Bachelor Informatica

University of Amsterdam

Modeling many-core processor interconnect scalability for the evolving performance, power and area relation

David Smelt

June 9, 2018

Supervisor(s): drs. T.R. Walstra

Signed:

Abstract

Novel chip technologies continue to face power and thermal limits accompanied by the evolving performance, power and area relation. CPU architectures are moving towards ever-increasing core counts to sustain compute performance growth. The imminent many-core era necessitates an efficient and scalable interconnection network.

This thesis elaborates on the underlying causes for compelled energy efficiency and its impacts on microarchitecture evolution. Scalability of various interconnect topologies is evaluated; pragmatically by means of x86 benchmarks and theoretically by means of synthetic traffic. Performance scalability statistics for both existing Intel x86 interconnects and alternative topologies are obtained by means of Sniper and gem5/Garnet2.0 simulations. Power and area models are obtained through McPAT for Sniper simulations and through DSENT for detailed gem5/Garnet2.0 NoC simulations. Garnet2.0 is extended for modeling of NoC power consumption and area with DSENT.

For three existing Intel x86 CPU architectures, microarchitectural details pertaining to scalability and interconnects are laid out. This illustrates the evolution of Intel's x86 CPU interconnection networks, from bus to increasingly more scalable point-to-point interconnects. Scalability of performance, power and area in select Intel x86 processors is examined with the Sniper x86 computer architecture simulator. Interconnect scalability of various bus, ring (NoC) and mesh (NoC) topologies in the simulated Haswell architecture is compared by means of Sniper's results, which include a power and area model by McPAT.

Synthetic traffic simulations at near-saturation injection rate show that for 16 cores, the fully connected topology shows performance equal to the flattened butterfly, both in terms of latency and throughput, due to having more links. Leakage power is the dominant factor in total power in this topology, at 11 nm technology, due to its large amount of links and buffers. The mesh topology draws less power and encompasses less area than the flattened butterfly, at the cost of a higher latency and lower throughput. Simulated mesh topologies scale in accordance with the theoretical asymptotic cost of $\mathcal{O}(n)$ for power and area. The flattened butterfly achieves substantially higher near-saturating injection rates than the mesh, thereby achieving throughputs $6\times$ and $2.4\times$ at 128 cores versus comparative mesh topologies, with concentration factors of 1 and 4, respectively.

Contents

CHAPTER 1

Introduction

1.1 The need for energy efficient multi-core microprocessors

With the release of Intel's Core 2 Duo CPUs in 2006, multi-core CPUs have become commonplace in the consumer market. Before long, single-core performance had hit a wall with the maximum clock speed seeing limited growth. Moore's law, stating that the number of transistors that fit on the same area of microprocessors doubles with each generation (two years), continues to apply. In the past decade, this has led chip architects to face a new problem: the power draw of increasingly larger multi-core microprocessors has increased exponentially, escalating energy and thermal efficiency issues. Simply adding more cores, permitted by transistor scaling, is impeded by the package power consumption. Chip architects are forced to limit the number of cores and clock speed, which in turn limits performance growth. The industry has seen a sustained microprocessor performance increase of 30-fold over one decade and 1000-fold over two decades. In order for future multi-core and many-core microprocessors to continue this trend, energy efficiency is paramount to their design.

Until the year 2006, growth in microprocessor performance has relied on transistor scaling, core microarchitecture techniques and cache memory architecture [1]. Dedicating a large portion of the abundant transistors to cache size has proven to achieve the most substantial performance/Watt increase. Microarchitecture techniques, such as a deep pipeline, are costly and require energy-intensive logic. Relying on transistor speed for continued performance growth is history due to energy concerns.

This new era of energy efficiency has demanded radical changes in both architecture and software design. Chip architects have abandoned many microarchitecture techniques and are increasingly focusing on heterogeneous cores and application-customized hardware. Software is compelled to exploit these architectural advancements and to increase parallelism in order to achieve performance growth.

1.2 The need for scalable interconnects

With the move towards many-core, which ITRS has been predicting over the past decade [2], data must be moved between cores efficiently. The bus or switched-media network that interconnects cores should be used as little as possible, to conserve energy. This emphasizes the need for data locality-optimized software and interconnect architectures. Memory hierarchies and cache coherence protocols should be optimized to cater to novel interconnect architectures. Memory hierarchies have consistently been growing in complexity and capacity. Support for heteregeneneous operation continues to broaden in novel microprocessor architectures. The conventional bus interconnect has proved a major lacking in performance, power and area scalability. Intel moved away from the bus with their introduction of the QuickPath point-to-point interconnect in November 2008. Prompt evolution of the performance, power and area relation and the imminent many-core era necessitate an efficient and scalable interconnection network.

1.3 Thesis overview and research question

Chapter 2 firstly elaborates on the underlying causes for evolution of the performance, power and area relation that effectuate continuous growth in core count and dark silicon. Secondly, the case is made for moving away from badly scalable bus interconnects to either moderately scalable hierarchical bus interconnects or highly scalable networks-on-chip (NoCs). Thirdly, the impact of compelled energy efficiency on interconnect and memory hierarchy evolution is laid out for select exemplary Intel x86 microarchitectures. Fourthly, the case is made for exploration of scalability by means of computer architecture simulators. Finally, primary characteristics of the computer architecture simulators and associated power and area modeling tools employed in this thesis' experiments are laid out.

Chapter 3 details the configuration of the employed Sniper computer architecture simulator and its associated McPAT power and area modeling tool, as well as the extensions built for the employed gem5/Garnet2.0 NoC simulator and its associated DSENT NoC power and area modeling tool.

Chapter 4 details the methodologies for this thesis' experiments and presents their results.

Chapter 5 summarizes previous chapters, thus concluding to which extent performance, power and area of the evaluated interconnect topologies scale with core count; pragmatically, by means of x86 benchmarks and theoretically, by means of gem5/Garnet2.0 synthetic NoC traffic.

CHAPTER 2

Theory and related work

2.1 Dennard scaling and the dark silicon era

As transistors become smaller, transistor supply voltage and threshold voltage (the amount of voltage required for the transistor to turn on) scale down. Dennard scaling states that chip power density, i.e. power consumption per unit of area, theoretically stays constant when scaling down transistor size and voltage [3]. However, scaling of transistors used in recent microprocessors is reaching its physical limits, due to voltage not being able to scale down as much as transistor gate length. Figure 2.1 on the following page shows that Dennard scaling can no longer be upheld for current CMOS-based chips[1]. Instead, power density tends to increase exponentially with current technology scaling, causing heat generation to become a limiting factor.

As the number of transistors grows exponentially, exponentially more parts of the chip need to be switched off (power gating) or run at a significantly lower operating voltage and frequency. Esmaeilzadeh et al. have coined this phenomenon "dark silicon" in their 2011 paper [4]. Esmaeilzadeh et al. project that at a process size of 8 nm, over 50% of the chip cannot be utilized; this part of the chip will be entirely "dark". Even when transistors are turned off, they leak a small amount of current, which increases exponentially with threshold voltage reduction[2]. The exponential growth in transistor count exacerbates this effect, causing power consumption due to leakage to increase exponentially as process size scales down. Along with the exponential increase in power density, this limits transistor voltage scalability.

2.2 Power consumption in CMOS chips

2.2.1 Taxonomy

The following definitions and equations are summarized from [7]. Power consumption in CMOS-based chips can be broken down into two main components: dynamic power and static (leakage) power. Dynamic power is the power consumed when transistors are switching between states and static power denotes the power drawn by leakage currents, even during idle states. Total power consumption is expressed as (2.1):

$$P_{total} = P_{dynamic} + P_{static} \tag{2.1}$$

The dynamic power component is proportional to the capacitance C, the frequency F at which gates are switching, squared supply voltage V and activity A of the chip, which denotes the number of gates that are switching. Additionally, a short circuit current I_{short} flows between the supply and ground terminals for a brief period of time τ, whenever the transistor switches states. Thus, dynamic power is expressed as (2.2):

$$P_{dynamic} = ACV^2F + \tau AVI_{short} \tag{2.2}$$

[1] CMOS stands for complementary metal-oxide-semiconductor, which is a technology for constructing integrated circuits, such as microprocessors, microcontrollers and other digital logic circuits. CMOS-based chips typically employ MOSFETs (metal-oxide-semiconductor field-effect transistors) for logic functions.

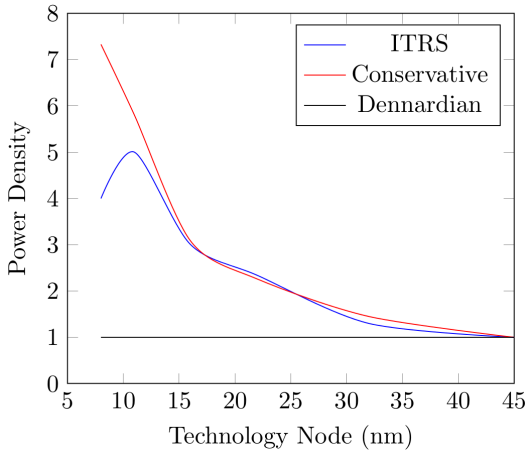


Figure 2.1: Three predictions of the increase in power density for CMOS-based chip technology scaling. Power density denotes the power consumption per unit of area. The Dennardian scaling law states that chip power density theoretically stays constant when scaling down transistor size and voltage. ITRS projections (2013) [5] and conservative Borkar projections (2010) [6] present an exponential rise in power density. Image source: [7].

Frequency varies with supply voltage V (alias V_{dd}) and threshold voltage V_{th} as in (2.3):

$$F \propto \frac{(V - V_{th})^2}{V} \tag{2.3}$$

Static power is expressed as (2.4):

$$P_{static} = V I_{leak} = V \times (I_{sub} + I_{ox}) \tag{2.4}$$

Leakage current I_{leak} is the primary factor in static power consumption and is split into two components: sub-threshold leakage I_{sub} and gate oxide leakage I_{ox}. Sub-threshold leakage is the current that flows between the source and drain of a transistor when gate voltage is below V_{th} (i.e. when turned off). Gate oxide leakage is the current that flows between the substrate (source and drain) and the gate through the oxide.

The continuous scaling down of transistor size has led to scaling down of V_{th}, which causes sub-threshold leakage to increase exponentially. Moore's law and the growing proportion of dark silicon exacerbate this issue, due to growing a occurrence of transistors residing in off-state. Scaling down transistor size involves reducing gate oxide thickness. The gate oxide leakage increases exponentially as oxide thickness is reduced, which causes gate oxide leakage to approach sub-threshold leakage.

Equation 2.2 on the previous page shows that scaling down transistor voltage can reduce dynamic power by a quadratic factor, outweighing the other – linear – factors. In previous CMOS-based technologies, static power used to be an insignificant contributor to total power. However, static power is increasing at such a considerable rate that it would exceed dynamic power with further scaling of existing technologies. In addition to thermal regulation techniques, leakage reduction techniques are therefore essential in future technology design.

2.2.2 CMOS versus the novel FinFET technology

On June 13, 2017 GlobalFoundries announced the availability of its 7 nm FinFET semiconductor technology. A fin field-effect transistor (FinFET) is a MOSFET tri-gate transistor, which has an elevated source-drain channel, so that the gate can surround it on three sides. In contrast, a conventional CMOS MOSFET comprises a planar 2D transistor.

Figure 2.2 on the following page shows dynamic and leakage power consumptions in the c432 benchmark circuit[3] for proposed 7 nm FinFET standard cell libraries and conventional bulk CMOS standard

cell libraries [8]. The 14 nm CMOS circuits have a V_{th} of 0.52 V and the 7 nm FinFET circuits have a normal V_{th} of 0.25 V and a high V_{th} of 0.32 V.

Firstly, the results show that the 0.55 V 14 nm CMOS operating at near-threshold voltage has the largest leakage power proportion. Secondly, due to their high on/off current ratio, FinFET devices experience a higher ratio between dynamic and leakage power consumption than CMOS circuits. Thirdly, the high V_{th} 7 nm FinFET consumes up to 20× less leakage power than the normal V_{th} 7 nm FinFET. Fourthly, the 0.55 V 14 nm CMOS consumes only slightly more power than the normal V_{th} 7 nm FinFET, due to the delay of the FinFET being much shorter than that of the CMOS, causing circuits to run much faster and consume more power. Fifthly, the normal V_{th} 7 nm FinFET exhibits speed-ups of 3× and 15× versus the 14 nm and 45 nm CMOS circuits, due to smaller gate length and parasitic capacitance. Sixthly, when operating in the super-threshold regime, on average, the normal V_{th} 7 nm FinFETs consume 5× and 600× less energy and the high V_{th} 7 nm FinFETs consume 10× and 1000× less energy than the 14 nm and 45 nm CMOS circuits, respectively. Finally, when operating in the near-threshold regime, on average, the high and normal V_{th} 7 nm FinFETs can consume 7× and 16× less energy than the 14 nm CMOS. FinFET devices thus present a promising substitute for CMOS-based devices, especially at 7 nm technology and beyond.

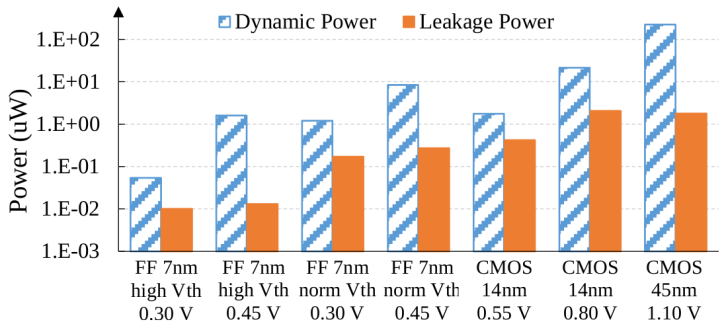


Figure 2.2: Dynamic and leakage power consumptions in the c432 benchmark circuit for proposed 7 nm FinFET standard cell libraries and conventional bulk CMOS standard cell libraries [7].

2.3 Microprocessors and the evolving performance, power and area relation

On June 20, 2016, Intel released their latest many-core x86 microprocessors comprising the Knights Landing (KNL) architecture. The KNL microprocessors are dubbed the Xeon Phi x200 series and feature either 64, 68 or 72 quad-threaded Intel Atom cores at a TDP[4] of 215-260 W. Section 2.7.3 on page 23 elaborates on the KNL architecture.

KNL's 72-core flagship models run at 1.5 GHz base clock frequency and are actually comprised of 76 cores, of which four are inactive, marking the dawn of the dark silicon era. For two-core workloads, all models can boost to a turbo frequency, which adds 200 MHz to their base frequency. Workloads of three cores and over can only achieve a boost of 100 MHz and workloads with high-AVX SIMD instructions actually effectuate a frequency reduction of 200 MHz. Although heterogeneity is still minimal, KNL exemplifies the effects that continued microprocessor technology advancement trends towards larger core counts have on performance, power and area. Additional cores lead to an increase in power consumption. Due to the aforementioned continued increase in power density and thermal runaway, cores need to be placed further apart to keep temperature down. Figure 2.3 illustrates the increase in area of KNL compared to previous 22-core and 16-core Xeon CPUs. Single-core performance in many-core architectures such as KNL suffers greatly, due to the substantial drop in clock frequency imposed by power draw and heat generation.

[3]Description of the ISCAS-85 C432 27-channel interrupt controller: `http://web.eecs.umich.edu/~jhayes/iscas.restore/c432.html` – accessed June 6, 2018.

[4]Thermal Design Power (TDP) is the maximum amount of heat that a CPU's cooling system is designed to dissipate during practical workloads.

High-end desktop and workstation microprocessors presently feature up to 18 cores, marked by the release of Intel's Core i9 7980XE for the Skylake-X architecture, in September 2017. The Core i9 7980XE has a TDP of 165 W and runs at a 2.6 GHz base clock frequency, with a possible boost to 4.0 GHz during quad-core workloads. This is a substantial drop in base frequency, compared to its 14-core sibling, the Core i9 7940X, which runs at 3.1 GHz. AMD's flagship offering, the Ryzen Threadripper 1950X, was released in August 2017 and comprises 16 cores at a TDP of 180 W. Its base frequency of 3.4 GHz is close to the 3.5 GHz of its 12-core sibling, the Ryzen Threadripper 1920X. The Ryzen Threadripper 1950X measures in at about 72×55 mm, with a core size of 11 mm^2, whereas the Core i9 7980XE measures in at about 53×45 mm, with a core size of 17 mm^2. AMD's current flagship high-end desktop CPU thus manages to attain a substantially lower power density than Intel's offering, while both are built using the 14 nm FinFET process. Intel still outperforms AMD in most benchmarks, although at a much worse price-performance ratio.

Heterogeneity is becoming an increasingly larger factor in CPU microarchitectures, with both AMD and Intel frequently updating their turbo boost feature sets. The high-end 16-to-18-core desktop CPUs currently offered by AMD and Intel, as well as Intel's Knights Landing architecture, are prime examples of the direction CPU microarchitecture design is heading in. The first iteration of desktop CPUs with around 32 cores will most likely borrow aspects from both Skylake-X and Knights Landing.

Figure 2.3: Size comparison of an Intel Xeon E5 v4 series $\leq$ 22-core CPU (Broadwell-EP), a Xeon Phi x200 series $\leq$ 72-core CPU (Knights Landing, socket LGA3647) and a Xeon D series $\leq$ 16-core CPU (Broadwell-DE).[5] Section 2.7.3 on page 23 details the Xeon Phi x200 series processors.

2.4 System-on-chip (SoC) architectures

A system-on-chip (SoC) is a single integrated circuit that houses all necessary components of a computer system. A typical multi-core SoC integrates a microprocessor and memory, as well as numerous peripherals such as GPU, WiFi or coprocessor(s). Distinct types of SoCs, such as field-programmable gate arrays (FPGAs) or certain application-specific integrated circuits (ASICs) also carry programmable logic. In the past decade, microprocessors have been integrating more and more peripherals on their die. Integrated graphics, memory controller, as well as PCI Express and DMI links are emphasizing the importance of data locality. This development is driving microprocessor architectures in the direction of SoC.

[5]Image source: `https://www.servethehome.com/big-sockets-look-intel-lga-3647/broadwell-ep-lga-2647-broadwell-de-package-size-comparison/` – accessed April 23, 2018.

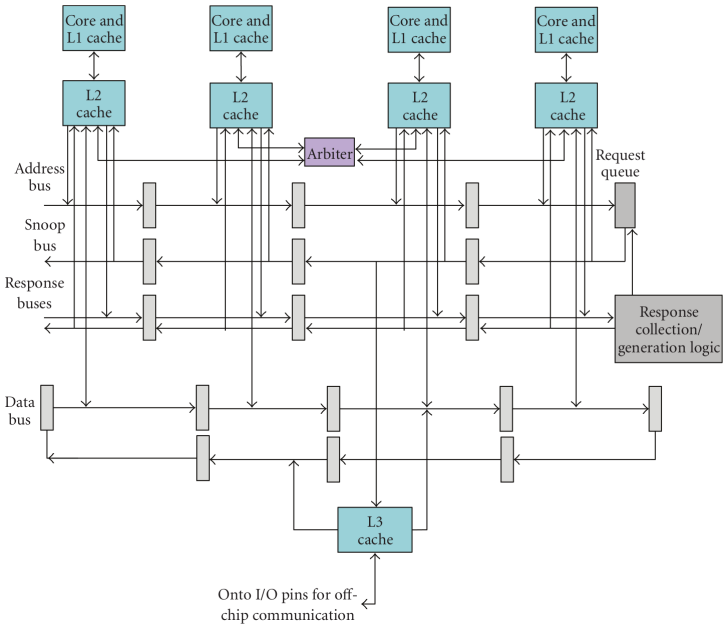


Figure 2.4: Detailed high-level diagram of a multi-core bus architecture. Image source: [9].

