bypass cannot be used, maximizing its utilization, and not requiring VCs. Therefore, the resulting designs are simpler, allow higher clock frequency and consume less power. We propose three versions of NEBB: NEBB-WH for flit-based flow controls; NEBB-VCT for packet-based flow controls; and NEBB-Hybrid a combination of the other two that maximizes the opportunities of bypass flits. NEBB-Hybrid is able to match the performance of the baseline without requiring VCs and with half the total buffer space.

In Chapter 5, we point out a limitation of SMART, and propose SMART++ to overcome this limitation by extending the analysis of potential packet-interleaving issues to multi-hop bypass routers. SMART only forwards packets when destination VCs are empty, requiring the use of many VCs to achieve high throughput and exploit multi-hop bypasses. However, this results in overly complex and power-hungry designs. As a solution to the limitation of SMART, we propose SMART++, a multi-hop bypass architecture that combines: SMART, multi-packet buffers, NEBB, and packet-by-packet arbitration. Multi-packet buffers allow the forwarding of packets when there is available space in the downstream router. Adding, NEBB removes the requirement for empty VCs to bypass single-flit packets. Finally, packet-by-packet arbitration enables multi-flit packets to take the bypass too. The combination of these mechanisms allows the design of cheap configurations with one or a few deep buffers instead of many small VCs. Thus, SMART++ obtains practically the same performance of SMART even without using VCs, which translates in area and power reductions due to the simpler logic required. The experimentation in an FPGA shows reductions of up to 5.49× in area and 4.99× in dynamic power.

In Chapter 6, we present S-SMART++, which combines the fundamentals of single- and multi-hop bypass to reduce both the router delay and the effective distance between nodes. S-SMART++ uses speculative requests to configure multi-hop paths that may left unused because packets may not complete their previous multi-hop. Standard single- and multi-hop architectures already employ speculation to use the bypass paths, but S-SMART++ goes one step further because it can configure multi-hops even when packets can not complete the previous multi-hop. S-SMART++ is comparable in terms of zero-load latency to SMART_2D for the same HPC_{Max} in 2D-Meshes, but without the high costs of broadcasting SSRs in both dimensions. From our point of view, S-SMART++ offers a more important improvement, that is the reduction of the effect of HPC_{Max} in the base latency of multi-hop bypass NoCs. By depending less on HPC_{Max}, there is more room for tuning the frequency of operation, the complexity of SA-G (input phase) and other general parameters to balance the design. In terms of cost, the results show increments with respect to SMART++ of up to 31% in area and 18% in power. However, being true that S-SMART++ introduces overhead, it is also true that it does not perfectly fit the base design of OpenSMART.

Future work

Chapter 7 presents related work, and it outlines several ideas of potential future work, combining the proposals in this thesis with previous works. Additionally, we present next some ideas to improve or extend the contributions of this thesis in the future.

Regarding BST (Chapter 4) we would like to:

- Improve the **simulation speed** of BookSim, specially in Full-System simulations. BookSim is a time driven simulator, so in every cycle it goes through the set of

NoC components checking whether they have packets/flits to evaluate or not. In FS simulators like gem5, which is event driven, this can introduce a significant overhead if the traffic offered to the NoC is low. A possible solution might be the parallelization of BookSim by dividing the NoC into multiple parts. Another possible solution might be the conversion of BookSim into an event driven simulator, but this is a complex task given its current architecture.

- Improve the implementation of OpenSMART with the following features to obtain more accurate evaluations: *free/avail_vc* signaling for VS as an alternative to VA with credits; Multi-flit support; and SMART_2D including buffer bypass. In this regard, implementing actual VLSI designs and integrate them it into a RISC-V open source many-core processor such as [Balkind2016] would be very valuable.

With respect to S-SMART++, it would be interesting to evaluate S-SMART++ in concentrated meshes with *ruche* links [Ou2020] to analyze the cost-benefit ratio of HPC_{Max}, the radix of the routers, and the network size. Lastly, we would like to improve the implementation of S-SMART++ in OpenSMART replacing router bypass with buffer bypass.

Another possible topic to address is the application of bypass routers to other types of systems. For example, single-hop bypass routers might help to reduce the overall power consumption in system area networks. In GPUs or hardware accelerators with high bandwidth demand, applying single-hop bypass might be used to: reduce the latency per hop in low-distance topologies such as the FBFLY; or apply QoS over the bypass paths to prioritize certain packet classes.

Finally, it would be interesting to get into the subject of wireless NoCs to study if SMART++ and S-SMART++ can improve proposals such as WiSMART [Duraisamy2017]. This is far from our field of expertise, but applying some insight from the bypass restrictions of SMART++ should be possible. The same applies to the speculative-SSRs of S-SMART++ that might reduce the latency of the multi-hop preparation and traversal, which is 4 cycles in WiSMART.

Publications

The main research described in this thesis has been presented in the following publications:

- Iván Pérez, Enrique Vallejo, and Ramón Beivide. "Improving the efficiency of router bypass". In: *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. Poster. Oct. 2018

- Iván Pérez, Enrique Vallejo, and Ramón Beivide. "Efficient Router Bypass via Hybrid Flow Control". In: *11th International Workshop on Network on Chip Architectures (NoCArc)*. Oct. 2018, pp. 1–6. DOI: 10.1109/NOCARC.2018.8541147

- Iván Pérez, Enrique Vallejo, and Ramón Beivide. "SMART++: Reducing Cost and Improving Efficiency of Multi-hop Bypass in NoC Routers". In: *International Symposium on Networks-on-Chip (NOCS)*. 2019, 5:1–5:8. ISBN: 978-1-4503-6700-4. DOI: 10.1145/3313231.3352364

- Iván Pérez et al. "BST: A BookSim-Based Toolset to Simulate NoCs with Single- and Multi-Hop Bypass". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Boston, MA, USA: IEEE, Aug. 2020, pp. 47–57. ISBN: 978-1-72814-798-7. DOI: 10.1109/ISPASS48437.2020.00015

- Iván Pérez, Enrique Vallejo, and Ramón Beivide. "Efficient bypass in mesh and torus NoCs". en. In: *Journal of Systems Architecture* 108 (Sept. 2020), p. 101832. ISSN: 13837621. DOI: 10.1016/j.sysarc.2020.101832
- Iván Pérez, Enrique Vallejo, and Ramón Beivide. "S-SMART++: a Low-Latency NoC Leveraging Speculative Bypass Requests". In: *IEEE Transactions on Computers (second round of review)* (2021)

Bibliography

[Abad2012] P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente, and J. Gregorio. "TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers". In: *IEEE/ACM Sixth International Symposium on Networks-on-Chip*. May 2012, pp. 99–106. DOI: 10.1109/NOCS.2012.19.

[Alazemi2018] Fawaz Alazemi, Arash AziziMazreah, Bella Bose, and Lizhong Chen. "Routerless Network-on-Chip". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Vienna, Feb. 2018, pp. 492–503. ISBN: 978-1-5386-3659-6. DOI: 10.1109/HPCA.2018.00049.

[Angepat2014] Hari Angepat, Derek Chiou, Eric S Chung, and James C Hoe. *FPGA-accelerated simulation of computer systems*. OCLC: 1205371786. 2014. ISBN: 978-1-62705-214-6.

[ARM2018] ARM. *The Arm CoreLink CMN-600 Coherent Mesh Network*. 2018. URL: https://www.arm.com/products/silicon-ip-system/corelink-interconnect/cmn-600 (visited on 01/15/2021).

[Asgarieh2019] Yashar Asgarieh and Bill Lin. "Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs". In: *ACM Transactions on Design Automation of Electronic Systems* 24.6 (Nov. 2019), pp. 1–25. ISSN: 1084-4309, 1557-7309. DOI: 10.1145/3356235.

[Badr2014] Mario Badr and Natalie D. Enright Jerger. "SynFull: Synthetic traffic models capturing cache coherent behaviour". In: *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, pp. 109–120. DOI: 10.1109/ISCA.2014.6853236.

[Badr2020] M. Badr, C. Delconte, I. Edo, R. Jagtap, M. Andreozzi, and N. E. Jerger. "Mocktails: Capturing the memory behaviour of proprietary mobile architectures". In: *ACM/IEEE 47th annual international symposium on computer architecture (ISCA)*. 2020, pp. 460–472. DOI: 10.1109/ISCA45697.2020.00046.

[Bakhoda2009] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. "Analyzing CUDA workloads using a detailed GPU simulator". In: *IEEE International Symposium on Performance Analysis of Systems and Software*. tex.ids: bakhoda2009. Boston, MA, USA, Apr. 2009, pp. 163–174. ISBN: 978-1-4244-4184-6. DOI: 10.1109/ISPASS.2009.4919648.

[Balkind2016] Jonathan Balkind et al. "OpenPiton: An Open Source Manycore Research Framework". In: *ACM SIGPLAN Notices* 51.4 (June 2016), pp. 217–232. ISSN: 0362-1340, 1558-1160. DOI: 10.1145/2954679.2872414.

[Bell2008] Shane Bell et al. "Tile64-processor: A 64-core SoC with mesh interconnect". In: *IEEE international solid-state circuits conference-digest of technical papers*. tex.organization: IEEE. 2008, pp. 88–598.

[Besta2014] Maciej Besta and Torsten Hoefler. "Slim Fly: A Cost Effective Low-Diameter Network Topology". In: *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. New Orleans, LA, USA, Nov. 2014, pp. 348–359. ISBN: 978-1-4799-5500-8. DOI: 10.1109/SC.2014.34.

[Besta2018] Maciej Besta, Syed Minhaj Hassan, Sudhakar Yalamanchili, Rachata Ausavarungnirun, Onur Mutlu, and Torsten Hoefler. "Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability". In: *ACM SIGPLAN Notices* 53.2 (Nov. 2018), pp. 43–55. ISSN: 0362-1340, 1558-1160. DOI: 10.1145/3296957.3177158.

[Bharadwaj2020] Srikant Bharadwaj, Jieming Yin, Bradford Beckmann, and Tushar Krishna. "Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling". In: *ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA, July 2020, pp. 1–6. ISBN: 978-1-72811-085-1. DOI: 10.1109/DAC18072.2020.9218539.

[Bhattacharya2017] Debajit Bhattacharya and Niraj K. Jha. "Analytical Modeling of the SMART NoC". In: *IEEE TMSCS* (2017).

[Bienia2008] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. "The PARSEC benchmark suite: Characterization and architectural implications". In: *17th international conference on Parallel architectures and compilation techniques*. 2008, pp. 72–81.

[Bienia2011] Christian Bienia. "Benchmarking modern multiprocessors". PhD thesis. Princeton University, Jan. 2011.

[Binkert2011] Nathan Binkert et al. "The gem5 simulator". In: *ACM SIGARCH Computer Architecture News* 39.2 (May 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718.

[Bland2009] Arthur S Bland, Ricky A Kendall, Douglas B Kothe, James H Rogers, and Galen M Shipman. "Jaguar: The world's most powerful computer". In: *Memory (TB)* 300.62 (2009), p. 362.

[Bohnenstiehl2017] Brent Bohnenstiehl, Aaron Stillmaker, Jon J. Pimentel, Timothy Andreas, Bin Liu, Anh T. Tran, Emmanuel Adeagbo, and Bevan M. Baas. "KiloCore: A 32-nm 1000-Processor Computational Array". In: *IEEE Journal of Solid-State Circuits* 52.4 (Apr. 2017), pp. 891–902. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2016.2638459.