2.5 Bus-based architectures

2.5.1 Buses

Historically, a bus used to be the predominant interconnect structure for SoCs. A bus is a shared medium for connecting multiple computer components on a single chip or across multiple chips. One sender at a time is allowed to communicate with the other members of the medium. New devices can easily be added to the bus, facilitating portability of peripherals between different systems. Figure 2.4 shows a detailed high-level diagram of a multi-core bus architecture. The bus-based interconnect pictured consists of multiple buses, a centralized arbitration unit, queues, and other logic. Sections 2.5.2 on the following page and 2.5.3 on page 13 explain intra-bus operation in greater detail.

Section 2.8 on page 24 shows the major deficit in scalability of the bus, as compared to a network-on-chip (NoC) interconnect with a 2D mesh topology. Buses used to be a cost-efficient interconnect structure, due to its modularity and relatively low design complexity. In the present multi-core era, a traditional bus is not a viable interconnect structure anymore. When too many cores (i.e. more than eight) are connected to the same bus, the available bandwidth of the bus becomes a bottleneck, since all devices on the bus share the same total bandwidth. Additionally, clock skew and delay become bottlenecks. Due to high switching activities and large capacitive loading, a bus consumes a significant amount of power.

When connecting more than eight cores, most literature prefers more scalable packet-switched interconnects, such as NoC (Section 2.9 on page 24). However, Udipi et al. propose a hierarchical bus-based on-chip network [10]. They show that bus-based networks with snooping cache protocols (see Section 2.5.2 on the following page) can exhibit superior latency, energy efficiency, simplicity and cost-effectiveness, compared to the large number of routers employed in packet-switched networks, with no performance loss at 16 cores. Section 2.5.3 on page 13 lays out the details of this proposed novel hierarchical bus architecture, while delving deeper into the the basics of bus design.

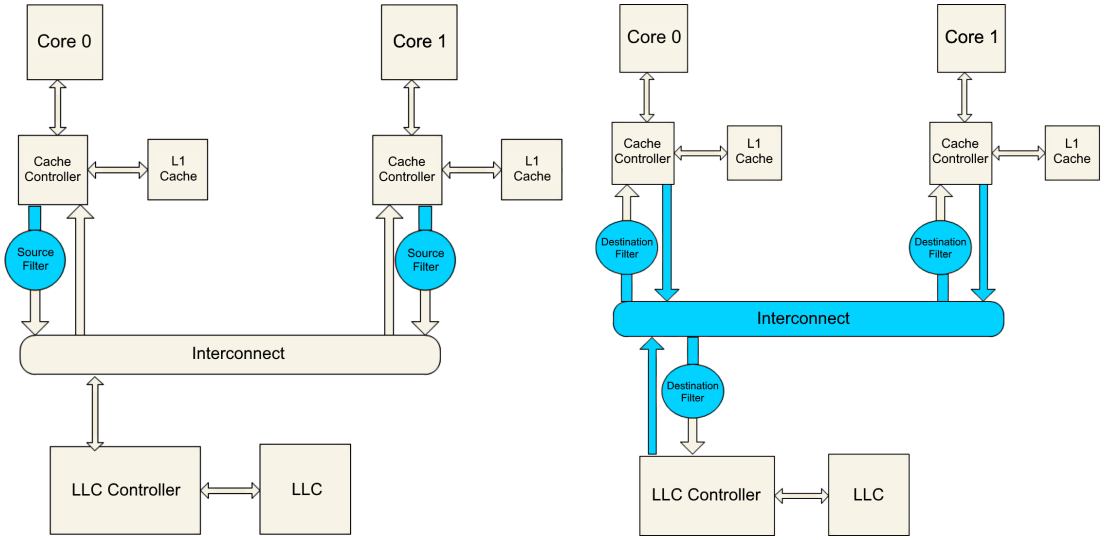


Figure 2.5: *Left*: source snoop filter. *Right*: destination snoop filter. Parts highlighted in blue show which parts of the standard snooping protocol are affected by the addition of one of the filters. Image source: [11].

2.5.2 Bus-based caches and cache coherence

A bus-based multiprocessor connects each of its CPUs to the same bus that the main memory connects to. Since this structure would quickly overload the bus, high speed cache memories are added between each CPU and the bus. Caches with high hit rates are imperative for performance, since this reduces the amount of bus requests for words of data. Cache coherence requires that for all CPUs any variable that is to be used must have a consistent value. In order for cache memories to stay coherent across each CPU, initially, write-through cache and snoopy cache have been developed.

In a write-through cache scheme, whenever a word is written to the cache, it is written through to main memory as well. Snoopy cache gets its name from the fact that all caches are constantly snooping on (monitoring) the bus. Whenever a cache detects a write by another cache to an address present in its cache, it either updates that entry in its cache with the new value (write-update) or it invalidates that entry (write-invalidate). Both write-through and snoopy cache can decrease power consumption and coherency traffic on the bus.

A snoop filter is a means to mitigate unnecessary snooping. It is based on a directory based structure and monitors all traffic in order to keep track of the coherency states of caches. This reduces snoop power consumption, but the filter itself will introduce some extra power consumption and complexity. Two types of traditional snoop filters exist: a source filter is located between the cache controller and the bus, whereas a destination filter sits at the bus side, only filtering transactions going through the TLB and into the cache. Figure 2.5 shows a schematic comparison between source and destination filters; the parts highlighted in blue show which parts of the standard snooping protocol are affected by the addition of one of the filters. The snoop filter can operate either exclusively or inclusively. An inclusive filter holds a superset of all addresses currently cached; a hit occurring in this filter means that the requested cache entry is held by caches and any miss will occur in the cache as well. An exclusive filter holds a subset of all addresses currently not cached; a hit occurring in this filter means that no cache holds the requested cache entry.

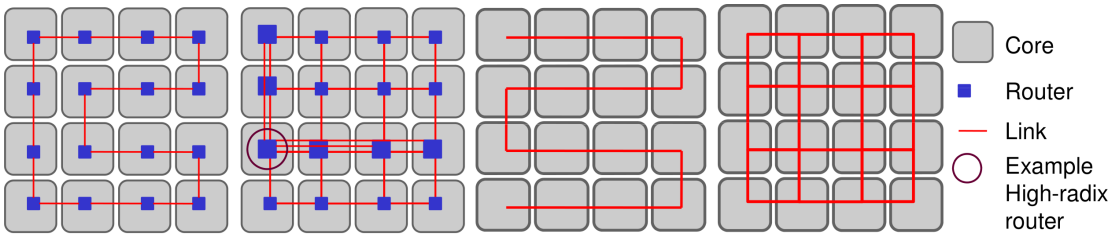
Intel's bus-based IA-32 and Intel 64 processors use the MESI (modified, exclusive, shared, invalid) cache protocol, which acts like a source filter, making snoop filters redundant. As implemented in these processors, the MESI protocol maintains cache coherence against other processors, in the L1 data cache and in the unified L2 and L3 caches. Each cache line can be flagged as any of the states shown in Table 2.1 on the next page.

Additionally, IA-32 and Intel 64 processors generally employ a write-back cache scheme. In this scheme, writes are initially only made to the cache. Only when cache lines need to be deallocated, such as when the cache is full or when invoked by one of the cache coherency mechanisms, a write-back operation is triggered, writing the cache lines to the main memory. If data of a miss resides in another

cache, the relevant cache line is posted on the bus and transferred to the requesting cache. In bus-based systems, such cache to cache transfers are generally faster than retrieving the line from main memory. Multi-core architectures generally include a shared L3 cache on the chip. In this case, a likely faster alternative would be to transfer the missed line from L3 cache. The MESI protocol, paired with a write-back scheme, reduces the amount of cache coherency traffic greatly, thereby reducing snoop induced power consumption greatly [12]. In Intel's bus-based architectures, the MESI protocol turned out to provide the best performance.

	Dirty?	**Unique?**	**Can write?**	**Can forward?**	**Can silent transition to:**	**Comments**
Modified	Dirty	Yes	Yes	Yes		Must write-back to share or replace
Exclusive	Clean	Yes	Yes	Yes	MSI	Transitions to M on write
Shared	Clean	No	No	Yes	I	Shared implies clean, can forward
Invalid	–	–	–	–		Cannot read

Table 2.1: Possible states for a cache line in the MESI protocol, implemented by bus-based Intel IA-32 and Intel 64 processors [13].



(a) Ring network (b) Flattened butterfly network (c) Conventional bus (d) Shorted bus

Figure 2.6: Example baseline routing structures for four different interconnects with a 4×4 grid topology [10].

2.5.3 A filtered segmented hierarchical bus-based on-chip network

Udipi et al. [10] model a processor with either 16, 32 or 64 cores. Each core has a private L1 cache and a slice of the shared L2 cache. Parameters for the 16-core model are shown in Table 2.2 on the following page. The 16-core processor is structured in tiles of 4×4 cores and the 64-core model is comprised of 8×8 tiles. The bus topology is based on a shorted bus, where all interconnects are electrically shorted all around, as illustrated in Figure 2.1(d). A repeater sits at every tile in order to reduce latency and allow high frequency operation. Each link is composed of two sets of wires, each of which heads in the opposite direction. This configuration improves performance compared to the conventional bus (Figure 2.1(c)), since each transaction delays the bus for fewer cycles. It also avoids the issue of indirection between coherence transactions, as with any bus. The bus arbiter assumes only one outstanding bus request per node. The request signal is activated until a grant is received, upon which the coherence request is placed on the address bus. The next request can be placed on the address bus after a set amount of cycles, whereas coherence responses are handled on a separate control and data bus.

Die parameters	10mm × 10mm, 32nm, 3GHz
L1 cache	Fully Private, 3 cycle 4-way, 32KB Data 2-way, 16KB Instr
L2 cache	Fully shared, unified S-NUCA 8-way, 32MB total, 2MB slice/tile 16 cycles/slice + network delay
Main memory latency	200 cycles
Router	4 VCs, 8 buffers/VC, 3 cycles

Table 2.2: General parameters for the 16-core model [10].

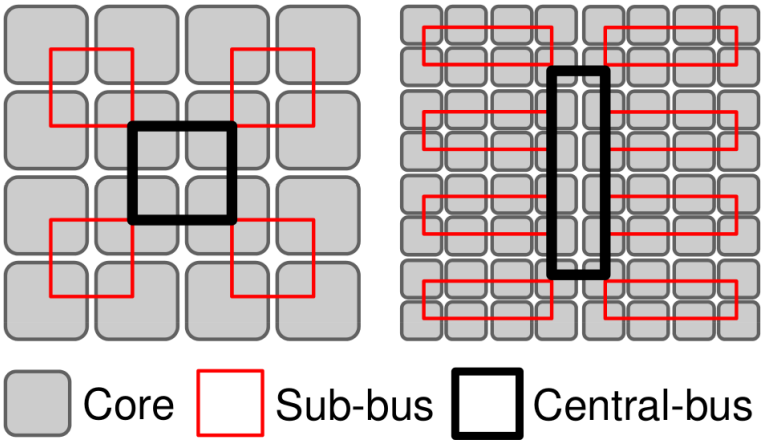


Figure 2.7: Segmented bus structures for the 16-core and 64-core processors, respectively. Each of the intra-cluster sub-buses is connected to one inter-cluster central bus. [10]

Udipi et al. [10] propose dividing the processor into several segments of cores. Within each segment, each core is connected through a shorted sub-bus. All sub-buses are connected to a central bus, as shown in Figure 2.7. The 16-core model is comprised of 4 segments of 4 cores. The 32-core model expands the central bus to connect 8 segments of 4 cores. The 64-core model increases the segment size to 8, connecting 8 cores. Alternatively, a segment size of 4 would significantly increase latency and ownership contention, since this causes 16 sub-buses to be connected to one large central bus. Passing of messages between the sub-buses and the central bus is enabled through simple tri-state gates, which are situated at each of their intersections.

The same global arbitration scheme as in the case of the single shorted bus is used. The global arbitration scheme is extended with a method that looks for three sequential cycles i, $i+1$ and $i+2$ where the originating sub-bus, central bus and remote sub-buses are free for broadcast, respectively. This allows every bus transaction to be pipelined in three stages: the originating sub-bus broadcast, the central bus broadcast and the remote sub-buses broadcasts. Thus, throughput is increased and bus contention is reduced. However, if a transaction has to wait for access to the central bus, the broadcast would have to be cancelled and retransmitted. Some small buffering of messages is implemented in order to alleviate this. This also resolves potential deadlock situations that may arise, for example when transaction A has completed its local broadcast and is waiting for the central bus, occupied by B, and B, having completed its central broadcast, is waiting for A's sub-bus.

A transaction passes through a Bloom filter, at the border of each segment, which decides whether a core outside of the segment needs to see the transaction. If so, the transaction arbitrates to get onto the central bus and thereafter to the remote buses. If not, the transaction is deemed complete. Counting Bloom filters are employed in order to remove invalidated elements from the filter. Additionally, OS-assisted page coloring is implemented, ensuring that the majority of transactions do not have to leave their local segment. These locality optimizations diminish link energy consumption significantly while improving performance.

Low-swing wiring is also employed. This reduces link energy consumption by reducing the voltage

range through which the wires are charged and discharged. Furthermore, Udipi et al. [10] extend their segmented filtered bus model to employ two parallel buses, interleaved by address. This increases concurrency and bandwidth, at the cost of an increase in wiring area and power, but avoiding the overheads of complex protocols and routing elements. The additional buses would introduce some leakage energy overhead, but no dynamic overhead. The leakage is likely to be lower than the leakage incurred by the buffers in an over-provisioned packet-switched network.

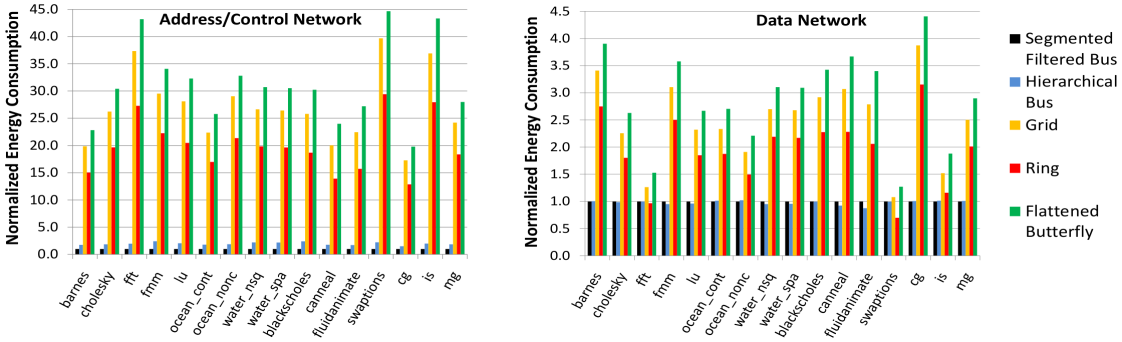


Figure 2.8: 16-core model: resulting relative energy consumption of the proposed segmented filtered bus and hierarchical bus, as well as three packet-switched networks, for both the address network and the data network – normalized with respect to the segmented filtered bus. [10]

Component	Energy (J)
2.5 mm low-swing wire	3.02e-14
Bus arbiter	9.85e-13
Bloom filter	4.13e-13
2.5 mm full-swing wire	2.45e-13
Single-entry buffer	1.70e-13
Tri-state gates (64)	2.46e-12
3x3 ring router	7.32e-11
7x7 flattened butterfly router	2.24e-10
5x5 grid router	1.39e-10

Table 2.3: Energy parameters for the 16-core model [10].

The paper points out four major contributors to energy consumption for the filtered hierarchical bus network. Their relative contributions to total energy consumption remain fairly constant across benchmarks. The average values are: 75.5% for link traversal, 12.3% for the tri-state gates, 7.7% for arbitration, 3.2% for Bloom filtering and, least significantly, 1.2% for message buffering. The resulting relative energy consumption of the proposed segmented filtered bus and hierarchical bus and packet-switched networks comprised of a grid, ring and flattened butterfly topology are compared for the 16-core model. Figure 2.8 displays charts of these results. In the address network, even the most energy efficient packet-switched network (ring) consumes an average of 20× as much energy as the segmented filtered bus, which achieves a best case reduction of 40× versus the flattened butterfly network. In the data network, an average energy reduction of 2× is achieved compared to the most energy efficient packet-switched network (once again, ring), with a best case reduction of 4.5× versus the flattened butterfly network. The energy consumptions of the segmented filtered bus and hierarchical bus in the data network are basically the same. The large energy difference between the bus-based and packet-switched networks can be explained by the fact that a single router traversal can consume up to 7× as much energy as a simple link traversal. Low-swing wiring further increases this disparity to up to 60×, as can be inferred from Table 2.3.

The segmented filtered bus outperforms all other tested networks in terms of execution time, by 1% compared to the next-best flattened butterfly network, with a best case improvement of 6%. This is due to the inherent indirection of a directory based system, as well as the deep pipelines of complex routers increasing zero-load network latency.

When scaling the 16-core model to 32 and 64 cores, the same latency and energy parameters are retained. In the 32-core model, an average energy reduction of 19× is observed, with a best case reduction of 25×, due to the same reasons as for the 16-core model. In this case, more nodes are making requests to the central bus, leading to slightly less efficient filters, increasing contention. An average performance drop of 5% is observed, with a worst case of 15%.

Compared to the flattened butterfly network, the 64-core model achieves average energy reductions of 13× and 2.5× in the address and data network, respectively, with best case reductions of 16× and 3.3×. This further impedes performance, once again due to increased contention, resulting in a 46% increase in execution time compared to the flattened butterfly network. Upon implementing multiple address interleaved buses, contention gets dispersed and performance deficits drop down to 12% shy of the flattened butterfly. Udipi et al. [10] do not quantify the energy drawbacks associated with this extension. Nonetheless, the substantial improvement in comparative energy efficiency of the baseline 64-core model, along with the 1.5-fold increase in energy efficiency by means of low-swing wiring in the 16-core model, indicate remarkable potential for the extension of bus scalability – albeit limited – beyond conventional belief. The 64-core address-interleaved filtered segmented bus could very well form the basis for a well balanced bus-based architecture in terms of power, performance, simplicity and cost-effectiveness, worthy of further research.

2.6 Ring- and mesh-based architectures

2.6.1 Core-uncore topology of ring- and mesh-based NUMA systems

Uniform memory access (UMA) architectures are characterized by each core of a multiprocessor experiencing the same access time to memory. In a traditional bus-based UMA system, only one core at a time can have access to any of the main memory modules. When large blocks of memory need to be accessed, this can lead to starvation of several cores. Therefore, non-uniform memory access (NUMA) was developed. NUMA assigns a separate chunk of memory to each core. A core accessing the memory assigned to a different core needs to traverse a separate, longer interconnect. Since this incurs more latency compared to accessing its "local" memory, the system experiences non-uniform memory access time.

With Intel's release of their Nehalem microarchitecture in November 2008, the QuickPath Interconnect (QPI) was introduced. QPI is a high-speed point-to-point processor interconnect that replaces the front-side bus (FSB). The QPI architecture relocates the memory controller, which used to be linked through the FSB, distributed next to each core in the processor package and all cores are interlinked by the QPI. This enables a NUMA architecture, as can be seen in Figure 2.9 on the following page. The QPI architecture pictured employs a fully connected topology.

Intel uses the term "uncore" to denote the parts of the chip external to the cores. Typically, the core includes the ALUs, FPUs, L1 and L2 cache, whereas the uncore includes the memory controller, L3 cache, CPU-side PCI Express links, possible GPU and the QPI. By bringing these parts physically closer to the cores, their access latency is reduced greatly. The uncore acts as a building block library, enabling a modular design for multiple socket processors. Starting with the Sandy Bridge microarchitecture (built at a 32nm process size) in 2011, Intel moved the L3 cache from the uncore to the core, for improved bandwidth, latency and scalability (by enabling L3 cache to be distributed in equal slices). The QPI is used for internal core to uncore communication, as well as external communication when connecting multiple sockets. Although the QPI architecture greatly ameliorates the bandwidth scalability issues of the FSB, both intra- and inter-processor communication introduce routing complexity.

[6]URL: `https://www.intel.com/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf` – accessed April 9, 2018.

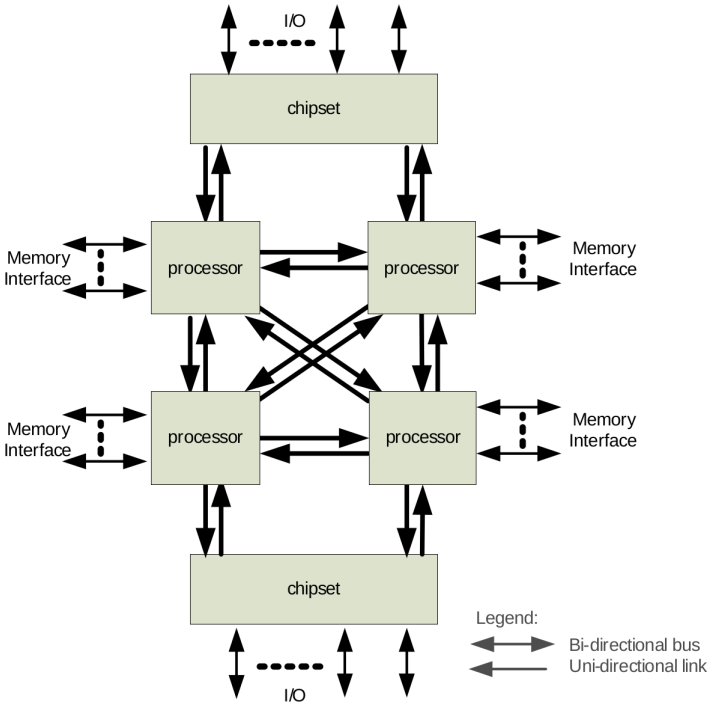


Figure 2.9: Diagram of Intel's QuickPath Interconnect architecture, enabling non-uniform memory access. Image source: *An Introduction to the Intel® QuickPath Interconnect*[6].

With the release of their Haswell microarchitecture (22 nm) in June 2013, Intel restructured the QPI architecture to form a scalable on-die dual ring. This QPI topology continued with the Broadwell microarchitecture (14 nm), which followed in September 2014. This was succeeded by the Skylake microarchitecture in August 2015, which replaced the ring with a full mesh topology, assuming a new name: Ultra Path Interconnect (UPI). Diagrams of example QPI and UPI topologies are shown in Figure 2.10 on the next page. The top diagram pertains to a high core count Intel Xeon E5 v4 processor, based on the Broadwell-EP microarchitecture, with the maximum amount of supported cores of 24 at a TDP of 145 W. As the number of cores increased from previous generations, the chip was divided in two halves, introducing a second ring to reduce latency and increase the bandwidth available to each core. The diagram shows two sets of red rings, each of which represents the QPI moving in either direction. Buffered switches facilitate communication between the two QPI rings, incurring a five-cycle penalty.

The bottom diagram in Figure 2.10 on the following page represents an extreme core count Intel Xeon Scalable processor, based on the Skylake-SP microarchitecture, with the maximum amount of supported cores of 28 at a TDP of 165-205 W. The UPI employs a full 2D mesh topology, providing more direct paths than the preceding ring structure. The mesh-structured UPI contains pairs of mirrored columns of cores. As is evident from the diagram, horizontal traversal covers a larger distance than vertical traversal; moving data between neighboring cores of different columnar pairs takes three cycles, whereas vertically neighboring cores require one cycle. Intel's Sub-NUMA Clustering (SNC) splits the processor into two separate NUMA domains, mitigating the latency penalty of traversal to distant cores and caches.

The 6×6 mesh of the above Intel Xeon Scalable processor includes nodes for the memory controllers and I/O. Three independent 16-lane PCIe pipeline nodes now enable multiple points of entry, greatly improving I/O performance. Select Skylake processors feature a dedicated fourth PCIe 16× link for connecting Intel's Omni-Path high-performance communication fabric. Omni-Path currently supports up to 100 Gb/s per port or link, which is about 22% faster than the theoretical maximum speed of Skylake's UPI.

Geared towards high performance computing, Knights Landing many-core processors feature two Omni-Path-dedicated PCIe 16× links and only one additional PCIe 4× link. Intel actually removed the

[7]URL: `https://software.intel.com/sites/default/files/managed/33/75/130378_hpcdevcon2017_SKL_Tuning.pdf` – accessed April 9, 2018.

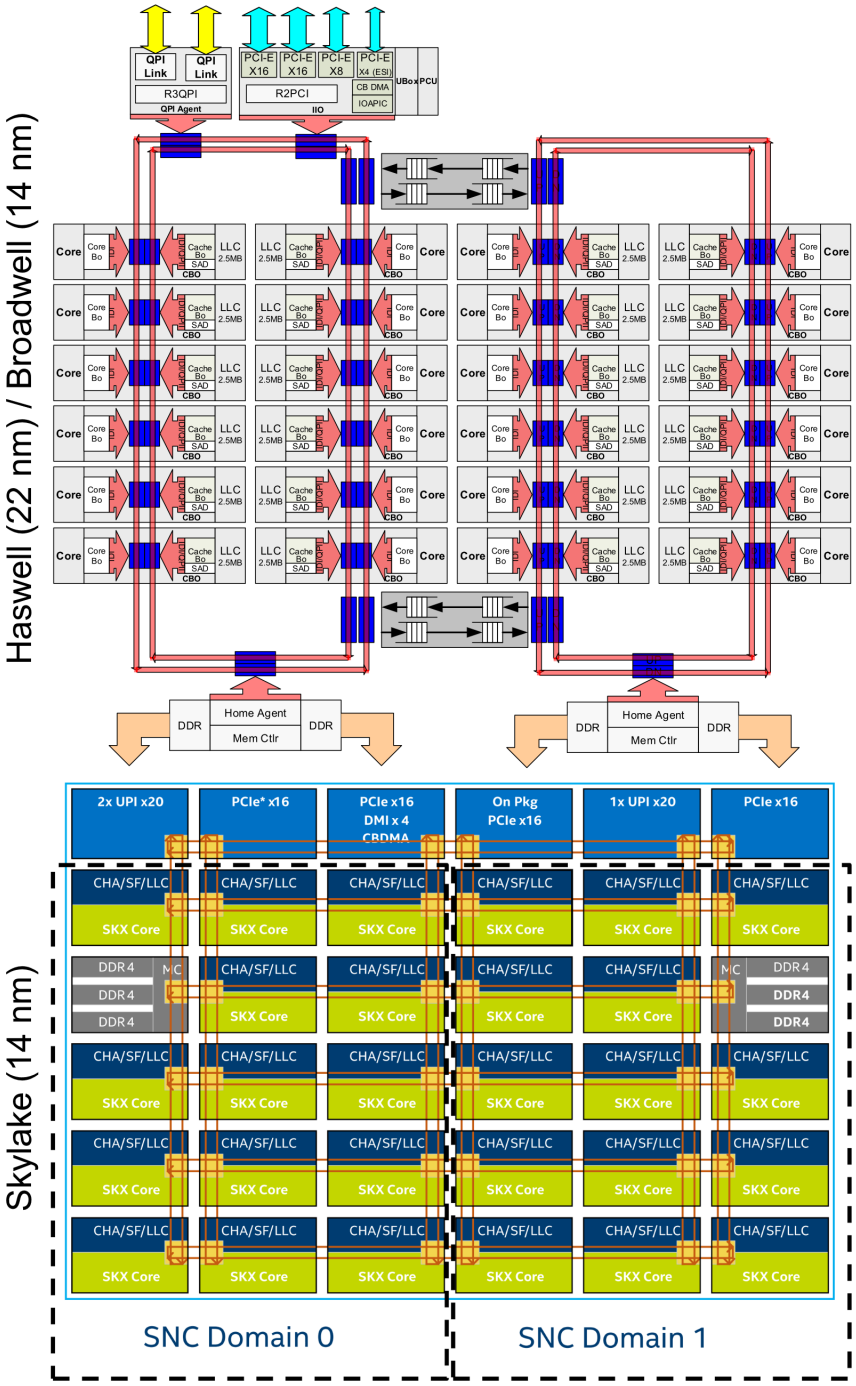


Figure 2.10: *Top*: diagram of the core-uncore ring topology of a 24-core Intel Xeon E5 v4 processor. Each set of two red rings represents the QuickPath Interconnect moving in either direction. Buffered switches interconnect the two sets of QPI rings.
Bottom: diagram of the core-uncore mesh topology of a 28-core Intel Xeon Scalable processor, successor of the top diagram's topology. The red arrows represent wired pathways of the Ultra Path Interconnect and the yellow squares represent switches. Intel's Sub-NUMA Clustering (SNC) splits the processor into two separate NUMA domains. The mesh enables a more direct path, as well as many more pathways, allowing operation at lower frequency and voltage while still providing higher bandwidth. Image source: *Tuning for the Intel® Xeon® Scalable Processor*.[7]

UPI interconnecting other sockets in these processors. This forces administrators of these systems to

interconnect multiple sockets via Omni-Path. Inter-socket communication is placed outside of the chip, onto the Omni-Path fabric, connected through the dedicated PCIe 16× link(s). This way, existing I/O channels are freed, allowing for increased memory bandwidth at the cost of an increase in base latency, for which the decreased contention strives to make up. The dynamicity of the new Omni-Path fabric reduces the design complexity and cost of the inter-socket architecture. This enhances scalability in large multi-socket many-core HPC systems. Section 2.7.3 describes the Knights Landing architecture in greater detail.

The UPI architecture also distributes the caching and home agent (CHA) over all cores, following the design of distributed LLC. This cuts down communication traffic between the CHA and LLC substantially, reducing latency. UPI's mesh topology enables a more direct core-uncore path, as well as many more pathways. Since this mitigates bottlenecks greatly, the mesh can operate at a lower frequency and voltage while still providing higher bandwidth (10.4 GT/s versus QPI's 9.6 GT/s) and lower latency. This testifies to the superior scalability of the mesh topology compared to the ring.

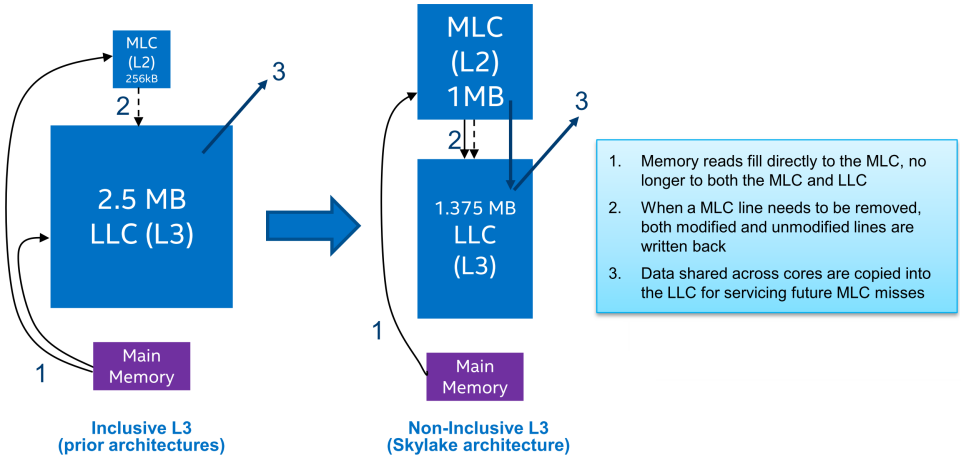


Figure 2.11: *Left flowchart*: cache hierarchy for Intel QPI-based server processors. *Right flowchart*: redesigned cache hierarchy for Intel UPI-based server processors. Image source: *Tuning for the Intel® Xeon® Scalable Processor*.[7]

2.6.2 Ring- and mesh-based caches and cache coherent NUMA

Along with the remodeling of the QPI to the UPI, Intel's Skylake microarchitecture (August 2015) features a redesigned hierarchy for the private mid-level cache (MLC, which is L2) and shared last level cache (LLC, which is L3). Figure 2.11 illustrates the changes. Prior architectures employ a shared-distributed cache hierarchy, where memory reads fill both the MLC and LLC. When an MLC line needs to be removed, both modified and unmodified lines are written back to the LLC. In this case, the LLC is the primary cache and contains copies of all MLC lines (i.e. inclusive). The Skylake architecture employs a private-local cache hierarchy, where memory reads fill directly to the MLC. Here, the MLC is the primary cache and the LLC is used as an overflow cache; copies of MLC lines may or may not exist in the LLC (i.e. non-inclusive). Data shared across cores are copied into the LLC for servicing future MLC cache misses. This new cache hierarchy grants virtualized use-cases a larger (factor of 4) private L2 cache free from interference. The increased L2 size also enables multithreaded workloads to operate on larger data per thread, reducing uncore activity.

The efficiency of a NUMA system relies heavily on the scalability and efficiency of the implemented cache coherence protocol. Modern NUMA architectures are primarily cache coherent NUMA (ccNUMA). The MESI cache coherence protocol provided the best performance in Intel's bus-based architectures. However, in higher core count ccNUMA systems, the MESI protocol would send an excessive amount of redundant messages between different nodes. When a core requests a cache line that has copies in multiple locations, every location may respond with the data. As a result, Intel adapted the standard MESI protocol to MESIF (modified, exclusive, shared, invalid, forward) for their point-to-point interconnected ccNUMA microarchitectures. Alternatively, the MOESI protocol, used for example in

AMD Opteron processors, adds an Owner state, which enables sharing of dirty cache lines without writing back to memory. Intel presumably did not implement the O state to favor reduced complexity over a minor performance gain.

MESIF was the first source-snooping cache coherence protocol, proposed by J.R. Goodman and H.H.J. Hum in 2004 [14]. In MESIF, the M, E, S and I states remain the same as in the MESI protocol (detailed in Section 2.5.2 on page 12), but the new F state takes precedence over the S state. Each cache line can be flagged as any of the states shown in Table 2.4 and a state machine for MESIF is shown in Figure 2.12 on the following page. Only one single instance of a cache line can be in the F state and only that instance may respond and forward its data. The other cache nodes containing the data are placed in the S state and cannot be copied. By assuring a single response to shared data, coherency traffic on the interconnect is reduced substantially. If an F-state cache line is evicted, copies may persist in the S state in other nodes. In this case, a request for the line is satisfied from main memory and received in state F. If an F-state cache line is copied, its state changes to S and the new copy gets the F state. This solves temporal locality problems, as the node holding the newest version of the data is the least likely to evict the cache line. If an array of cache lines is in high demand due to spatial locality, the ownership for these lines can be dispersed among several nodes. This enables spreading the bandwidth used to transmit the data across several nodes. The MESIF protocol provides a 2-hop latency for all common memory operations.

	Dirty?	**Unique?**	**Can write?**	**Can forward?**	**Can silent transition to:**	**Comments**
Modified	Dirty	Yes	Yes	Yes		Must write-back to share or replace
Exclusive	Clean	Yes	Yes	Yes	MSIF	Transitions to M on write
Shared	Clean	No	No	**No**	I	Does not forward
Invalid	–	–	–	–		Cannot read
Forwarding	Clean	Yes	No	**Yes**	SI	Must invalidate other copies to write

Table 2.4: Possible states for a cache line in the MESIF protocol, implemented by Intel ccNUMA processors [13].

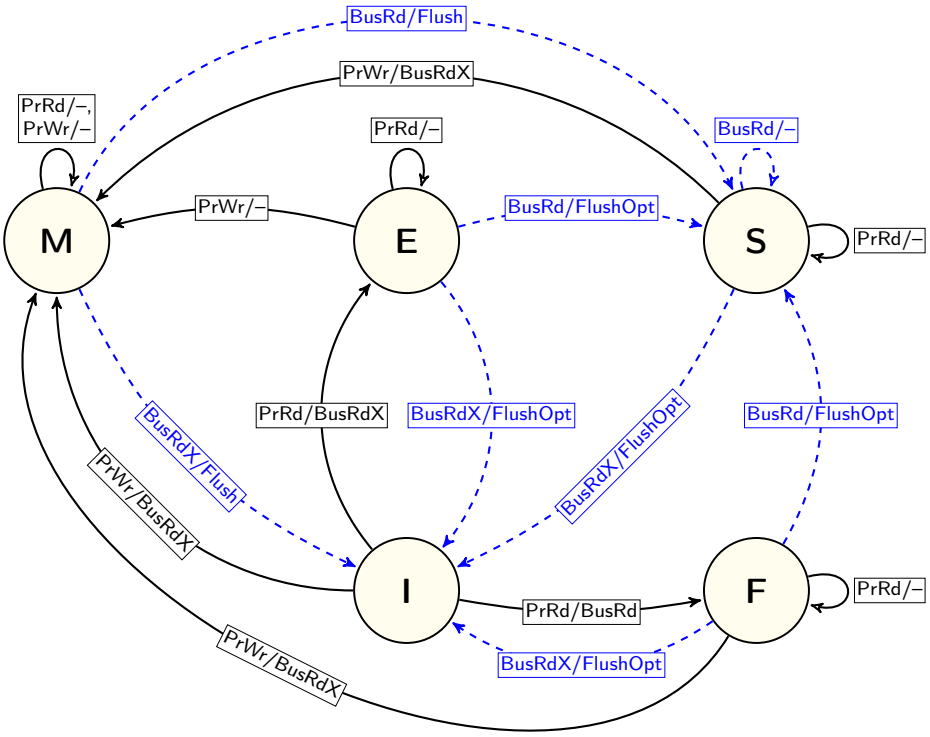


Figure 2.12: Finite state machine defined for a cache block in the MESIF protocol. Legend:

——➔: processor-initiated transaction.
- - -➔: snoop-initiated transaction.