[Butenhof1997] David R Butenhof. *Programming with POSIX threads*. 1997.

[Carlson2011] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation". In: *International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, 52:1–52:12.

[Carlson2014] Trevor E Carlson, Wim Heirman, Kenzo Van Craeynest, and Lieven Eeckhout. "Barrierpoint: Sampled simulation of multi-threaded applications". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2014, pp. 2–12.

[Carrion1997] C. Carrion, R. Beivide, J. A. Gregorio, and F. Vallejo. "A flow control mechanism to avoid message deadlock in k-ary n-cube networks". In: *Fourth International Conference on High-Performance Computing*. Dec. 1997, pp. 322–329. DOI: 10.1109/HIPC.1997.634510.

[Catania2015] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. "Noxim: An open, extensible and cycle-accurate network on chip simulator". In: *IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. July 2015, pp. 162–163. DOI: 10.1109/ASAP.2015.7245728.

[Chandra2001] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. 2001.

[ChangkyuKim2003] Changkyu Kim, D. Burger, and S.W. Keckler. "Nonuniform cache architectures for wire-delay dominated on-chip caches". In: *IEEE Micro* 23.6 (Nov. 2003), pp. 99–107. ISSN: 0272-1732. DOI: 10.1109/MM.2003.1261393.

[Chapman2008] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. Vol. 10. 2008.

[Chen2011] Lizhong Chen, Ruisheng Wang, and Timothy M. Pinkston. "Critical Bubble Scheme: An Efficient Implementation of Globally Aware Network Flow Control". In: *IEEE International Parallel & Distributed Processing Symposium*. Anchorage, AK, USA, May 2011, pp. 592–603. ISBN: 978-1-61284-372-8. DOI: 10.1109/IPDPS.2011.63.

[Chen2016] X. Chen and N. K. Jha. "Reducing Wire and Energy Overheads of the SMART NoC Using a Setup Request Network". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.10 (Oct. 2016), pp. 3013–3026. DOI: 10.1109/TVLSI.2016.2538284.

[Chen2016a] Lizhong Chen, Di Zhu, Massoud Pedram, and Timothy M. Pinkston. "Simulation of NoC power-gating: Requirements, optimizations, and the Agate simulator". In: *Journal of Parallel and Distributed Computing* 95 (2016), pp. 69–78. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2016.03.006.

[Chrysos2014] George Chrysos. "Intel® xeon Phi™ coprocessor-the architecture". In: *Intel Whitepaper* 176 (2014), p. 43.

[Dally1986] William J. Dally and Charles L. Seitz. "The torus routing chip". In: *Distributed Computing* 1.4 (Dec. 1986), pp. 187–196. ISSN: 0178-2770, 1432-0452. DOI: 10.1007/BF01660031.

[Dally1987] Dally and Seitz. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks". In: *IEEE Transactions on Computers* C-36.5 (May 1987), pp. 547–553. ISSN: 0018-9340. DOI: 10.1109/TC.1987.1676939.

[Dally2003] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA, 2003. ISBN: 0-12-200751-4.

[Das2018] Abhijit Das, Sarath Babu, John Jose, Sangeetha Jose, and Maurizio Palesi. "Critical Packet Prioritisation by Slack-Aware Re-Routing in On-Chip Networks". In: *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. Turin, Oct. 2018, pp. 1–8. ISBN: 978-1-5386-4893-3. DOI: 10.1109/NOCS.2018.8512164.

[Daya2014] Bhavya K. Daya, Chia-Hsin Owen Chen, Suvinay Subramanian, Woo-Cheol Kwon, Sunghyun Park, Tushar Krishna, Jim Holt, Anantha P. Chandrakasan, and Li-Shiuan Peh. "SCORPIO: A 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering". In: *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. Minneapolis, MN, USA, June 2014, pp. 25–36. ISBN: 978-1-4799-4394-4. DOI: 10.1109/ISCA.2014.6853232.

[Dennard1974] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.1974.1050511.

[Denzel2008] Wolfgang E. Denzel, Jian Li, Peter Walker, and Yuho Jin. "A Framework for End-to-end Simulation of High-performance Computing Systems". In: *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. event-place: Marseille, France. 2008, 21:1–21:10. ISBN: 978-963-9799-20-2.

[Dimitrakopoulos2015] Giorgos Dimitrakopoulos, Anastasios Psarras, and Ioannis Seitanidis. *Microarchitecture of Network-on-Chip Routers*. 2015.

[Duato2003] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. 2003.

[Duraisamy2017] Karthi Duraisamy and Partha Pratim Pande. "Enabling High-Performance SMART NoC Architectures Using On-Chip Wireless Links". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.12 (Dec. 2017), pp. 3495–3508. DOI: 10.1109/TVLSI.2017.2748884.

[Duran2011] Alejandro Durán, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. "OmpSs: A Proposal for Programming Heterogeneous Multi-core Architectures". In: *Parallel Processing Letters* 21.2 (2011). Publisher: World Scientific, pp. 173–193.

[Ferreron2017] Alexandra Ferrerón, Radhika Jagtap, Sascha Bischoff, and Roxana Ruşitoru. "Crossing the architectural barrier: Evaluating representative regions of parallel HPC applications". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2017, pp. 109–120.

[Flynn1972] Michael J. Flynn. "Some Computer Organizations and Their Effectiveness". In: *IEEE Transactions on Computers* C-21.9 (Sept. 1972), pp. 948–960. ISSN: 0018-9340. DOI: 10.1109/TC.1972.5009071.