Each slash-separated label specifies a transaction stimulus and its subsequent course of snoop action, respectively. Transaction stimuli:

PrRd: this processor (core) initiates a read request for a certain cache block.
PrWr: this processor initiates a write request for a certain cache block.
BusRd: snoop request indicating another processor is requesting to read the cache block. A request to main memory is made for the most up-to-date copy of this block. If another cache holds the most up-to-date copy of this block, it posts the block onto the bus (FlushOpt) and cancels the memory read request of the initiating processor. Next, the block is flushed to main memory as well.
BusRdX: snoop request indicating another processor, which does not hold the cache block yet, is requesting exclusive access to write to the cache block and obtain the most up-to-date copy as well. Other caches snoop this transaction and invalidate their potential copies of the block. A request to main memory is made for the most up-to-date copy of this block. If another cache holds the most up-to-date copy of this block, it posts the block onto the bus (FlushOpt) and cancels the memory request of the initiating processor.
BusUpgr: snoop request indicating another processor, which already holds a copy of the block, is requesting to write to the cache block.
Flush: snoop request indicating another processor is requesting to write the cache block back (flush) to main memory.
FlushOpt: snoop request indicating another processor has posted the cache block onto the bus in order to supply it to the other processors in a cache-to-cache transfer.

Example: core P1 requests a read (PrRd) of cache block c; since it does not hold this block, it resides in state I. Core P2's cache holds the most up-to-date copy of c in state F. P1 initiates a BusRd snoop request. P2 snoops this and flushes c to the bus, demoting its own state to S: (F) - - BusRd/FlushOpt -➔ (S). P1 acquires c and changes state to F, since it now holds the most up-to-date copy: (I) —— PrRd/BusRd ——➔ (F).

2.7 Present-day Intel many-core microprocessors

2.7.1 Knights Ferry (45 nm)

On May 31, 2010, Intel announced the first processor, codenamed Aubrey Isle (45 nm), for their new Many Integrated Core (MIC) prototypical architecture Knights Ferry. Initially intended to be used as a GPU, as well as high performance computing (HPC), the Knights Ferry prototypes only supported single precision floating point instructions. Unable to compete with AMD and Nvidia's contemporary models, Knights Ferry never made it to release. However, development continued into the MIC architecture Knights Corner.

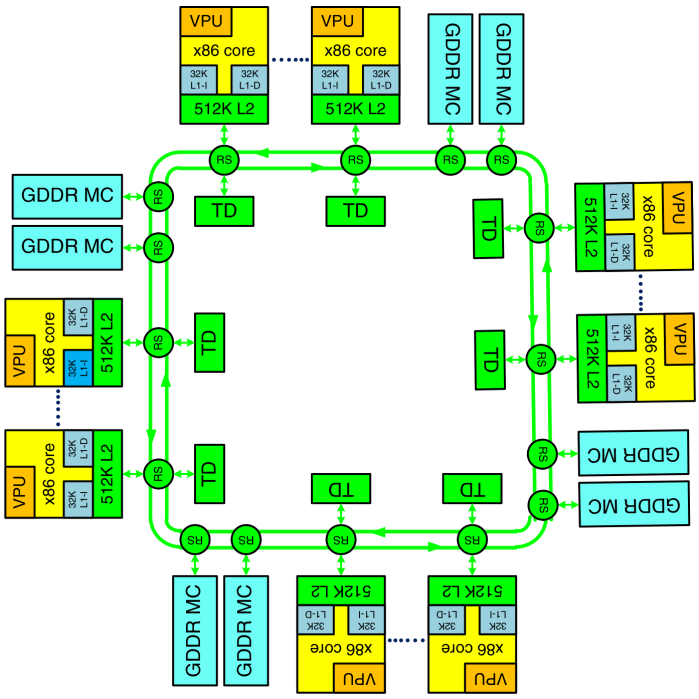


Figure 2.13: Block diagram of the Knights Corner interconnect architecture. The coprocessor features eight symmetrically interleaved memory controllers connecting the on-board GDDR5 memory, totalling 8 or 16 GB. Components are interconnected by a 512-bit high-bandwidth bidirectional ring, as indicated by the green paths. VPU: Vector Processing Unit, a 512-bit SIMD engine. RS: routing switch. TD: Tag Directory for L2 cache coherence. Image source: [15].

2.7.2 Knights Corner (22 nm)

The first Knights Corner (KNC) processor was released on November 12, 2012. The KNC product line, marketed as the Xeon Phi x100 series, exists of coprocessors, built at a 22 nm process size, ranging from 57 cores to 61 cores with quad-hyperthreading (four threads per core) at a TDP of 270-300 W. 6 to 16 GB of ECC (Error Correcting Code) GDDR5 memory is embedded on the package. Presently discontinued, KNC coprocessors were produced on either a PCIe 2.0 $\times$16 or an SFF 230-pin card. The cores of KNC coprocessors are based on a modified version of the original Pentium P54C microarchitecture. This makes KNC x86-compatible, allowing use of existing parallelization software.

The primary components of a KNC coprocessor, its processing cores, caches, memory controllers and PCIe client logic are interconnected by a 512-bit high-bandwidth bidirectional ring, as with the Haswell and Broadwell microarchitectures. Figure 2.13 illustrates the KNC interconnect architecture. The ring interconnect is comprised of four main rings: request, snoop, acknowledgement and 64-byte data. The memory controllers are symmetrically interleaved throughout the interconnect for more consistent routing.

On an L2 cache miss, an address request is sent to the corresponding tag directory on the ring. Next, a forwarding request is sent to either another core, if its cache holds the requested address, or to the memory

controllers. The cost of each data transfer on the ring is proportional to the distance between the source and destination, which in the worst case is in the order of hundreds of cycles [15]. This makes KNC's L2 cache miss latency an order of magnitude worse than that of multi-core processors.

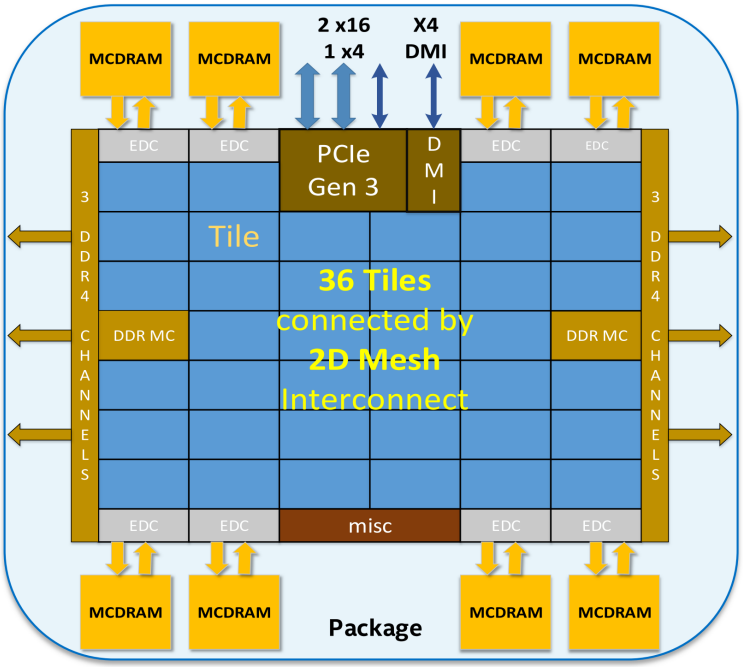


Figure 2.14: Block diagram of the Knights Landing interconnect architecture[8]. The 2D mesh interconnects up to 36 compute tiles, each of which contains two Intel Atom cores with two Vector Processing Units (VPUs) per core. Each pair of cores carries 1 MB of private L2 cache. EDC (Error detection and correction) denotes the eight memory controllers, each of which connects 2 GB of on-board MCDRAM. Two dedicated PCIe 3.0 16× links enable communication with the Omni-Path fabric at 100 Gb/s per link.

2.7.3 Knights Landing (14 nm)

The Knights Corner architecture was followed by Knights Landing (KNL) on June 20, 2016, with the release of the Xeon Phi x200 series processors. The KNL platform features three different configurations. Initially, KNL featured a PCIe add-on card coprocessor variant, similar to the preceding KNC architecture. The coprocessor variant never made it to the general market and was discontinued by August 2017. Intel opted to direct their focus with respect to HPC add-on cards onto their upcoming FPGA Programmable Acceleration Cards[9]. The other two variants make up the main line-up of KNL.

The standalone host processor variant can boot and run an OS, just like common CPUs. It features either 64, 68 or 72 quad-threaded Intel Atom cores at a TDP of 215-260 W. The 72-core microarchitecture runs at 1.5 GHz base clock frequency and is actually comprised of 76 cores, of which four are inactive. For two-core workloads, all models can boost to a turbo frequency, which adds 200 MHz to their base frequency. Workloads of three cores and over can only achieve a boost of 100 MHz and workloads with high-AVX SIMD instructions actually effectuate a frequency reduction of 200 MHz. In addition to the DDR4 main memory of the system, the KNL host processor has access to its on-board 16 GB of MCDRAM. The final KNL variant extends the standalone host processor variant with an integrated Omni-Path fabric.

In KNL, the QPI/UPI interconnecting other sockets is removed. This forces administrators of HPC-targeted KNL systems to interconnect multiple sockets via Omni-Path. Avinash Sodani, chief architect of the KNL chip at Intel, mentions that Intel's decision not to support multiple KNL sockets via UPI stems

[8]Image source: `https://www.mcs.anl.gov/petsc/meetings/2016/slides/mills.pdf` – accessed April 24, 2018.
[9]URL: `https://www.altera.com/solutions/acceleration-hub/platforms.html` – accessed April 24, 2018.

from the fact that snooping with the high memory bandwidth would easily swamp any UPI channel.[10] Omni-Path currently supports up to 100 Gb/s per port or link, which is about 22% faster than the theoretical maximum speed of Skylake's UPI. Inter-socket communication is placed outside of the chip, onto the Omni-Path fabric, connected through two dedicated PCIe 16× links. This way, existing I/O channels are freed, allowing for increased memory bandwidth at the cost of an increase in base latency, for which the decreased contention strives to make up. The dynamicity of the new Omni-Path fabric reduces the design complexity and cost of the inter-socket architecture. This enhances scalability in large multi-socket many-core HPC systems.

The KNL system is comprised of up to 36 compute tiles, interconnected in a 2D mesh, as illustrated in 2.14 on the preceding page. Each tile contains two Intel Atom cores with two Vector Processing Units (VPUs) and 32 KB of L1 cache per core. Each pair of cores carries 1 MB of private L2 cache, which is kept coherent by a distributed tag directory. An L3 cache is not included, since Intel found that targeted HPC workloads benefited less from it compared to adding more cores [16]. The MCDRAM has the option to function as an L3 cache, though, which is one of the three optional memory modes. In addition to this "cache mode", "flat mode" extends the physical address space of the main memory with physical addressable MCDRAM. "Hybrid mode" allows the MCDRAM to be split into one cache mode part and one flat mode part.

KNL employs the MESIF protocol and features a unique cache topology to minimize cache coherency traffic [16]. The L2 cache includes the L1 data cache (L1d), but not the L1 instruction cache (L1i). Lines filling L1i are copied to the L2 cache, but when those lines are evicted from L2 due to inactivity, the L1i copy is not invalidated. Additionally, each L2 cache line stores "presence" bits to track which of them are actively used in L1d.

The on-board MCDRAM has achieved over 450 GB/s of aggregate memory bandwidth in the Stream triad[11] benchmark [16]. This is substantially faster than KNC's on-board GDDR5 memory, which has a theoretical maximum bandwidth of 352 GB/s, though limited to a maximum achievable speed of approximately 200 GB/s due to the ring interconnect's bandwidth limitations plus the overhead of ECC. The MCDRAM also substantially outperforms the remote DDR4 memory, which has a theoretical maximum bandwidth of 102.4 GB/s.

KNC's adapted Pentium P54C cores are replaced in KNL with modified cores of the 14 nm Airmont microarchitecture. KNL is not only x86-compatible; it is binary compatible with prior Xeon processors as well. The Airmont microarchitecture delivers three times the peak performance or the same performance at five times lower power over previous-generation Intel Atom cores.[12] Each core operates out-of-order and is able to achieve peak performance at just one thread per core for certain applications, in contrast to KNC, which required at least two threads per core [17].

Knights Hill was planned to be the first 10 nm fabricated architecture in the Intel Xeon Phi series. After severe delays in manufacturing their 10 nm process, Intel cancelled the architecture in late 2017, in favor of an entirely new microarchitecture specifically designed for exascale computing, for which details are yet unknown.

2.8 Cost scalability of bus, point-to-point and NoC interconnects

Bolotin et al. analyze the generic cost in area and power of network-on-chip (NoC) and alternative interconnect architectures [18]. Cost assessments are summarized in Table 2.5 on the next page. The major deficit in scalability of the bus, as compared to a 2D $n \times n$ mesh NoC, is evident.

2.9 Networks-on-chip (NoCs)

A network-on-chip (NoC) is a packet-switching network embedded on a chip, typically interconnecting intellectual property cores in systems-on-chip. A NoC typically consists of routers, network interfaces

[10] URL: `https://web.archive.org/web/20150905141418/http://www.theplatform.net/2015/03/25/more-knights-landing-xeon-phi-secrets-unveiled/`.

[11] URL: `https://www.cs.virginia.edu/stream/` – accessed April 24, 2018.

[12] Source: `https://newsroom.intel.com/news-releases/intel-launches-low-power-high-performance-silvermont-microarchitecture/` – accessed April 23, 2018.

[13] Image source: `http://www.gem5.org/wiki/images/d/d4/Summit2017_garnet2.0_tutorial.pdf` – accessed May 19, 2018.

Interconnect	Power dissipation	Total area	Operating frequency
$n \times n$ **mesh NoC**	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Non-segmented bus	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n^3\sqrt{n})$	$\mathcal{O}\left(\frac{1}{n^2}\right)$
Segmented bus	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n^2\sqrt{n})$	$\mathcal{O}\left(\frac{1}{n}\right)$
Point-to-point	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n^2\sqrt{n})$	$\mathcal{O}\left(\frac{1}{n}\right)$

Table 2.5: Cost scalability comparison for various interconnect architectures [18].

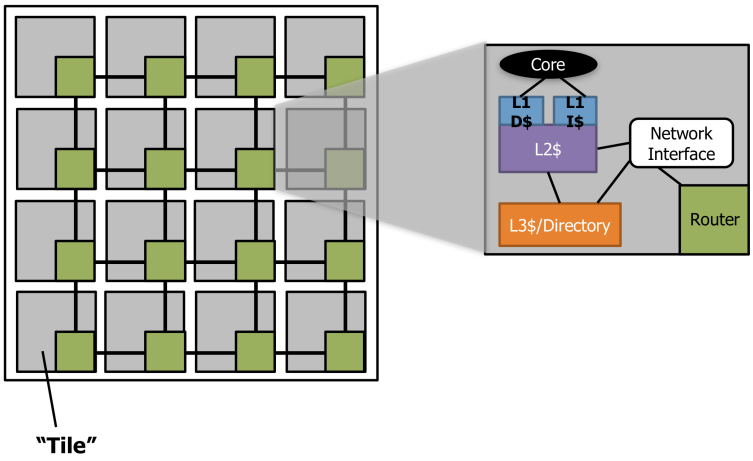


Figure 2.15: Topology of a typical NoC[13]. The topology defines how routers, network interfaces, coherency nodes and links are organized. The topology shown is a 4×4 mesh, composed of 16 tiles. A tile houses one router and in this example a single core and a single network interface.

(NIs) and links. Figure 2.15 illustrates a NoC topology, which defines how routers, NIs, coherency nodes and links are organized. A router directs traffic between nodes according to a specified switching method, flow control policy, routing algorithm and buffering policy. A network interface (NI) serves to convert messages between the different protocols used by routers and cores. Another imporant purpose of the NI is to decouple computation from communication, allowing use of both infrastructures independent of each other. A link is composed of a set of wires and interconnects two routers in the network. A link may consist of one or more logical or physical channels, each of which is composed of a set of wires.

The switching method defines how data is sent from a source to a destination node. Two main types of switching methods exist: circuit switching and packet (or flit) switching. In circuit switching, the complete path from source to destination node is established and reserved before actually starting to send the data. Preliminary setup increases latency overhead, but once the path is defined, throughput is enhanced due to not needing buffering, repeating or regenerating. Packet switching is the most common technique in NoCs. In packet switching, routers communicate through packets or flits. Flits (flow control units) are the atomic units that form packets and streams.

A flow control policy determines how packets move along the NoC and how resources such as buffers and channel bandwidth are allocated. For instance, deadlock-free routing can be established by commanding avoidance of certain paths. Virtual channels (VCs) are often used in flow control, to improve performance by avoiding deadlocks and reducing network congestion. VCs multiplex a single physical channel over several logically separate channels with individual and independent buffer queues. A deadlock is caused by a cyclic dependency between packets in the network, where nodes are waiting to access each other's resources. In livelock, packets do continue to move through the network, but they do not advance to their destination.

The routing algorithm determines for a packet arriving at a router's input port which output port to forward it to. The output port is selected according to the routing information embodied in the header of a packet. The buffering policy defines the number, location and size of buffers, which are used to enqueue packets or flits in the router in case of congestion.

Contrary to a bus, NoC nodes are connected by point-to-point wiring. Thus, for any network size, local performance is not degraded. Interconnect bandwidth is not shared by connected nodes, but actually aggregates with the size of the network. Since arbitration is distributed over the different routers, there is no aggregation of arbitration latency overhead, unlike in a bus. On the other hand, NoC design is complex, which incurs extra latency overhead due to decision making and makes for a more difficult implementation than a bus.

2.9.1 NoC performance

NoC performance evaluation can be broken down into the following two major categories: average packet latency and normalized network throughput. Latency (or delay) is the time elapsed from packet creation at the source node to packet reception at the destination node. Throughput is the total amount of received packets per unit time. Both latency and throughput are dependent on the applied traffic pattern and injection rate.

In addition to latency and throughput, NoC designers should consider fairness and Quality-of-Service. Also, variance in packet latencies and stability of the network when driven beyond saturation can impact performance significantly [19]. At high loads, packet queueing latency increases and a subset of packets can experience very high queueing latency, which is detrimental to latency-sensitive applications.

The average packet latency results from taking the sum of all average packet latency components and is generally recorded in unit cycles. In gem5/Garnet2.0 (detailed in Section 2.11 on page 29), for example, the packet latency components consist of the packet network latency, which is the packet travel time from source to destination, and the packet queueing latency, which is the sum of packet enqueue time and packet dequeue time. Packet enqueue time represents the time a packet has had to wait at the source node before being injected into the network. Packet dequeue time represents the time a packet has had to wait at the destination node before acknowledgement of its reception.

Since network throughput is an important performance metric for a NoC, one could conclude that the maximum network throughput provides a major metric for NoC peak performance. However, this does not take into account network contention and output contention. A more appropriate metric for evaluating NoC performance is the maximum sustainable network throughput, as this does take contention into account. The maximum sustainable network throughput is computed for the maximum continuous traffic load (injection rate) for which the average packet latency does not increase toward infinity.

Consequently, performance of a NoC is best represented by its latency-throughput relation. Figure 2.16 on the following page shows an alleged-typical network latency-throughput relation proposed by Ni [20] in 1996. The network pictured can achieve up to approximately 18% of the maximum network throughput. The plot shows that for this particular network, if traffic persists after the network has reached its maximum sustainable throughput, the throughput will decrease while the latency increases. The synthetic traffic simulations detailed in 4.2.1 on page 45 do not coincide with this plot. The bit-complement traffic pattern most closely resembles the curve in Figure 2.16, but it has an asymptotic normalized network throughput. Other literature, such as [21] (Figure 2.17 on the following page) and [22], coincides with my findings. Ni does not specify the configuration of the network pictured. Ni's model specifies "normalized network throughput' as the sustained network throughput normalized to the maximum network throughput. The maximum network throughput is a constant, though. If Ni meant the maximum network throughput to represent the injection rate, the model is not valid either. Therefore, I am inclined to refute Ni's proposed latency-throughput model.