[Fu2016] Haohuan Fu et al. "The Sunway TaihuLight supercomputer: system and applications". In: *Science China Information Sciences* 59.7 (June 2016), p. 072001. ISSN: 1869-1919. DOI: 10.1007/s11432-016-5588-7.

[Galles1997] M. Galles. "Spider: a high-speed network interconnect". In: *IEEE Micro* 17.1 (Jan. 1997), pp. 34–39. ISSN: 0272-1732. DOI: 10.1109/40.566196.

[Gara2005] A. Gara et al. "Overview of the Blue Gene/L system architecture". In: *IBM Journal of Research and Development* 49.2.3 (Mar. 2005), pp. 195–212. ISSN: 0018-8646, 0018-8646. DOI: 10.1147/rd.492.0195.

[Gomez2019] Constantino Gómez, Francesc Martínez, Adrià Armejach, Miquel Moretó, Filippo Mantovani, and Marc Casas. "Design Space Exploration of Next-Generation HPC Machines". In: *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 54–65. DOI: 10.1109/IPDPS.2019.00017.

[Graham2008] Richard L. Graham and Galen Shipman. "MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives". In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by Alexey Lastovetsky, Tahar Kechadi, and Jack Dongarra. Vol. 5205. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg, 2008, pp. 130–140. ISBN: 978-3-540-87474-4. DOI: 10.1007/978-3-540-87475-1_21.

[Grass2016] Thomas Grass, César Allande, Adrià Armejach, Alejandro Rico, Eduard Ayguadé, Jesús Labarta, Mateo Valero, Marc Casas, and Miquel Moretó. "MUSA: a multi-level simulation approach for next-generation HPC machines". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2016, pp. 526–537. DOI: 10.1109/SC.2016.44.

[Gratz2006] Paul Gratz, Changkyu Kim, Robert McDonald, Stephen W. Keckler, and Doug Burger. "Implementation and Evaluation of On-Chip Network Architectures". In: *International Conference on Computer Design*. ISSN: 1063-6404. San Jose, CA, USA, Oct. 2006, pp. 477–484. ISBN: 978-0-7803-9706-4. DOI: 10.1109/ICCD.2006.4380859.

[Grobelny2007] Eric Grobelny, David Bueno, Ian Troxel, Alan D George, and Jeffrey S Vetter. "FASE: A framework for scalable performance prediction of HPC systems and applications". In: *Simulation* 83.10 (2007), pp. 721–745.

[Gropp1999] William Gropp, William D Gropp, Ewing Lusk, Anthony Skjellum, and Argonne Distinguished Fellow Emeritus Ewing Lusk. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. 1999.

[Grot2009a] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. "Express Cube Topologies for on-Chip Interconnects". In: *IEEE 15th International Symposium on High Performance Computer Architecture*. Raleigh, NC, USA, Feb. 2009, pp. 163–174. ISBN: 978-1-4244-2932-5. DOI: 10.1109/HPCA.2009.4798251.

[Hamerly2005] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. "Simpoint 3.0: Faster and more flexible program phase analysis". In: *Journal of Instruction Level Parallelism* 7.4 (2005), pp. 1–28.

[Haring2012] Ruud Haring et al. "The IBM Blue Gene/Q Compute Chip". In: *IEEE Micro* 32.2 (Mar. 2012), pp. 48–60. ISSN: 0272-1732. DOI: 10.1109/MM.2011.108.

[Hassan2013] Syed Minhaj Hassan and Sudhakar Yalamanchili. "Centralized buffer router: A low latency, low power router for high radix NoCs". In: *Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. 2013, pp. 1–8.

[Hassan2014] Syed Minhaj Hassan and Sudhakar Yalamanchili. "Bubble sharing: Area and energy efficient adaptive routers using centralized buffers". In: *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. 2014, pp. 119–126.

[Hennessy2019] John L. Hennessy. *Computer architecture: a quantitative approach*. Sixth edition. Cambridge, MA, 2019. ISBN: 978-0-12-811905-1.

[Hesse2015] Robert Hesse and Natalie Enright Jerger. "Improving DVFS in NoCs with Coherence Prediction". In: *9th International Symposium on Networks-on-Chip - NOCS '15*. Vancouver, BC, Canada, 2015, pp. 1–8. ISBN: 978-1-4503-3396-2. DOI: 10.1145/2786572.2786595.

[Hong2016] Byungchul Hong, Gwangsun Kim, Jung Ho Ahn, Yongkee Kwon, Hongsik Kim, and John Kim. "Accelerating Linked-list Traversal Through Near-Data Processing". In: *2016 International Conference on Parallel Architectures and Compilation*. Haifa Israel, Sept. 2016, pp. 113–124. ISBN: 978-1-4503-4121-9. DOI: 10.1145/2967938.2967958.

[Hoskote2007] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. "A 5-GHz Mesh Interconnect for a Teraflops Processor". In: *IEEE Micro* 27.5 (Sept. 2007), pp. 51–61. ISSN: 0272-1732. DOI: 10.1109/MM.2007.4378783.

[Hsieh2012] Mingyu Hsieh, Kevin Pedretti, Jie Meng, Ayse Coskun, Michael Levenhagen, and Arun Rodrigues. "SST + Gem5 = a Scalable Simulation Infrastructure for High Performance Computing". In: *5th International ICST Conference on Simulation Tools and Techniques*. SIMUTOOLS. 2012, pp. 196–201. ISBN: 978-1-4503-1510-4.

[Hunter2007] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.55.

[Jafri2010] S. A. R. Jafri, Y. J. Hong, M. Thottethodi, and T. N. Vijaykumar. "Adaptive Flow Control for Robust Performance and Energy". In: *International Symposium on Microarchitecture (Micro)*. ISSN: 1072-4451. 2010, pp. 433–444. DOI: 10.1109/MICRO.2010.48.