Exceeding maximum sustainable network throughput results in oversaturation of the network. Routing and flow control methods should be designed to avoid such oversaturation. Ideally, a NoC should avoid saturation altogether, although the ideal maximum packet latency depends on the latency-senstivity of the application and power constraints.

2.9.2 NoC power and area

Ganguly et al. specify the inter-switch wire length l for 2D mesh architecture as given by (2.5):

$$l = \frac{\sqrt{\text{Area}}}{\sqrt{M} - 1} \tag{2.5}$$

where "Area" denotes the area of the silicon die used and M is the number of intellectual property blocks [24].

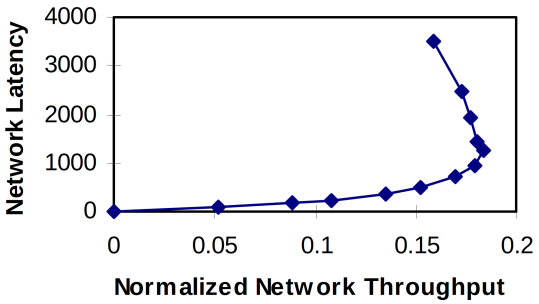


Figure 2.16: Questionable network latency-throughput relation by [20]. "Network latency" is the average packet latency in cycles. "Normalized network throughput" is the sustained network throughput in packets per cycle, normalized with respect to the maximum network throughput. This particular network can achieve up to approximately 18% of the maximum network throughput.

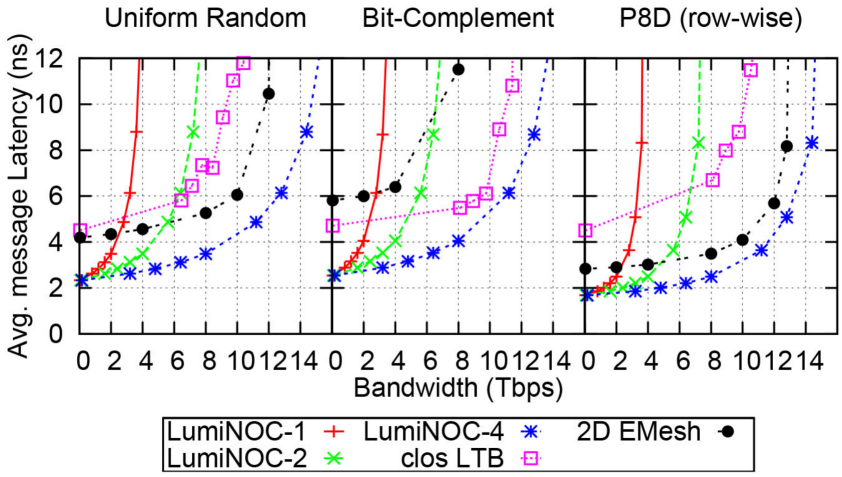


Figure 2.17: Performance comparison of the following 64-node NoCs under uniform random, bit-complement and P8D synthetic traffic loads: nano-photonic LumiNOC (with 1, 2 or 4 network layers), conventional electrical 2D mesh and Clos LTBw (low target bandwidth) [21].

2011 International Technology Roadmap for Semiconductors (ITRS) projects that, as process size shrinks, global interconnect wire delay keeps scaling up, in the order of nanoseconds, while gate delay keeps scaling down, in the order of picoseconds [2]. Placing repeaters can substantially mitigate the delay, but these will increase the power and area of the chip.

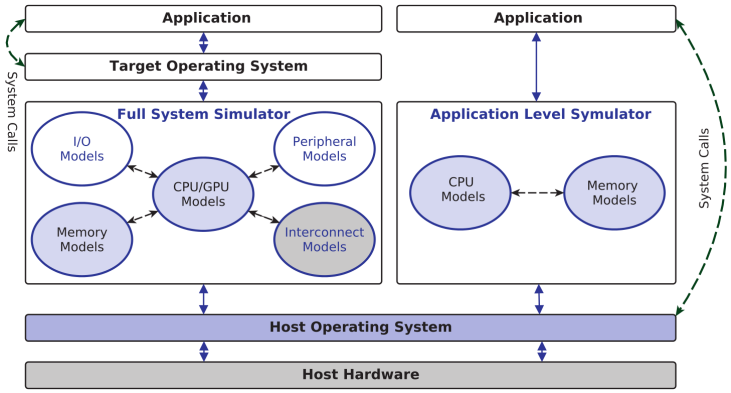


Figure 2.18: Full-system versus application-level computer architecture simulator. Full-system simulators can simulate a full-fledged OS plus simulator-agnostic applications, whereas application-level simulators emulate simulator-customized applications on the host OS. Image source: [25].

2.10 Computer architecture simulators

2.10.1 Overview

Due to power constraints, emerging multi-core and many-core designs continue to become more complex and a plethora of architecture configurations and optimizations require exploration. A competent computer architecture simulator offers a valuable tool for comparing scalability of – both existing and hypothetical – microarchitectural configurations. The relative competency of a simulator depends on the complexity of the configurable feature set and the accuracy of results with respect to aspects relevant to the research at hand. Both high complexity and high accuracy naturally come at the cost of simulation times. In order to evaluate the scalability of the performance, power and area trinity, integration of a competent network interconnect simulator in employed computer architecture simulators is essential.

Computer architecture simulators can be classified into two categories: full-system simulators and user/application-level simulators. Figure 2.18 on the previous page illustrates the difference between these categories. A full-system simulator can run a complete, real software stack on the simulated system, without modification. Thus, a full-fledged OS can be simulated and the performance results take overheads of the OS and background applications into account. Application-level simulators emulate an application on the host OS. The main advantages of application-level simulators are that they are easier to design and update than full-system simulators and they put a significantly lower footprint on the host system, which leads to much faster simulation times. Application support for these simulators needs to be explicitly built in, which marks a major disadvantage. Also, performance of certain applications is severely affected by OS overhead, especially in the case of large I/O requests, which can lead to unrealistic results.

2.10.2 Summary of selected computer architecture simulators

Table 2.6 on page 30 shows a comparative overview of various many-core architecture simulators considered for this thesis' experiments. Both Sniper and gem5 were selected, for reasons explained in the following section. Chapter 4 on page 42 presents results for experiments carried out using the selected simulators.

Sniper [26] is a high abstraction level x86 simulator, which emulates execution on the host architecture. Sniper extends the Graphite simulator [27] and integrates McPAT [28] for modeling of power, area and timing. Heirman et al. compared the performance accuracy of Sniper and the accompanied power model accuracy of McPAT to real hardware. They report average performance and power accuracy mismatches of 22.1% and 8.3%, respectively, for a set of SPEComp benchmarks.

The gem5 simulator sprung from the merging of previous simulators M5 [29] and GEMS [30]. gem5 is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. gem5 is written partially in Python and C++. This combines the flexible and succinct high-level scripting of Python with the high performance of C++. A disadvantage of this is that building certain code extensions, affecting files interlinking the programming languages, requires writing large amounts of code. gem5 includes the detailed NoC simulator Garnet2.0. gem5 also includes DSENT (Design Space Exploration of Networks Tool) for cross-hierarchical area and power modeling for both optical and electrical NoCs [31]. In 2012, Butko et al. evaluated the performance accuracy of gem5 full-system mode, by comparing simulation results for a wide range of benchmarks to results obtained on real hardware [32]. They report an accuracy mismatch varying from 1.39% to 17.94%, depending on the memory traffic. Butko et al. pinpoint gem5's overly simple DDR memory model as the cause for the largest accuracy discrepancies.

2.10.3 Motivation for selected simulators

This thesis aims to model scalability for existing Intel x86 interconnects and future alternative interconnects. For the former purpose, Sniper was selected, since it is well tailored to simulation of various modern Intel x86 microarchitectures due to its extensive high-level configurability. Also, a sizeable amount of documentation and acadademic literature on Sniper exists, as well as excellent developer support on its discussion forums[14]. For alternative interconnect evaluation, Sniper proved too restrictive. Sniper

[14] Sniper discussion forum URL: `https://groups.google.com/d/forum/snipersim` – accessed June 1, 2018.

supports a bus, ring, mesh and torus topology, but NoC configurability and statistics are limited and modification would require a large amount of framework extensions. Wim Heirman, one of Sniper's creators, notes that NoC-centric design studies are better performed on an alternative, detailed NoC simulator[15]. Instead, for scalability evaluation of various NoC topologies, the Garnet2.0 NoC simulator, included with gem5, was selected.

Garnet2.0 was selected for its canonical integration with the comprehensive and robust gem5 computer architecture simulator, its modernity (2016), its detailed NoC modeling and its modularity of topology specification. Additionally, its well-factored and well-documented code suggests robustness. Section 2.11 outlines the Garnet2.0 NoC simulator in detail.

A viable alternative, the TOPAZ NoC simulator [33] also natively supports integration with gem5 since its release in 2012. Unlike Garnet2.0, TOPAZ natively supports various complex routers, such as adaptive bubble and deterministic bubble, various types of flow control and the following topologies: ring, mesh (2D and 3D), torus (2D and 3D), midimew (2D) and square midimew (2D). Akin to Garnet2.0, various synthetic traffic patterns are included. Nevertheless, initial familiarization with Garnet2.0 proved it to be adequate for the scope of this thesis. Exploration of TOPAZ was therefore foregone in favor of Garnet2.0, which is more recent and subjectively has a more succinct framework.

2.11 gem5/Garnet2.0 - NoC simulator

2.11.1 Infrastructure overview

Garnet2.0 provides a detailed cycle-accurate NoC model inside gem5. Released in 2016, it builds upon the original Garnet model [37]. Modeled networks allow configuration of the number of virtual networks (VNET), control message size, flit size, the number of virtual channels (VCs) per VNET, routing algorithm and the number of data and control flit buffers per VC. Any heterogeneous topology can be modeled through a Python script. Garnet2.0 extends gem5's "ruby" memory system model, which contains parent classes for Topology and Routing. Due to ruby's memory address range implementation, the number of directory controllers in a topology is limited to powers of two.

The `GarnetNetwork` class instantiates the main network components: network interfaces (NIs), routers, network links and credit links. A network link can either be an "external link", which connects a single router to one or more NIs, or an "internal link", which interconnects a pair of routers. Network links carry flits; by default, a control protocol message is put into a single flit and data messages are five flits wide. A credit link carries VC buffer credits between routers for flow control. A latency in cycles can be assigned to each router, internal link and external link.

The `GarnetNetwork` class calls the `Topology` class, which defines the routers, internal links and external links. Figure 2.19 on page 31 illustrates the infrastructure of a Garnet2.0 topology. Internal links are constructed unidirectionally and external links are constructed bidirectionally. In addition to latency, internal links can be specified a weight for dimension order routing, as well as a port direction to facilitate customized routing. Nodes connected through internal links can include cache controllers, directory controllers and DMA controllers. Each network interface connects to one coherence controller. gem5 handles the modeling of CPUs and coherence controllers. Each CPU is connected via its private L1 cache controller.

The `Router` class houses one routing unit, switch allocator, crossbar switch and for each port one input or output unit. Figure 2.20 on page 31 shows a diagram of the architecture. The following stages are performed by the router, in order. Buffer write: the input unit buffers the incoming flit in its VC. Route compute: the routing unit computes the output port for the buffered flit. VC allocation: the switch allocator arbitrates the input ports by selecting a VC from each input port, in a round robin manner. Switch allocation: the switch allocator arbitrates the output ports by selecting one input VC as the winner for each output port, in a round robin manner. Buffer read: the flits that won switch allocation are pulled from their respective input units. Switch traversal: flits that won switch allocation traverse the crossbar switch. Link traversal: flits traverse links from the crossbar to their target routers. By default, all stages except for link traversal are performed in one cycle. Link traversal occurs in the next cycle.

[15] URL: `https://groups.google.com/d/msg/snipersim/QV2ycYYsL0k/P7Cbf2Du4hsJ` – accessed June 1, 2018.

[16] Image source: `http://www.gem5.org/wiki/images/d/d4/Summit2017_garnet2.0_tutorial.pdf` – accessed May 19, 2018.

Simulator	**Graphite** [27]	**gem5** [34]	**PriME** [35]	**Sniper** [26]	**ZSim** [36]
Simulation scope	Application-level	Full-system (FS mode), application-level (System-call Emulation, SE mode)	Application-level	Application-level	Application-level
Scalability	High	Poor	Very high	High	High
Speed	Moderate to very fast, depending on synchronization scheme	Very slow	Very Fast	Fast	Fast
Complexity of setup	High	Low	Low	Low	High
Accuracy	Aims to imitate cycle-accuracy with slowest synchronization scheme	Quasi-cycle accurate: by means of ISA simulation	Not cycle accurate: one-CPI and profiling-based core models	Not cycle accurate: interval core model	Not strictly cycle accurate: instruction-driven timing models
Parallelization	Pthreads	Sequential	MPI	Pthreads	Bound-weave
Many-core support	Up to 1024 cores	FS mode: up to 64 cores; SE mode: limited support for 256 to 1024 cores	Up to 2048 cores	Up to 1024 cores	Up to 1024 cores
Supported cache coherency protocols	Directory-based MSI and MOSI	Directory-based MESI, AMD MOESI, generic MOESI, token MOESI	Bus-based and directory-based MESI	Directory-based MSI, MESI and MESIF	Directory-based MESI
Modeling features	Core models, memory subsystems, NoCs	Core models, pipelined model, memory subsystems, complex NoCs, system execution modes	One CPI and profiling-based core models, memory subsystems, 2D mesh NoC	Interval and instruction window-centric in-order, out-of-order and SMT core models, NoCs, memory subsystems	Detailed DBT-accelerated core models, memory subsystems
Supported ISAs	x86	ARM, ALPHA, MIPS, Power, RISC-V, SPARC, x86 and ISA-agnostic	x86	x86	x86
Supported Workloads	Pthread applications	Multi-threaded applications in SE mode and a variety of workloads in FS mode	MPI applications	Most modern and complex workloads, including Pthread-based multi-threaded applications	Most modern and complex workloads, including multi-threaded applications

Table 2.6: Comparative overview of various many-core architecture simulators. Analytical assessments loosely based on [25].

2.11.2 Capability assessment

As touched on in Section 2.10.3 on page 28, Garnet2.0 provides a presumably robust and well-documented framework for detailed NoC simulation. Still in its infancy, it is lacking many features, such as adaptive routing techniques and optimization techniques, such Dynamic Voltage and Frequency Scaling (DVFS). Routers and links have a cycle-accurate latency, but each link is assumed to be a straight edge of unknown length, apparently tailored to grid-based topologies. Moreover, the number of included topologies

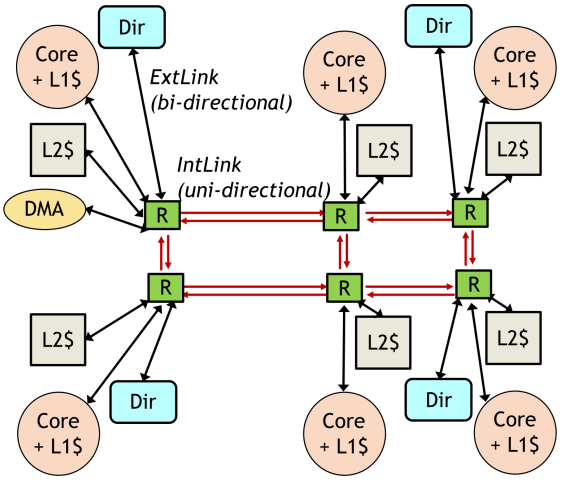


Figure 2.19: Diagram illustrating the Garnet2.0 topology infrastructure[16]. Topology pictured: 2×4 mesh. Implemented in `MeshDirCorners_XY`, included with Garnet2.0. Six routers interconnect six cores, each of which has one L1 and one L2 cache controller. Four directory nodes are distributed over the corners of the mesh.

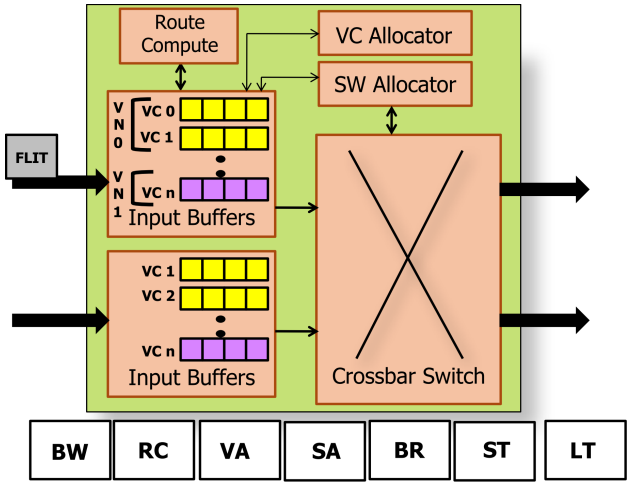


Figure 2.20: Diagram of the Garnet2.0 router architecture[16]. The router performs the following actions in order: **BW**: Buffer Write. **RC**: Route Compute. **VA**: VC Allocation. **SA**: Switch Allocation. **BR**: Buffer Read. **ST**: Switch Traversal. **LT**: Link Traversal.

is remarkably limited, as the only included well-scalable topology is the 2D mesh.

Garnet2.0's code is well factored for out-of-the-box operation, which has its disadvantages. For example, the `RoutingUnit` class, in which routing algorithms are implemented, is not VC-aware. It contains a pointer to its parent `Router`, which in turn references its `InputUnit` and `OutputUnit` classes holding the VC states for the router's ports, as well as a pointer to the `GarnetNetwork` class, which references all routers in the network. By default, Garnet2.0 only supports shortest path routing, which is leveraged from its parent ruby `Topology` and `Routing` infrastructures. Thus, implementing sophisticated adaptive routing techniques require a lot of code extensions and possibly code refactoring, which can interfere with other network models depending on the leveraged parent modules.

The performance of both gem5 and Garnet2.0 scales badly with an increase in the number of modules and nodes, respectively. gem5 supports only sequential operation and requires a lot of working memory. For this reason, many-core gem5/Garnet2.0 experiments, detailed in Section 4.2 on page 45, are carried out for a minimal amount of cycles: 20,000. Garnet2.0's accuracy has not been evaluated in presently existing literature. The ascertained consistency of the results of my experiments does suggest adequate accuracy. Regardless, Garnet2.0 does present a prime candidate for scalability evaluation of customized

topologies, due to its. More details on Garnet2.0 can be found in the gem5 wiki linked in the footnote[17].

2.11.3 Synthetic on-chip network traffic simulations

Garnet2.0 provides a framework for simulating synthetic traffic within a Garnet network. The Garnet synthetic traffic injector (GSTI) uses an ISA-agnostic coherence protocol, called Garnet_standalone. Thus, it can be used to evaluate the theoretic capabilities of a constructed Garnet network. A list of GSTI parameterized options can be found in the gem5 wiki[18].