[Jagtap2016] Radhika Jagtap, Stephan Diestelhorst, Andreas Hansson, Matthias Jung, and Norbert When. "Exploring system performance using elastic traces: Fast, accurate and portable". In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. Agios Konstantinos, Samos Island, Greece, July 2016, pp. 96–105. ISBN: 978-1-5090-3076-7. DOI: 10.1109/SAMOS.2016.7818336.

[Jain2016] Nikhil Jain, Abhinav Bhatele, Sam White, Todd Gamblin, and Laxmikant V. Kale. "Evaluating HPC Networks via Simulation of Parallel Workloads". In: *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. Salt Lake City, UT, USA, Nov. 2016, pp. 154–165. ISBN: 978-1-4673-8815-3. DOI: 10.1109/SC.2016.13.

[Jerger2017] Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh. *On-Chip Networks, Second Edition*. Vol. 12. 3. 2017.

[Jiang2013] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. "A detailed and flexible cycle-accurate Network-on-Chip simulator". In: *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2013, pp. 86–96. DOI: 10.1109/ISPASS.2013.6557149.

[Jiang2015] Nan Jiang, Larry Dennison, and William J. Dally. “Network endpoint congestion control for fine-grained communication”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis on - SC ’15*. Austin, Texas, 2015, pp. 1–12. ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807600.

[Kahle2019] James A. Kahle, Jaime Moreno, and Dan Dreps. “2.1 Summit and Sierra: Designing AI/HPC Supercomputers”. In: *IEEE International Solid- State Circuits Conference - (ISSCC)*. San Francisco, CA, USA, Feb. 2019, pp. 42–43. ISBN: 978-1-5386-8531-0. DOI: 10.1109/ISSCC.2019.8662426.

[Kahng2009] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. “ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration”. In: *Design, Automation & Test in Europe Conference & Exhibition*. Nice, Apr. 2009, pp. 423–428. ISBN: 978-1-4244-3781-8. DOI: 10.1109/DATE.2009.5090700.

[Kahng2015] Andrew B. Kahng, Bill Lin, and Siddhartha Nath. “ORION3.0: A Comprehensive NoC Router Estimation Tool”. In: *IEEE Embedded Systems Letters* 7.2 (June 2015), pp. 41–45. ISSN: 1943-0663, 1943-0671. DOI: 10.1109/LES.2015.2402197.

[Kannan2015] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H. Loh. “Enabling interposer-based disintegration of multi-core processors”. In: *48th International Symposium on Microarchitecture - MICRO-48*. Waikiki, Hawaii, 2015, pp. 546–558. ISBN: 978-1-4503-4034-2. DOI: 10.1145/2830772.2830808.

[Kerbyson2001] Darren J Kerbyson, Henry J Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J Wasserman, and Mike Gittings. “Predictive performance and scalability modeling of a large-scale application”. In: *Supercomputing, ACM/IEEE 2001 Conference*. 2001, pp. 39–39.

[Kim2007] J. Kim, J. Balfour, and W. Dally. “Flattened Butterfly Topology for On-Chip Networks”. In: *International Symposium on Microarchitecture (MICRO)*. 2007, pp. 172–182. DOI: 10.1109/MICRO.2007.29.

[Krawezik2003] Géraud Krawezik. “Performance comparison of MPI and three openMP programming styles on shared memory multiprocessors”. In: *fifteenth annual ACM symposium on Parallel algorithms and architectures - SPAA ’03*. San Diego, California, USA, 2003, p. 118. ISBN: 978-1-58113-661-6. DOI: 10.1145/777412.777433.

[Krishna2010] Tushar Krishna, Jacob Postman, Christopher Edmonds, Li-Shiuan Peh, and Patrick Chiang. “SWIFT: A SWing-reduced interconnect for a Token-based Network-on-Chip in 90nm CMOS”. In: *2010 IEEE International Conference on Computer Design*. tex.ids: krishna2010. Amsterdam, Netherlands, Oct. 2010, pp. 439–446. ISBN: 978-1-4244-8936-7. DOI: 10.1109/ICCD.2010.5647666.

[Krishna2013] T. Krishna, Chia-Hsin Owen Chen, Woo Cheol Kwon, and Li-Shiuan Peh. “Breaking the on-chip latency barrier using SMART”. In: *International Symposium on High Performance Computer Architecture (HPCA)*. Feb. 2013, pp. 378–389. DOI: 10.1109/HPCA.2013.6522334.

[Krishna2017] Tushar Krishna. “GARNET2.0: A detailed on-chip network model inside a full-system simulator”. In: *gem5 workshop, ARM Research Summit*. 2017.

[Kumar2007] Amit Kumar, Partha Kunduz, AP Singhx, L-S Pehy, and NK Jhay. “A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS”. In: *25th International Conference on Computer Design (ICCD)*. Oct. 2007, pp. 63–70. DOI: 10.1109/ICCD.2007.4601881.

[Kumar2007a] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. “Express Virtual Channels: Towards the Ideal Interconnection Fabric”. In: *34th Annual International Symposium on Computer Architecture*. ISCA ’07. event-place: San Diego, California, USA. New York, NY, USA, 2007, pp. 150–161. ISBN: 978-1-59593-706-3. DOI: 10.1145/1250662.1250681.

[Kumar2008] Amit Kumar, Li-Shiuan Peh, and Niraj K. Jha. “Token Flow Control”. In: *International Symposium on Microarchitecture (MICRO)*. 2008, pp. 342–353. ISBN: 978-1-4244-2836-6.

[Kwon2017] H. Kwon and T. Krishna. “OpenSMART: Single-cycle multi-hop NoC generator in BSV and Chisel”. In: *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2017, pp. 195–204. DOI: 10.1109/ISPASS.2017.7975291.

[Li2009] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: *42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*. New York, New York, 2009, p. 469. ISBN: 978-1-60558-798-1. DOI: 10.1145/1669112.1669172.

[Li2016] Zimo Li, Joshua San Miguel, and Natalie Enright Jerger. “The runahead network-on-chip”. In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Barcelona, Spain, Mar. 2016, pp. 333–344. ISBN: 978-1-4673-9211-2. DOI: 10.1109/HPCA.2016.7446076.

[LizhongChen2013] Lizhong Chen and T. M. Pinkston. “Worm-Bubble Flow Control”. In: *IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. Shenzhen, Feb. 2013, pp. 366–377. ISBN: 978-1-4673-5587-2. DOI: 10.1109/HPCA.2013.6522333.

[Lowe-Power2020] Jason Lowe-Power et al. “The gem5 Simulator: Version 20.0+”. In: *arXiv:2007.03152 [cs]* (Sept. 2020). arXiv: 2007.03152.

[Ma2012] Sheng Ma, Natalie Enright Jerger, and Zhiying Wang. “Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip”. In: *International Symposium on High-Performance Computer Architecture (HPCA)*. 2012, pp. 1–12. ISBN: 978-1-4673-0827-4. DOI: 10.1109/HPCA.2012.6169049.

[Ma2015] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger. “Leaving One Slot Empty: Flit Bubble Flow Control for Torus Cache-Coherent NoCs”. In: *IEEE Transactions on Computers* 64.3 (Mar. 2015), pp. 763–777. DOI: 10.1109/TC.2013.2295523.

[Magnusson2002] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. “Simics: A full system simulation platform”. In: *Computer* 35.2 (Feb. 2002), pp. 50–58. ISSN: 00189162. DOI: 10.1109/2.982916.

[Marjanovic2014] Vladimir Marjanović, José Gracia, and Colin W Glass. “Performance modeling of the HPCG benchmark”. In: *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. 2014, pp. 172–192.

[Miyazaki2012] Hiroyuki Miyazaki, Yoshihiro Kusano, Naoki Shinjou, Fumiyoshi Shoji, Mitsuo Yokokawa, and Tadashi Watanabe. “Overview of the K computer system”. In: *Fujitsu Sci. Tech. J* 48.3 (2012), pp. 302–309.

[Mohammad2017] Alian Mohammad, Umur Darbaz, Gabor Dozsa, Stephan Diestelhorst, Daehoon Kim, and Nam Sung Kim. “dist-gem5: Distributed simulation of computer clusters”. In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Santa Rosa, CA, USA, Apr. 2017, pp. 153–162. ISBN: 978-1-5386-3890-3. DOI: 10.1109/ISPASS.2017.7975287.

[Moscibroda2009] Thomas Moscibroda and Onur Mutlu. “A Case for Bufferless Routing in On-chip Networks”. In: *International Symposium on Computer Architecture*. event-place: Austin, TX, USA. 2009. ISBN: 978-1-60558-526-0. DOI: 10.1145/1555754.1555781.

[Mubarak2012] Misbah Mubarak, Christopher D. Carothers, Robert Ross, and Philip Carns. “Modeling a Million-Node Dragonfly Network Using Massively Parallel Discrete-Event Simulation”. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. Salt Lake City, UT, Nov. 2012, pp. 366–376. ISBN: 978-0-7695-4956-9. DOI: 10.1109/SC.Companion.2012.56.

[Mubarak2017] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns. “Enabling Parallel Simulation of Large-Scale HPC Network Systems”. In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 28.1 (Jan. 2017), pp. 87–100. ISSN: 1045-9219.

[Mullins2004] R. Mullins, A. West, and S. Moore. "Low-latency virtual-channel routers for on-chip networks". In: *31st Annual International Symposium on Computer Architecture*. Munchen, Germany, 2004, pp. 188–197. ISBN: 978-0-7695-2143-5. DOI: 10.1109/ISCA.2004.1310774.

[Nagarajan2020] Vijay Nagarajan, Daniel J Sorin, Mark D Hill, and David Allen Wood. *A primer on memory consistency and cache coherence*. OCLC: 1141095013. 2020. ISBN: 978-1-68173-711-9.

[Nicopoulos2006] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers". In: *International Symposium on Microarchitecture*. ISSN: 1072-4451. 2006. DOI: 10.1109/MICRO.2006.50.

[Norollah2018] A. Norollah, D. Derafshi, H. Beitollahi, and A. Patooghy. "PAT-Noxim: A Precise Power Thermal Cycle-Accurate NoC Simulator". In: *International System-on-Chip Conference (SOCC)*. 2018, pp. 163–168. DOI: 10.1109/SOCC.2018.8618491.

[Nowatzki2013] Tony Nowatzki, Michael Ferris, Karthikeyan Sankaralingam, Cristian Estan, Nilay Vaish, and David Wood. "Optimization and Mathematical Modeling in Computer Architecture". In: *Synthesis Lectures on Computer Architecture* 8.4 (Sept. 2013), pp. 1–144. ISSN: 1935-3235, 1935-3243. DOI: 10.2200/S00531ED1V01Y201308CAC026.

[Olofsson2016] Andreas Olofsson. "Epiphany-V: A 1024 processor 64-bit RISC system-on-chip". In: *arXiv preprint arXiv:1610.01832* (2016).

[Ou2020] Yanghui Ou, Shady Agwa, and Christopher Batten. "Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology". In: *14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. Hamburg, Germany, Sept. 2020, pp. 1–8. ISBN: 978-1-72818-847-8. DOI: 10.1109/NOCS50636.2020.9241710.