The following is a summary of the packet generation procedure of GSTI. Every cycle, each CPU performs a Bernoulli trial with probability equal to the set injection rate to determine whether to generate a packet or not. The injection rate is defined as the number of packets per node per cycle. All randomization is performed deterministically. If the CPU is to generate a new packet, the packet destination will be computed based on the set synthetic traffic pattern. GSTI supports the following synthetic traffic patterns: uniform random, tornado, bit complement, bit reverse, bit rotation, neighbor, shuffle and transpose. Packet destination is embedded into the bits after block offset in the packet address. The packet is randomly tagged as either a read, write or instruction fetch request. The packet is sent to the Garnet_standalone cache controller. The cache controller extracts the destination from the packet address and injects it into the appropriate VNET. Read requests (control protocol) are injected into VNET 0, instruction fetch requests (control protocol) are injected into VNET 1 and write requests (data protocol) are injected into VNET 2, The packet traverses the network and once it has reached the directory controller, it is dropped. The injector terminates after the set amount of simulation cycles has elapsed.

2.11.4 Deadlock detection

A high recorded latency value is congruent with a high `vc_busy_counter` value, which is recorded for each virtual network for each router. The `vc_busy_counter` is reset when any output VC resides in the idle state for one cycle. If any `vc_busy_counter` reaches the predefined deadlock threshold, the simulation is aborted. By default, the deadlock threshold is set to 50,000. This means that if any `vc_busy_counter` reaches 50,000, a deadlock is detected and the simulation is aborted. However, it is possible that the system is not in deadlock, but in livelock. In this case, flits keep moving along the virtual network, without reaching their destination.

[17] Garnet2.0 overview: `http://www.gem5.org/Garnet2.0` – accessed May 29, 2018.
[18] URL: `http://www.gem5.org/Garnet_Synthetic_Traffic` for gem5/Garnet2.0 – accessed May 29, 2018.

CHAPTER 3

My work

3.1 Sniper x86 simulator – configured architectures

Sniper provides a reasonably detailed list of configurable microarchitecture parameters. For instance, the reorder buffer (ROB) can be configured as in-order or out-of-order and the numbers of ROB outstanding loads/stores and reservation states can be specified. For L1, L2 and L3 caches, associativity, block size and both data and tag access times can be specified, among numerous other parameters. Additionally, latencies for static instructions can be configured. For each newly configured microarchitecture, all configurable parameters are filled insofar as their values were able to be found from respectable internet sources.

Microarchitecture parameters were copied from Agner Fog's microarchitecture of Intel, AMD and VIA CPUs optimization guide[1], among other sources. Instruction latencies were copied from Agner Fog's lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs[2]. Configured parameters can be found in the relevant `sniper/config/*.cfg` files in this project's source code.

Sniper configuration files were created for Haswell and Knights Landing microarchitectures. Since the FFT benchmark used in the experiments restricts core counts to powers of two, Haswell is configured for 4, 8, 16, 32 and 64 cores and Knights Landing is configured for 64 and 128 cores.

Various Haswell configuration files were created for different interconnects, in order to evaluate interconnect scalability of this architecture. The shared L3 cache of 8 MB is modeled by Snipers implementation of NUCA[3] slicing. The different topologies modeled for Haswell are: ring, bus, mesh and concentrated mesh. The segregated dual ring interconnect employed in actual mid-to-high-core-count Haswell CPUs is not modeled. In ring topologies, one DRAM controller is placed per 12 cores, corresponding to the actual Haswell architecture. Bus topologies are assigned a single DRAM controller, which is conventional. Mesh topologies are assigned two DRAM controllers, corresponding to Skylake and Knights Landing architectures. These experiments are laid out in Section 4.1.1 on page 42.

For the Knights Landing architecture, effects of scaling its core count from the actual 64 cores to 128 are evaluated. To this end, two configuration files were created for the 128-core model: the first is configured for the 64-core model's actual clock frequency of 1.3 GHz and supply voltage of 1.125 V, the second is configured for a clock frequency of 800 MHz and supply voltage of 0.85 V. All current available versions of McPAT are limited to 22 nm technology models. Consequently, power and area models for the Knights Landing architecture are askew.

[1] URL: `http://www.agner.org/optimize/microarchitecture.pdf` – accessed May 25, 2018.
[2] URL: `http://www.agner.org/optimize/instruction_tables.pdf` – accessed May 25, 2018.
[3] Non-Uniform Cache Access (NUCA), introduced in [38], facilitates splitting the cache into a large number of banks and employing a NoC to allow fast access to nearby banks. A multitude of varieties based on the original NUCA structure exists in literature.

3.2 Built gem5/Garnet2.0 extensions

Work started on the latest gem5 source code[4] as of April 27, 2018. The extensions listed below refer to this version of gem5 and Garnet2.0.

3.2.1 Added framework assist scripts

`./buildgarnet`: The Garnet_standalone simulation binaries (for use with the Garnet synthetic traffic injector) can be built by running this Bash script, which executes: `scons -j`*n* `build/NULL/gem5.debug PROTOCOL= Garnet_standalone` where *n* is the number of cores on the host machine.

`./buildx86`: The x86 simulation binaries (for use with both system-call emulation mode and full-system mode) can be built by running this Bash script, which executes: `scons -j`*n* `build/X86_MESI_Two_Level/gem5.fast PROTOCOL=MESI_Two_Level` where *n* is the number of cores on the host machine. The simulations will employ two cache levels with the MESI coherence protocol.

`./rungarnet`: The Garnet_standalone binaries and added extensions can be executed more conveniently by running this Bash script. The script calculates topology-specific parameters and generates a uniquely formatted output directory name within the `m5out` directory. Some parameters can be supplied from command line arguments; others have to be modified by editing the script. The script prints usage information by running it with no arguments.

`./runfft`: Runs the Splash2 FFT benchmark [39] in gem5 system-call emulation mode with a Garnet2.0 network, which can be specified in similar fashion to `./rungarnet`. Problem size M can be specified as the fifth parameter and clock frequency for all CPUs can be specified as the sixth parameter. The FFT benchmark will use 2^M complex doubles as data points. M must be an even number.

`./rundsent`: Takes one or more arguments, each of which should specify a simulation output directory. For each output directory, the script runs `util/on-chip-network-power-area-2.0.py` and writes the output to `dsent_out.txt` within the output directory.

`./grepnetworkstats.py`: Automatically run by `./rungarnet` and `./runfft`. Copies the relevant network statistics from the simulation's `stats.txt` to `network_stats.txt`.

`./grepdebug.py`: If `GARNETDEBUG=1` is set in `rungarnet`, is automatically run after a Garnet_standalone simulation. Lines matching the specified `grepdebug.py:greps[]` are copied from `debug.txt` to `debug_parsed.txt`. Optionally, a sorted output is written to `debug_parsed_sorted.txt`.

`./plotlatencythroughput.py`: Takes one argument: the root directory containing multiple simulation output directories. For each injection rate, gathers the reception rate (throughput) and the average packet latency from the simulation's `stats.txt`. The three statistics are appended to respectively formatted `*-latencythroughput.txt` files within the specified root directory.

3.2.2 Splash2 FFT benchmark for SE-mode x86 simulations

Newly implemented in `configs/example/fft_benchmark.py`, borrowing code from gem5's included `configs/example/se.py` and `configs/splash2/run.py`. The Splash2 benchmarks require compilation with a Pthreads implementation of the PARMACS macros. Unable to get this to work, I resorted to copying the Splash2 benchmarks included with the Sniper simulator to the `sniper_splash2` directory. Defunct hooks used by Sniper were removed, after which the FFT benchmark is working at expected. The `./runfft` script provides a convenient command-line option to specify primary Garnet2.0 topology parameters, as well as the FFT benchmark's problem size and the simulated CPU clock frequency.

[4] URL to the gem5 repository: `https://github.com/gem5/gem5`; URL to the latest commit used: `https://github.com/gem5/gem5/commit/5187a24d496cd16bfe440f52ff0c45ab0e185306` – accessed May 19, 2018.

3.2.3 Topology visualization with LaTeX/TikZ

Newly implemented in `configs/topologies/TikzTopology.py`. If parameter `--tikz` is used, will write LaTeX/TikZ code for visualizing a topology to `topology.tex` in the simulation's output directory. Example visualizations are shown in the following section. Links with a weight of 1 get a thicker edge than links with higher weight (lower order), to emphasize dimension order routing. For flattened butterfly topologies with more than 64 routers, only the first 64 routers' edges are drawn, due to LaTeX memory limitations.

`./tex2png`: Automatically run from `./rungarnet` and `./runfft`. Converts `topology.tex` in the given output directory to PDF. If the `imagemagick` package is installed, the topology PDF is converted to `topology.png` as well. If `CLEAN_UP=1` is set, the LaTeX files will be removed, keeping only the PNG-file.

3.2.4 Topologies

A new parameter `--concentration-factor=`n was added, for all of the implemented topologies listed below. n denotes the number of CPUs per router. Each CPU is connected through one L1 cache controller node. The number of directory nodes can be specified with `--num-dirs=`d, with $d \leq$ the number of cache nodes. The directory nodes are distributed evenly across the routers. All DMA nodes are connected to the first router – Garnet_standalone does not employ any DMA nodes. Each topology is limited to two dimensions and each link is bidirectional.

Line

Newly implemented in `configs/topologies/Line.py`. Mimics a bus interconnect within a NoC. Constructed line topologies comprise a $1 \times n$ grid, where n is the number of routers.

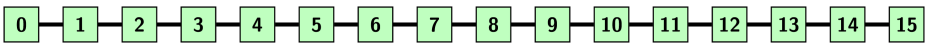


Figure 3.1: Diagram of the 16-router line topology.

Fully Connected

Newly implemented in `configs/topologies/FullyConnected.py`. Each router is connected to all other routers. Each link has the same dimension order routing weight of 1, since fully connected topologies inherently avoid deadlock [40]. Each link that is a straight edge is assigned a latency of 1 cycle and the same distance of l (square dimensions; 4 routers) or l' (odd dimensions; 8 routers and up) in DSENT link power and area modeling (see 3.2.5 on page 38). This implementation consequently restricts fully connected topologies to two mesh rows (grid rows). More elaborate implementations are forgone due to the inherent dire scalability of fully connected topologies, as well as time constraints. Each sloped link is assigned a latency of `ceil(proportional_distance)`, where `proportional_distance` is the distance proportional to a straight edge. Sloped links are set to distance of l or $l' \times$ `proportional_distance` in DSENT.

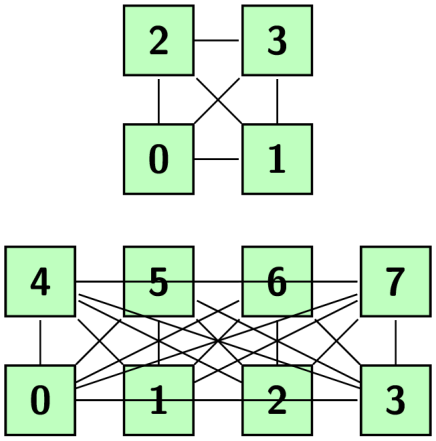


Figure 3.2: Diagrams of fully connected topologies for 4 and 8 routers. Each router is connected to all other routers. Links spanning more than two routers are omitted from the diagram. Each link has the same dimension order routing weight of 1, since fully connected topologies inherently avoid deadlock.

Ring

Newly implemented in `configs/topologies/Ring.py`. Constructed ring topologies comprise a $2 \times n$ grid, where n is half of the number of routers. The optional escape VC deadlock avoidance scheme, detailed in 3.2.7 on page 41, is not functioning correctly; many workloads will still deadlock. Therefore, this topology's implementation is deficient.

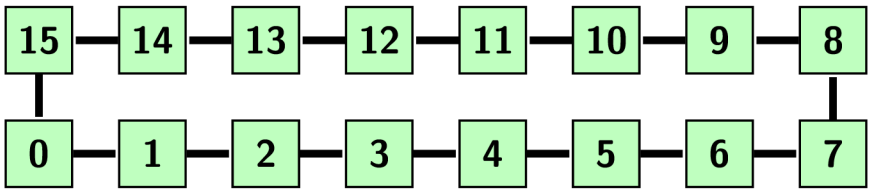


Figure 3.3: Diagram of the 16-router ring topology.

Hierarchical ring

Newly implemented in `configs/topologies/HierarchicalRing.py`. No microarchitectural differentiation between the central ring and any of the sub-rings is implemented in Garnet2.0. No deadlock avoidance scheme is available. Therefore, this topology's implementation is deficient. For configurations with a number of mesh rows (grid rows) greater than 4, the number of directory nodes is limited to the number of mesh rows and the number of cores is limited to 128, due to unresolved deadlock issues.

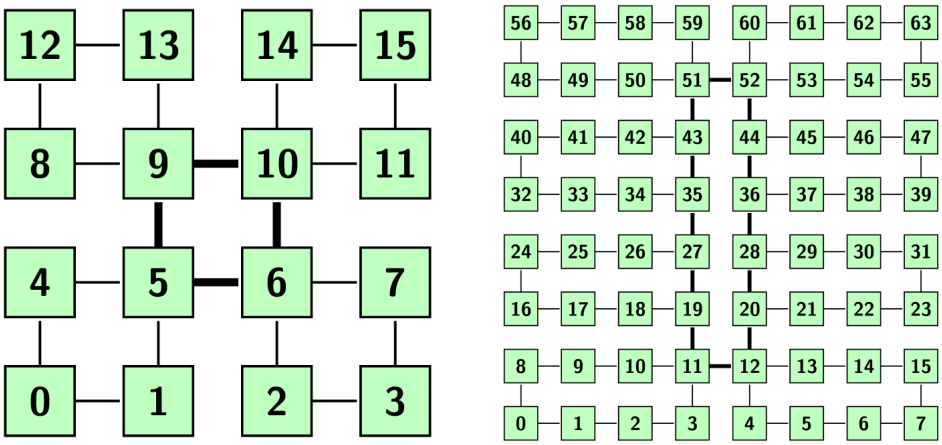


Figure 3.4: Diagrams of hierarchical ring topologies for 16 and 64 routers. The central ring connects multiple sub-rings.

Mesh and concentrated mesh

Mesh with XY dimension order routing is included with Garnet2.0 in `configs/topologies/Mesh_XY.py`. The topology's source code was modified to allow for a concentration factor, variable amounts of cache and directory nodes and TikZ visualization. XY dimension order routing is enforced by an assigned weight of 1 for each horizontal link and an assigned weight of 2 (lower precedence) for each vertical link.

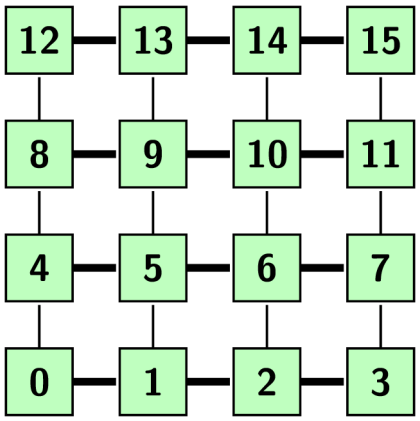


Figure 3.5: Diagram of the 16-router mesh topology. With a concentration factor of 4, the diagram shows a 16-router 64-core concentrated mesh topology.

Flattened butterfly

Newly implemented in `configs/topologies/FlattenedButterfly.py`. Each row and each column is fully connected. The flattened butterfly debut paper proposes a concentration factor of $n = 4$ for 64 cores [41]. For $n = 4$, each router has a radix of 10: each router connects to 3 neighboring routers in the x-dimension, 3 neighboring routers in the y-dimension and 4 cores. This implementation allows other concentration factors as well. The repeaters and pipeline registers for links connecting non-neighboring routers, suggested by the flattened butterfly paper, are not implemented. The repeaters and pipeline registers for these links suggested by the flattened butterfly paper are not implemented. Instead, the latency of these links is multiplied by the proportional distance between routers.

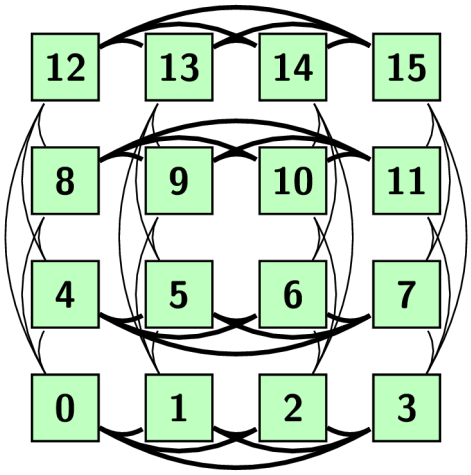


Figure 3.6: Diagram of the 16-router flattened butterfly topology. Each router can be connected to n CPUs, where n is the concentration factor. For the 4×4 2D mesh pictured, the flattened butterfly debut paper suggests concentration factors of $n = 4$ for 64 cores and $n = 8$ for 128 cores [41]. Links are drawn curved purely for visualization; the implementation assumes straight edges for all links.

3.2.5 DSENT – power and area modeling

NoC power and area models are generated by DSENT, version 0.91 (June 26, 2012), included in the gem5 commit referenced in footnote 4 on page 34.

`configs/topologies/TopologyToDSENT.py`: Contains a new class `TopologyToDSENT`, which is called within a topology's class. It writes topology-specific values to both a router and a link configuration file, `router.cfg` and `electrical-link.cfg` respectively, in the simulation's output directory. These values include the numbers of control buffers (pertaining to VNETs 0 and 1) and data buffers (pertaining to VNET 2), as well as the number of bits per flit. These configuration files serve as a scheme, to be later updated by `util/on-chip-network-power-area.py` for each individual router and link.

The router configuration file contains parameters and equations for calculating router power and area, based on statistics passed by the Python script. Similary, the link configuration file is used to calculate link power. To estimate the total area of a router, the release paper of ORION 2.0, a NoC power and area modeling tool, suggests taking the sum of the router's building blocks and adding 10% to account for global whitespace [23]. This factor is implemented in the router configuration file.

`util/on-chip-network-power-area-2.0.py`: Based on `util/on-chip-network-power-area.py`, which was outdated (2014) and included with gem5/Garnet2.0. The original script feeds limited Garnet1.0 simulation statistics to DSENT for modeling of NoC power and area. The resulting models assume a global injection rate for all routers and links. These limitations are eliminated in the newly created version 2.0 script, which is geared towards parsing of Garnet2.0 statistics. Passing of router- and link-specific arguments to DSENT was not functioning correctly for some arguments. This is fixed by instead running `sed -i` shell commands. The script is to be run with one argument: the simulation's output directory.

gem5's canonical implementation of DSENT defines the number of input ports of a router as the number of ports connecting unidirectional internal links (interconnecting a pair of routers). Similarly, the number of output ports are defined as the number of ports connecting unidirectional external links (between a router and NI-connected cache and directory controllers). The discrepancy with Garnet 2.0's bidirectional external links is accounted for. As explained in Section 2.11.1 on page 29, credit links carry VC buffer credits only and their influence on power and area results is deemed negligible. Therefore, the links considered in DSENT power and area modeling are limited to network links.

The injection rate in gem5's canonical implementation of DSENT is defined as the number of flits per cycle per port or link, in contrast to the Garnet synthetic traffic injector, which defines it as the number of packets per node per cycle. Injection rates for the flit buffers, crossbar and switch allocator are updated for each router, based on simulation statistics. Similarly, both the injection rate and wire length for each

link are updated. Individual injection rates are used in calculation of dynamic power.