[Papamichael2012] Michael K. Papamichael and James C. Hoe. "CONNECT: Re-examining Conventional Wisdom for Designing NoCs in the Context of FPGAs". In: *International Symposium on Field Programmable Gate Arrays (FPGA)*. 2012, pp. 37–46. ISBN: 978-1-4503-1155-7. DOI: 10.1145/2145694.2145703.

[Papamichael2015] Michael K. Papamichael, Cagla Cakir, Chen Suny Chia-Hsin, Owen Cheny, James C. Ho, Ken Mai, Li-Shiuan Pehy, and Vladimir Stojanovic. "DELPHI: a framework for RTL-based architecture design evaluation using DSENT models". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Philadelphia, PA, USA, Mar. 2015, pp. 11–20. ISBN: 978-1-4799-1957-4. DOI: 10.1109/ISPASS.2015.7095780.

[Parasar2019] Mayank Parasar and Tushar Krishna. “BINDU: Deadlock-freedom with One Bubble in the Network”. In: *International Symposium on Networks-on-Chip (NOCS)*. 2019, 3:1–3:8. ISBN: 978-1-4503-6700-4. DOI: 10.1145/3313231.3352359.

[Pellegrini2020] Andrea Pellegrini et al. “The Arm Neoverse N1 Platform: Building Blocks for the Next-Gen Cloud-to-Edge Infrastructure SoC”. In: *IEEE Micro* 40.2 (Mar. 2020), pp. 53–62. ISSN: 0272-1732, 1937-4143. DOI: 10.1109/MM.2020.2972222.

[Perez2018] Iván Pérez, Enrique Vallejo, and Ramón Beivide. “Efficient Router Bypass via Hybrid Flow Control”. In: *11th International Workshop on Network on Chip Architectures (NoCArc)*. Oct. 2018, pp. 1–6. DOI: 10.1109/NOCARC.2018.8541147.

[Perez2018a] Iván Pérez, Enrique Vallejo, and Ramón Beivide. “Improving the efficiency of router bypass”. In: *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. Poster. Oct. 2018.

[Perez2019] Iván Pérez, Enrique Vallejo, and Ramón Beivide. “SMART++: Reducing Cost and Improving Efficiency of Multi-hop Bypass in NoC Routers”. In: *International Symposium on Networks-on-Chip (NOCS)*. 2019, 5:1–5:8. ISBN: 978-1-4503-6700-4. DOI: 10.1145/3313231.3352364.

[Perez2020] Iván Pérez, Enrique Vallejo, Miquel Moreto, and Ramón Beivide. “BST: A BookSim-Based Toolset to Simulate NoCs with Single- and Multi-Hop Bypass”. In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Boston, MA, USA, Aug. 2020, pp. 47–57. ISBN: 978-1-72814-798-7. DOI: 10.1109/ISPASS48437.2020.00015.

[Perez2020a] Iván Pérez, Enrique Vallejo, and Ramón Beivide. “Efficient bypass in mesh and torus NoCs”. In: *Journal of Systems Architecture* 108 (Sept. 2020), p. 101832. ISSN: 13837621. DOI: 10.1016/j.sysarc.2020.101832.

[Perez2021] Iván Pérez, Enrique Vallejo, and Ramón Beivide. “S-SMART++: a Low-Latency NoC Leveraging Speculative Bypass Requests”. In: *IEEE Transactions on Computers (second round of review)* (2021).

[Psarras2016] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. “ShortPath: A Network-on-Chip Router with Fine-Grained Pipeline Bypassing”. In: *IEEE Trans. Comput. (TC)* 65.10 (Oct. 2016), pp. 3136–3147. DOI: 10.1109/TC.2016.2519916.

[Puente2002] V. Puente, J. A. Gregorio, and R. Beivide. “SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems”. In: *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*. 2002, pp. 15–22. DOI: 10.1109/EMPDP.2002.994207.

[QinhongZhang2015] Qinhong Zhang, Meng Zhou, Juan Chen, and Hao Yang. "A homogeneous many-core x86 processor full system framework based on NoC". In: *4th International Conference on Computer Science and Network Technology (ICCSNT)*. Harbin, China, Dec. 2015, pp. 794–797. ISBN: 978-1-4673-8173-4. DOI: 10.1109/ICCSNT.2015.7490861.

[Rabaey2003] Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*. 2nd ed. Prentice Hall electronics and VLSI series. Upper Saddle River, N.J, 2003. ISBN: 978-0-13-090996-1.

[Rahmani2010] Amir-Mohammad Rahmani, Khalid Latif, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. "Research and practices on 3D networks-on-chip architectures". In: *NORCHIP 2010*. 2010, pp. 1–6.

[Rajovic2016] Nikola Rajovic et al. "The Mont-Blanc Prototype: An Alternative Approach for HPC Systems". In: *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. Salt Lake City, UT, USA, Nov. 2016, pp. 444–455. ISBN: 978-1-4673-8815-3. DOI: 10.1109/SC.2016.37.

[Ramaswamy2018] Ajay Ramaswamy, Nalini Kumar, Aravind Neelakantan, Herman Lam, and Greg Stitt. "Scalable Behavioral Emulation of Extreme-Scale Systems Using Structural Simulation Toolkit". In: *International Conference on Parallel Processing (ICPP)*. 2018, 17:1–17:11.

[Requena2008] Crispín Requena, M.E. Gómez, Pedro López, and José Duato. "BPS: A bufferless switching technique for NoCs". In: (Feb. 2008).

[Rodrigues2011] A. F. Rodrigues et al. "The structural simulation toolkit". In: *ACM SIGMETRICS Performance Evaluation Review* 38.4 (Mar. 2011), pp. 37–42. ISSN: 0163-5999. DOI: 10.1145/1964218.1964225.