DSENT includes four electrical technology models, all of which are LVT (Low Voltage Threshold): “bulk”, at 45 nm, 32 nm and 22 nm process sizes and tri-gate (multi-gate), at a 11 nm process size. The 11 nm tri-gate technology model is used in all experiments detailed in Chapter 4 on page 42, since this model represents the most novel lithography.

DSENT returns dynamic power, leakage power and area statistics for each building block of a router. The primary contributing building blocks of a router are the flit buffers, crossbar switch, switch allocator and clock distributor. Simply adding the total area for each of the routers together would not illustrate topology-specific influences on the CPU die area. `on-chip-network-power-area-2.0.py` features a new extension to calculate the approximate CPU die area, which is presented in the following paragraphs.

Simulation core count	**Scale to model**	**Model**	**Model core count**	**Model die size (mm^2)**
1-4	No	14nm Skylake-D Server (quad-core)	4	122.6
5-10	Yes	14nm Skylake Server (LCC)	10	325.44
11-12	No	""	""	""
13-20	Yes	14nm Skylake Server (HCC)	18	485.0
21-28	Yes	14nm Skylake Server (XCC)	28	694.0
29-63	No	""	""	""
64-76	Yes	14nm Knights Landing (XCC)	76	682.6

Table 3.1: Five existing Intel x86 server CPUs used as comparative models in DSENT core area estimation. If the simulated CPU die size is not scaled to the model die size, the model die size is simply adopted.

For five existing 14 nm Intel x86 server CPUs, the die area in mm^2 and the number of cores are defined in the `getCoreAreaForCoreCount(num_cpus)` function, as shown in Table 3.1. Since DSENT does not include a 14 nm technology model and the comparative die size models are used merely for a rough estimation of CPU area, differences from the tri-gate technology model are not accounted for. For numbers of simulated cores, “num_cpus”, that are congruent with existing server CPUs, the die size is assumed to scale linearly with core count and is therefore calculated by (3.1). For other numbers of simulated cores, the die size is not scaled and the model die size is used. Thus, CPU area for simulations with a much larger core count than 76 is assumed to scale down significantly, corresponding to the continued shrinking of future process sizes.

$$\text{proportional\_die\_size} = \text{model\_die\_size} \times \frac{\text{num\_cpus}}{\text{model\_core\_count}} \tag{3.1}$$

The estimated area of a single CPU in m^2 is then calculated by (3.2):

$$\text{cpu\_area} = \frac{\text{proportional\_die\_size} \times 10^{-6}}{\text{num\_cpus}} \tag{3.2}$$

The uncore part of the die, including NoC(s), is assumed to take up 30% of the CPU area. Therefore, the area of a single core is set to (3.3):

$$\text{core\_area} = 0.7 \times \text{cpu\_area} \tag{3.3}$$

Cache and directory controllers are assumed to be located at a 45° angle from the router at a distance of $0.1\times$ the square root of the core area. This takes into account a maximum concentration factor of 4 CPUs per router. Thus, the external link (interconnecting router and CPU through an NI) wire lengths k are set to (3.4):

$$k = 0.1 \times \sqrt{\text{core\_area}} \tag{3.4}$$

The lateral space between a router and a CPU is therefore $\frac{k}{\sqrt{2}}$.

The area of the silicon die used is calculated by (3.5):

$$\begin{aligned}\text{Area} =&(\text{nrows} \times (\frac{k}{\sqrt{2}} + \sqrt{\text{router\_area}}) + \text{num\_vertical\_cpus} \times \sqrt{\text{core\_area}}) \\ &\times (\text{ncols} \times (\frac{k}{\sqrt{2}} + \sqrt{\text{router\_area}}) + \text{num\_horizontal\_cpus} \times \sqrt{\text{core\_area}})\end{aligned} \tag{3.5}$$

where "nrows" is the number of rows in the topology, "num_vertical_cpus" is the number of CPUs placed along the y-axis, "ncols" is the number of columns in the topology, "num_horizontal_cpus" is the number of CPUs placed along the x-axis and "ncpus" is the number of CPUs in the mesh. For concentrated meshes two CPUs per router are placed along the y-axis. For square mesh-based topologies (3.5) simplifies to:

$$\text{Mesh\_area} = (\text{nrows} \times (\frac{k}{\sqrt{2}} + \sqrt{\text{router\_area}}) + \sqrt{\text{ncpus}} \times \sqrt{\text{core\_area}}))^2 \tag{3.6}$$

For square mesh-based topologies internal link (interconnecting a pair of routers) wire lengths l are set using (2.5):

$$l = \frac{\sqrt{\text{Mesh\_area}}}{\sqrt{\text{ncpus}} - 1} \tag{2.5 reiterated}$$

For odd-dimensional topologies internal link wire lengths l' are set using (3.7):

$$l' = \frac{\text{Area\_xymax}}{\text{cpus\_xymax} - 1} \tag{3.7}$$

where "Area_xymax" = `max`(y-component of Area, x-component of Area) and "cpus_xymax" = `max`(num_vertical_cpus, num_horizontal_cpus). Out of all considered topologies, only the flattened butterfly has internal links that span non-neighboring routers. The wire lengths of these links are set to $n \times l$, where n is the proportional distance between routers. The repeaters and pipeline registers for these links suggested by the flattened butterfly paper are not implemented.

Since wire length calculation requires the router area, the link power results rely on the router area results. Wire delay is set according to 2011 International Technology Roadmap for Semiconductors (ITRS) projections [2], as interpreted by Al Khanjari and Vanderbauwhede [42]. For core counts up to 76, 14 nm CMOS is assumed, for which ITRS projected a global wire delay of 1 ns/mm, which amounts to one clock cycle per millimeter of wire for a 1 GHz CPU frequency. For core counts greater than 76, future many-core 10 nm architectures are assumed, for which ITRS projected a global wire delay of 33.8 ns/mm.

`ext/dsent/interface.cc`: Included with gem5/Garnet2.0. This file is fed Garnet2.0 simulation statistics by `util/on-chip-network-power-area*.py`. It was modified to accomodate the new `util/on-chip-network-power-area-2.0.py`.

3.2.6 Routing algorithms

Implemented in `src/mem/ruby/network/garnet2.0/RoutingUnit.cc`. gem5/Garnet2.0 comes with the following routing algorithms, which can be chosen by parameter `--routing-algorithm=`*i*:

0: Table based routing. Calculates the shortest paths between nodes. In case of multiple possible paths, the path with the minimum weight is chosen. In case of multiple equally weighted paths, one of the paths is chosen at random.

1: XY routing for mesh topology. Will always take the shortest path. Forces a flit to always take an x-direction path firstly and a y-direction secondly.

Added routing algorithm parameters are:

2: Random routing. Ignores shortest paths and instead chooses a random possible direction for the flit. Will deadlock quickly in most trials.

3: Adaptive routing. Not implemented thusfar.

3.2.7 Deadlock avoidance

Ring topology – escape VC

Attempted implementation in `src/mem/ruby/network/garnet2.0/OutputUnit.cc`. Required for a ring topology to not deadlock. Unfortunately, it is not functioning correctly, as many workloads will still deadlock. If parameter `--escapevc` is used, VC 0 will be used as an escape VC. Requires a number of VC's ≥ 2. Requires a shortest path routing algorithm. VC 0 will provide an acyclic escape path, while other VC's are allowed to be cyclic. Flits requesting to cross the eastmost vertical link on the ring are prohibited access to VC 0.

Hierarchical ring topology

Unfortunately hierarchical rings with a number of mesh rows (grid rows) greater than 4 suffer unresolved deadlock issues. For these topologies, the number of directory nodes is limited to the number of mesh rows and the number of cores is limited to 128.

CHAPTER 4

Experiments

4.1 Sniper multi-core simulator

The experiments detailed in the following sections are established used the unmodified latest version of the Sniper simulator[1], which at the time of writing is version 6.1, released on March 24th, 2015. The following paragraphs describe methodology characteristics shared by the Haswell and Knights Landing experiments.

Various benchmarks of the Splash2 [39] and PARSEC [43] suites were run for different interconnect topologies. The Splash2 FFT benchmark was found to best highlight disparities in resulting performance, power and area models. Therefore, experiments are limited to the FFT benchmark. All benchmarks are run for a problem size of $M = 22$, which specifies 2^M complex doubles as data points. Documentation of the FFT benchmark suggests a base problem size of 65,536 complex data points ($M = 16$) for up to 64 processing units. Since the FFT benchmark restricts core counts to powers of two, the Haswell simulations are run for 4, 8, 16, 32 and 64 cores and the Knights Landing simulations are run for 64 and 128 cores. This restricts topology dimensions as well.

For power and area modeling, the latest version of Sniper includes McPAT v1.0, which was released in August 2013. McPAT is updated to the latest version, which at the time of writing is version 1.3, released in February 2015. Between versions 1.0 and 1.3, no difference in the components of the power and area model was observed. Also, total area and leakage power results have negligible differences. The newest version of McPAT returns 4% lower total dynamic power numbers for the Haswell 64-core mesh, though. McPAT v1.3 is assumed to be more accurate than previous versions and is therefore used in power and area modeling of Sniper simulations. All current versions of McPAT are limited to 22 nm technology models. Consequently, power and area models for Knights Landing architecture simulations are askew.

4.1.1 Haswell interconnect scalability

Methodology

In order to evaluate interconnect scalability of the Haswell architecture, various Sniper configuration files were created according to the methodology explained in 3.1 on page 33. Primary parameters for all simulations are: 3 GHz, 1 thread per core, 1.2 V supply voltage, 22 nm technology, out-of-order ROB core model, 32 KB private L1d and L1i caches, 256 KB private L2 cache, 8 MB shared L3 NUCA cache, MESIF cache coherence protocol. The different topologies modeled for Haswell are: ring, bus, mesh and concentrated mesh. The segregated dual ring interconnect employed in actual mid-to-high-core-count Haswell CPUs is not modeled. In ring topologies, one DRAM controller is placed per 12 cores, corresponding to the actual Haswell architecture. Bus topologies are assigned a single DRAM controller, which is conventional. Mesh topologies are assigned two DRAM controllers, corresponding to Skylake and Knights Landing architectures.

[1] Website of the Sniper simulator: `http://snipersim.org` – accessed June 5, 2018.

Results

The charts below display the results and are composed as follows. Compute performance is measured by total CPU execution time. Normalized compute performance results represent the speed-up relative to the worst performing architecture. Interconnect (bus, ring NoC or mesh NoC) power consumption is the sum of interconnect dynamic power and bus leakage power. Leakage power is the sum of sub-threshold leakage and gate leakage. Normalized interconnect power consumption results represent the increase in interconnect power relative to the architecture that consumes the least power. Interconnect area is the sum of areas of all building blocks (e.g. routers, buffers) and interconnect wiring. Normalized interconnect area results represent the increase in interconnect area relative to the architecture that encompasses the smallest area.

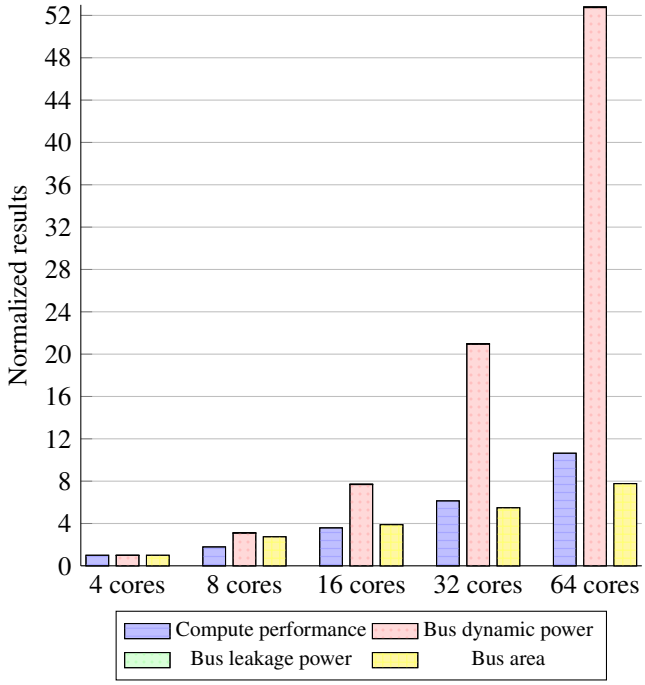


Figure 4.1: Haswell bus scalability for the FFT benchmark in the Sniper simulator with McPAT power and area modeling.

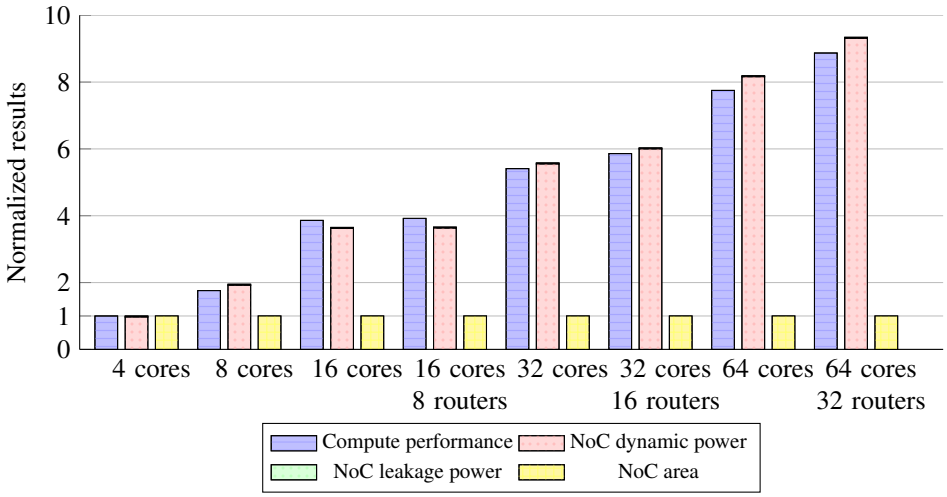


Figure 4.2: Haswell ring NoC scalability for the FFT benchmark in the Sniper simulator with McPAT power and area modeling.

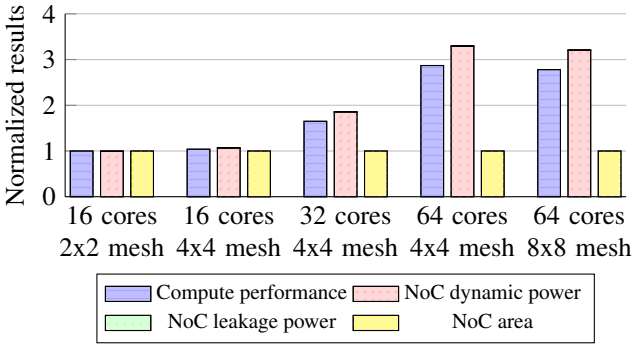


Figure 4.3: Haswell mesh NoC scalability for the FFT benchmark in the Sniper simulator with McPAT power and area modeling.

Evaluation

First, it must be noted that Sniper's implementation of McPAT (`tools/mcpat.py`) returns the same NoC total area in all experiments, whether version 1.0 or 1.3 or McPAT was used. The NoC area equals the router area, which amounts to 0.04 mm^2. Areas of the router's building blocks, VC buffer area for instance, are also the same for all NoC topologies. This cannot be correct, since a ring has less links than a mesh with an equal amount of routers and therefore less router input and output ports, which should result in a smaller router area. Either McPAT or – most likely – Sniper's implementation of McPAT must be lacking. NoC area is expected to increase with an asymptotic $\mathcal{O}(n)$, as referenced in Section 2.8 on page 24. Since NoC area results are presumed invalid, $\mathcal{O}(n)$ factoring is foregone, while the normalized results of "1" are included for completeness. The leakage power fraction for all simulated interconnects is also uncharacteristically small for 22 nm technology; almost invisible in the charts. Even with a bufferless NoC, which, as stated, this McPAT model does not employ, leakage power is expected to comprise a significantly larger portion of total power. Total power results do scale as expected, though.

Figure 4.1 shows the performance, power and area results for the simulated Haswell bus architecture, normalized to the 4-core configuration. Compute performance at 8 and 16 cores scales well; both achieving a normative speed-up of 90% of their theoretically expected maximum. At 32 and 64 cores speed-up starts to deteriorate, achieving normative speed-ups of 77% and 67%, respectively. Leakage power is negligible; not visible in the chart. Total power does scale as expected with close to $\mathcal{O}(n\sqrt{n})$, highlighting the dire scalability of the bus. Area results scale with $\mathcal{O}(n)$, which is far from the expected $\mathcal{O}(n^3\sqrt{n})$ for a non-segmented bus and $\mathcal{O}(n^2\sqrt{n})$ for a segmented bus. Again, this is indicative of likely errors in Sniper's implementation of McPAT.

Figure 4.2 shows the performance, power and area results for the simulated Haswell ring NoC architecture, normalized to the 4-core configuration. Performance scales worse than the bus architecture, marked by average normative speed-ups of 70% and 52% for the 32-core and 64-core topologies, respectively. For core counts of 32 and up, the concentrated ring topologies perform substantially better than their one-router-per-core counterparts. These topologies also consume more power, due to a higher amount of instructions per cycle.

Figure 4.3 shows the performance, power and area results for the simulated Haswell mesh NoC architecture, normalized to the 16-core 2×2 mesh. Compared to the 16-core 2×2 mesh, the 32-core and 64-core topologies achieve average normative speed-ups of 83% and 71%, respectively. Performance therefore scales slightly worse than the bus architecture, which achieves normative speed-ups relative to 16 cores of 86% for 32 cores and 74% for 64 cores. Similar to the ring topology, power scales with $\mathcal{O}(n)$ proportionally to the number of instructions per cycle.

Noteworthy is that, in the FFT benchmark, performance of the 64-core bus scales better than the 8×8 mesh NoC. The former also achieves an execution time of only 5% slower than the latter. This can be explained by the low amounts of coherency traffic induced by the FFT benchmark. In the 64-core bus, the recorded number of loads from RAM per cycle is only 0.09 and the number of loads from remote cache is only 0.003. A benchmark that stresses the interconnection network would be better suited to highlight differences in interconnect scalability. To this end, please refer to the Garnet2.0 synthetic traffic experiments in Section 4.2.3 on page 47.

4.1.2 Knights Landing many-core scalability

To evaluate the effects of scaling core count in the Knights Landing architecture from its actual 64 cores to 128, Sniper configuration files were created according to the methodology explained in 3.1 on page 33. Due to time constraints, results of these experiments are omitted.

4.2 gem5/Garnet2.0 NoC simulator

The Garnet2.0 experiments detailed in the following sections are established with the extensions introduced in Section 3.2 on page 34, which are built upon the gem5 source code of April 27, 2018, referenced in footnote 4 on page 34.

4.2.1 NoC latency-throughput relation

The Garnet2.0 synthetic traffic injector is used to examine the impact of different synthetic traffic patterns on the latency-throughput relation, detailed in Section 2.9.1 on page 26. Simulations are run for 20,000 cycles for the 64-core 8×8 Mesh topology. The zero-load latency for this topology is 7 cycles. Primary simulation parameters are: 64 directories, 512-bit link width, XY-routing algorithm, 4 VCs per VNET, 8 buffers per data VC, 1 buffer per control VC, random injection into all three VNETs. Garnet2.0's defaults of 1-flit sized control packets (VNETS 0 and 1) and 5-flit sized data packets (VNET 2) are maintained. Consequently, two thirds of injected packets are 1-flit and one third is 5-flit.