[Rosenband2004] D.L. Rosenband. "The ephemeral history register: flexible scheduling for rule-based designs". In: *Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE '04.* San Diego, CA, USA, 2004, pp. 189–198. ISBN: 978-0-7803-8509-2. DOI: 10.1109/MEMCOD.2004.1459853.

[Rovinski2019] Austin Rovinski et al. "Evaluating Celerity: A 16-nm 695 giga-RISC-V instructions/s manycore processor with synthesizable PLL". In: *IEEE Solid-State Circuits Letters* 2.12 (2019). Publisher: IEEE, pp. 289–292.

[Sadasivam2017] Satish Kumar Sadasivam, Brian W. Thompto, Ron Kalla, and William J. Starke. "IBM Power9 Processor Architecture". In: *IEEE Micro* 37.2 (Mar. 2017), pp. 40–51. ISSN: 0272-1732. DOI: 10.1109/MM.2017.40.

[Sanchez2013] Daniel Sanchez and Christos Kozyrakis. “ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems”. In: *International Symposium on Computer Architecture (ISCA)*. 2013, pp. 475–486. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485963.

[Sodani2016] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. “Knights landing: Second-generation Intel Xeon Phi product”. In: *IEEE Micro* 36.2 (2016). Publisher: IEEE, pp. 34–46.

[Strohmaier2020] Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. *Top500 supercomputer sites*. Nov. 2020. URL: www.top500.org (visited on 01/15/2021).

[Sun2012] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. “DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling”. In: *IEEE/ACM Sixth International Symposium on Networks-on-Chip*. Lyngby, Denmark, May 2012, pp. 201–210. ISBN: 978-1-4673-0973-8. DOI: 10.1109/NOCS.2012.31.

[TairumCruz2018] Miguel Tairum Cruz, Sascha Bischoff, and Roxana Rusitoru. “Shifting the Barrier: Extending the Boundaries of the Barrier-Point Methodology”. In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Belfast, Apr. 2018, pp. 120–122. ISBN: 978-1-5386-5010-3. DOI: 10.1109/ISPASS.2018.00023.

[Tamir1992] Yuval Tamir and Gregory L. Frazier. “Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches”. In: *IEEE Transactions on Computers* 41.6 (June 1992), pp. 725–737. ISSN: 0018-9340. DOI: 10.1109/12.144624.

[Towles2014] Brian Towles, J. P. Grossman, Brian Greskamp, and David E. Shaw. “Unifying On-chip and Inter-node Switching Within the Anton 2 Network”. In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. ISCA ’14. event-place: Minneapolis, Minnesota, USA. Piscataway, NJ, USA, 2014, pp. 1–12. ISBN: 978-1-4799-4394-4.

[Vaish2016] Nilay Vaish, Michael C. Ferris, and David A. Wood. “Optimization Models for Three On-Chip Network Problems”. In: *ACM Transactions on Architecture and Code Optimization* 13.3 (Sept. 2016), pp. 1–27. ISSN: 1544-3566, 1544-3973. DOI: 10.1145/2943781.

[Vangal2008] Sriram R Vangal et al. “An 80-Tile sub-100-W teraFLOPS processor in 65-nm CMOS”. In: *IEEE Journal of solid-state circuits* 43.1 (2008). Publisher: IEEE, pp. 29–41.

[Wang2013] Ruisheng Wang, Lizhong Chen, and Timothy Mark Pinkston. "Bubble Coloring: Avoiding Routing- and Protocol-induced Deadlocks with Minimal Virtual Channel Requirement". In: *International Conference on Supercomputing (ICS)*. event-place: Eugene, Oregon, USA. 2013, pp. 193–202. ISBN: 978-1-4503-2130-3. DOI: 10.1145/2464996.2465436.

[Wang2014] D. Wang, C. Lo, J. Vasiljevic, N. Enright Jerger, and J. Gregory Steffan. "DART: A programmable architecture for NoC simulation on FPGAs". In: *IEEE Transactions on Computers* 63.3 (2014), pp. 664–678. DOI: 10.1109/TC.2012.121.

[Won2015] Jongmin Won, Gwangsun Kim, John Kim, Ted Jiang, Mike Parker, and Steve Scott. "Overcoming far-end congestion in large-scale networks". In: *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. Burlingame, CA, USA, Feb. 2015, pp. 415–427. ISBN: 978-1-4799-8930-0. DOI: 10.1109/HPCA.2015.7056051.

[Yang2017] Lei Yang, Weichen Liu, Peng Chen, Nan Guan, and Mengquan Li. "Task Mapping on SMART NoC: Contention Matters, Not the Distance". In: *54th Annual Design Automation Conference 2017*. Austin TX USA, June 2017, pp. 1–6. ISBN: 978-1-4503-4927-7. DOI: 10.1145/3061639.3062323.

[Yoo2003] Andy B Yoo, Morris A Jette, and Mark Grondona. "SLURM: Simple linux utility for resource management". In: *Workshop on job scheduling strategies for parallel processing*. 2003, pp. 44–60.

[Yoshida2012] Toshio Yoshida, Mikio Hondo, Ryuji Kan, and Go Sugizaki. "SPARC64 VIIIfx: CPU for the K computer". In: *Fujitsu Sci. Tech. J* 48.3 (2012), pp. 274–279.

[Yoshida2018] Toshio Yoshida. "Fujitsu high performance CPU for the post-K computer". In: *Hot chips*. Vol. 30. 2018.

[Zheng2010] Gengbin Zheng, Gagan Gupta, Eric J. Bohm, Isaac Dooley, and Laxmikant V. Kalé. "Simulating Large Scale Parallel Applications Using Statistical Models for Sequential Execution Blocks". In: *International Conference on Parallel and Distributed Systems (ICPADS)*. 2010, pp. 221–228.