The Results are plotted in Figure 4.4. Evidently, "uniform random" traffic gives the most stable curve. "Bit-complement" is the most taxing traffic pattern, where sustained throughput decreases as latency increases due to heavy contention. "Neighbor" is the least taxing traffic pattern, since all packets traverse only one hop.

The bit-complement simulations, rerun with 1 VC per VNET, unsurprisingly lead to faster network saturation. The plot also illustrates that bit-complement has an asymptotic maximum sustainable throughput, which in this network is 0.1 packet/node/cycle. As explained in 2.9.1 on page 26, results coincide with most literature, such as [21] and [22].

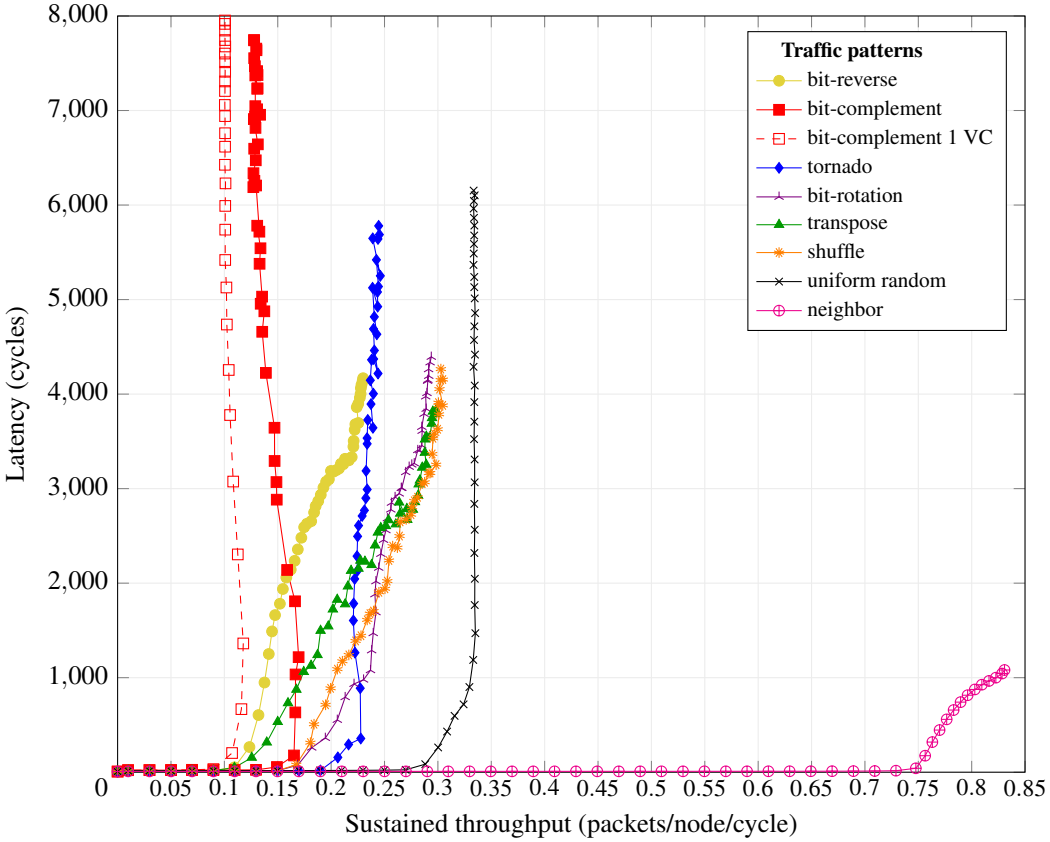


Figure 4.4: Latency-throughput relation for different synthetic traffic patterns in a 64-core 8×8 Mesh topology, for 20,000 cycles, with 4 VCs per VNET. Sustained throughput denotes the reception rate (the number of packets received per node per cycle).
For each traffic pattern, latency peaks at the maximum injection rate (offered throughput) of 1 packet/node/cycle. "Uniform random" gives the most stable curve. "Bit-complement" is the most taxing traffic pattern and "neighbor" is the least taxing one.
The bit-complement simulations, rerun with 1 VC, unsurprisingly lead to faster network saturation. The plot also illustrates that bit-complement has an asymptotic maximum sustainable throughput, which in this network is 0.1 packet/node/cycle.

4.2.2 Determining network saturation

These experiments intend to determine the point of network saturation, which is the minimum continuous traffic load (injection rate) for which the average packet latency increases toward infinity. Maximum sustainable throughput is therefore defined as the maximum continuous traffic load for which the network does not saturate. The uniform random simulations for the 64-core 8×8 Mesh topology from the experiments in the previous section are rerun for 200,000 cycles. Figure 4.5 on the next page shows a higher resolution version for the uniform random traffic plot of Figure 4.4 and a second curve for the same simulations run for 200,000 cycles.

After either 20,000 or 200,000 cycles, at an injection rate of 0.27 packets/node/cycle, uniform random achieves a throughput of 0.27 packets/node/cycle at a latency of 25 cycles. After either 20,000 or 200,000 cycles, at an injection rate of 0.28, uniform random achieves a throughput of 0.28. However, the latency jumps from 32 at 20,000 cycles to 95 at 200,000 cycles. At an injection rate of 0.29, both plots experience a slight reduction in throughput growth, achieving a throughput of 0.288, while the latency jumps from 85 at 20,000 cycles to 475 at 200,000 cycles. This illustrates the rapid deterioration of network performance for sustained traffic during network saturation. The point of saturation for uniform random traffic in this network is therefore at the injection rate where the two plots diverge: 0.27 packets/node/cycle, with an asymptotic maximum sustainable throughput of 0.27 packets/node/cycle. The 20,000 cycle plot shows some outliers around a throughput of 0.29, due to the short simulation time.

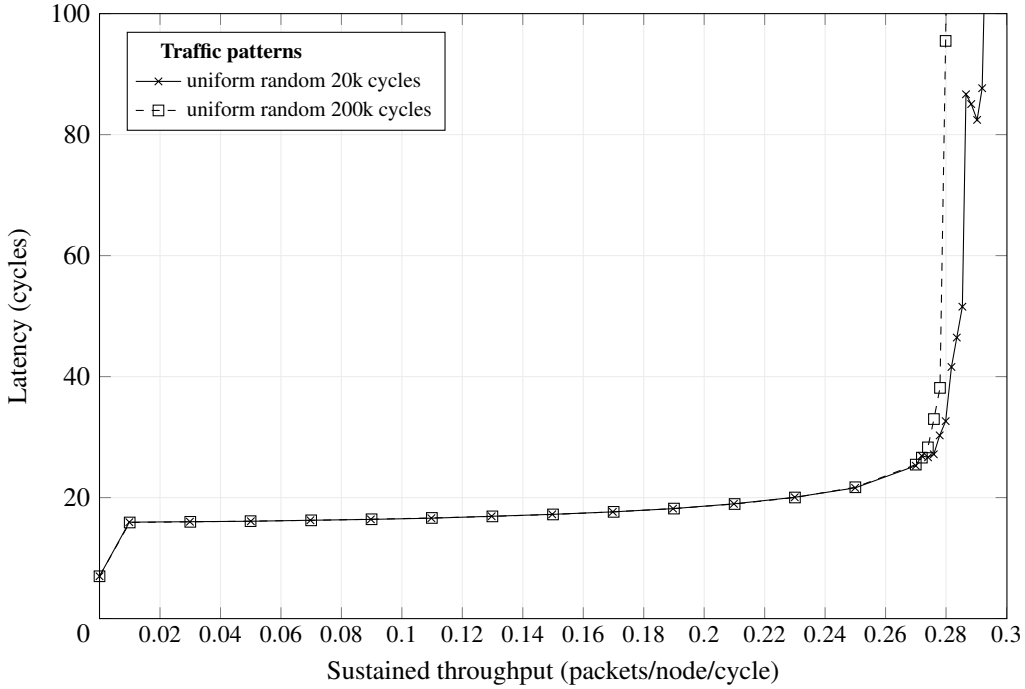


Figure 4.5: Plots demonstrating the point of network saturation for uniform random traffic in the 64-core 8×8 Mesh topology, detailed in Section 4.2.1 on page 45. When traffic is sustained during network saturation, the average packet latency increases toward infinity. Latency deterioration in saturated networks exacerbates with time. The point of saturation for uniform random traffic in this network is therefore at the injection rate where the two plots diverge: 0.27 packets/node/cycle, with an asymptotic maximum sustainable throughput of 0.27 packets/node/cycle.

4.2.3 Performance, power and area at network saturation

Methodology

To compare theoretical performance, power and area for different topologies, synthetic on-chip network traffic simulations are run for different configurations of the implemented Topologies in gem5/Garnet2.0. Although seldomly matching traffic generated by real-world applications, synthetic traffic patterns are well suited for evaluating capability limits of NoCs. Figure 4.4 on the preceding page shows that uniform random traffic exhibits the most stable latency-throughput curve in a mesh topology for short simulation times. Since all implemented topologies are grid-based as well, the uniform random traffic pattern is chosen to compare the scalability of different topologies. For uniform random traffic, the Garnet Synthetic Traffic injector selects both the source and destination router for a packet randomly, deterministically.

To provide a fair comparison between different topologies and network parameters, each topology is evaluated at the injection rate effectuating the maximum sustainable throughput. Section 4.2.2 on the previous page explains that the maximum sustainable throughput is defined as the maximum continuous traffic load for which the network does not saturate. All simulations are run for a low amount of 20,000 cycles, due to long execution times for simulations and gem5's memory requirements growing with both simulation time and the number of nodes. Figure 4.5 shows 20,000 cycles to achieve adequate accuracy for a 64-router mesh. For non-saturating injection rates, the 1024-router flattened butterfly topology showed only a 1% disparity in results between 10,000- and 20,000-cycle simulations. The maximum sustainable throughput, at which each topology is evaluated, is therefore selected with an error margin of 0.02.

Shared simulation parameters are: 512-bit link width, shortest path routing algorithm, 4 VCs per VNET, 8 buffers per data VC, 1 buffer per control VC, random injection into all three VNETs. Garnet2.0's defaults of 1-flit sized control packets (VNETS 0 and 1) and 5-flit sized data packets (VNET 2) are maintained. Each CPU is connected through one L1 cache controller node. The number of directory nodes equals the number of routers and is limited to powers of two, with a maximum of 256, due to

memory address limitations of gem5's ruby memory system model. For 512- and 1024-router topologies, the 256 directory nodes are distributed evenly across the routers. No other nodes are employed, as per default in Garnet2.0. Section 2.11.1 on page 29 describes the Garnet2.0 infrastructure in detail.

The default deadlock threshold of 50,000 is upheld. This deadlock threshold cannot be reached with 20,000-cycle simulations, so deadlock detection is effectively disabled. Flattened butterfly and mesh topologies employ XY-routing for deadlock avoidance. Fully connected topologies inherently avoid deadlock. Due to their inherent cyclical nature, ring and hierarchical ring topologies are prone to deadlock. As explained in 3.2.4 on page 35 no valid deadlock avoidance scheme is implemented. The proposed escape VC deadlock avoidance scheme for rings is disabled. The ring and hierarchical ring topologies exhibit erratic behavior when deadlock occurs. Although underperforming due to inevitable deadlock, these topologies are still included in experiments, for reference.

Results for 16 and 32 cores

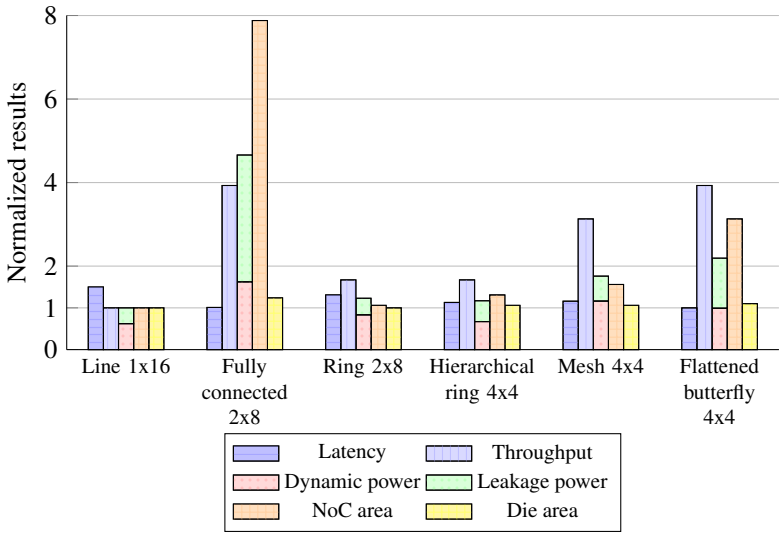


Figure 4.6: Performance, power and area for various 16-core topologies, under near-saturation uniform random traffic. Dynamic power and leakage power are normalized to 1 for the line topology and for other topologies they are normalized to the total power of the line topology.

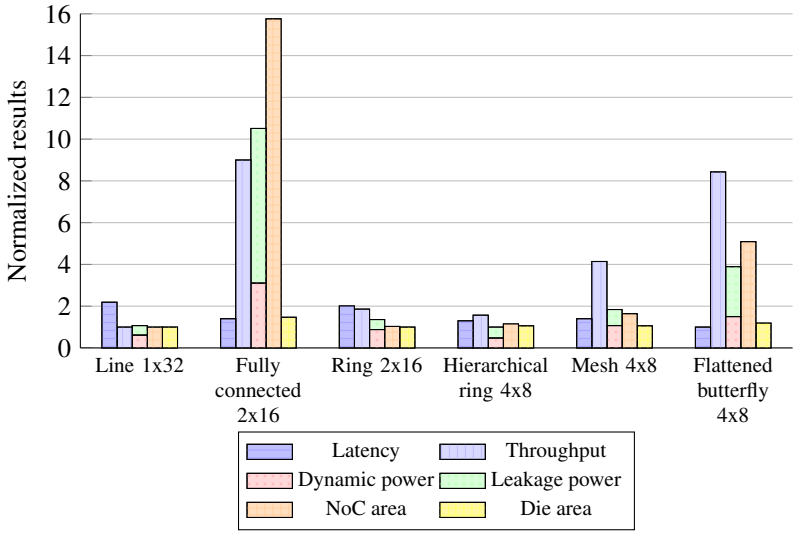


Figure 4.7: Performance, power and area for various 32-core topologies, under near-saturation uniform random traffic. Dynamic power and leakage power are normalized to 1 for the hierarchical ring topology and for other topologies they are normalized to the total power of the hierarchical ring topology.

Evaluation for 16 and 32 cores

Figure 4.6 shows the normalized performance, power and area results for various 16-core topologies. Figure 4.7 shows the results for the same topologies, scaled to 32 cores. The 16-core and 32-core flattened butterfly topologies do not employ the canonical concentration factor. The results for 64 and 128 cores, below, do include the concentration factor of 4.

For 16 cores, the fully connected topology shows performance equal to the flattened butterfly, both in terms of latency and throughput. Both these topologies outperform the other topologies, due to having more links, allowing more throughput and mitigating network congestion. The fully connected topology exhibits a substantial increase in power and area compared to the flattened butterfly, though. Leakage power is the dominant factor in total power in this topology, at 11 nm technology, due to its large amount of links and buffers. For the same reason, the flattened butterfly has about equal proportions of dynamic and leakage power. The line topology, mimicking a bus structure within a NoC, unsurprisingly performs worst on all accounts. The ring and hierarchical ring perform about the same, due to their inadequate implementation, as explained in Section 3.2.4 on page 35. The mesh topology draws less power and encompasses less area than the flattened butterfly, at the cost of a higher latency and lower throughput.

Results for 64 and 128 cores

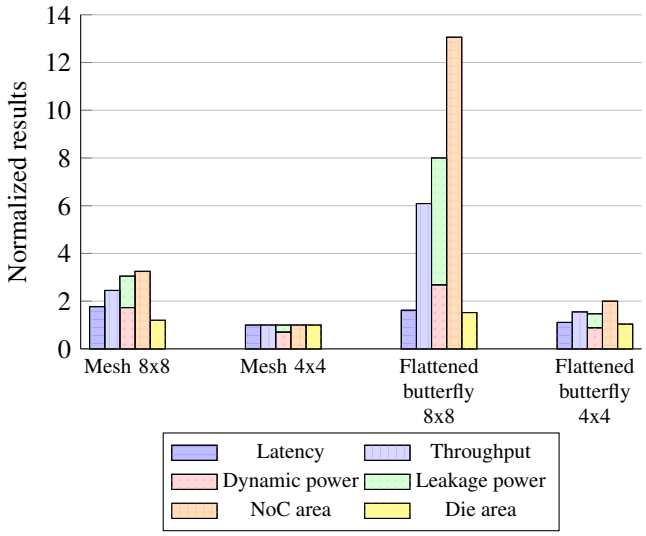


Figure 4.8: Performance, power and area for mesh and flattenedbutterfly 64-core topologies, under near-saturation uniform random traffic. Concentration factors of 1 and 4 CPUs per router are applied. Dynamic power and leakage power are normalized to 1 for the 4×4 mesh and for other topologies they are normalized to the total power of the 4×4 mesh.

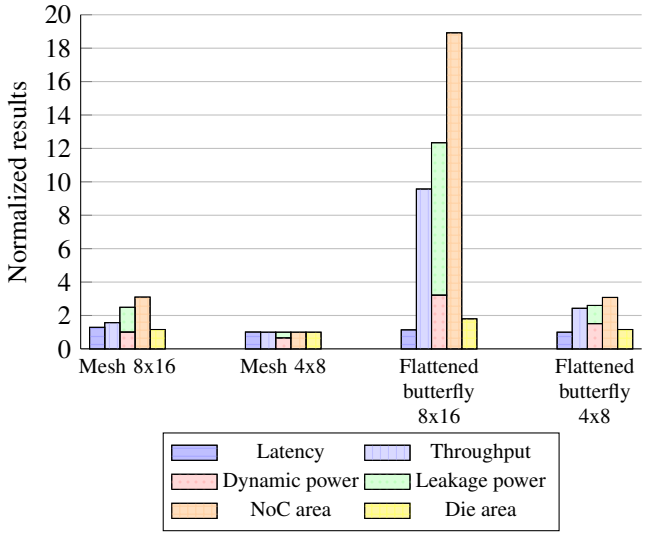


Figure 4.9: Performance, power and area for mesh and flattened butterfly 128-core topologies, under near-saturation uniform random traffic. Concentration factors of 1 and 4 CPUs per router are applied. Dynamic power and leakage power are normalized to 1 for the 4×8 mesh and for other topologies they are normalized to the total power of the 4×8 mesh.

Evaluation for 64 and 128 cores

Figure 4.8 shows the normalized performance, power and area results for mesh and flattened butterfly 64-core topologies. Concentration factors of 1 and 4 CPUs per router are applied. Figure 4.9 shows the results for the same topologies, scaled to 128 cores. The results for 64 and 128 cores, below, do include the concentration factor of 4.

For each mesh topologies in both figures, the scaled results are nearly identical, typifying the theoretical asymptotic cost of $\mathcal{O}(n)$ for mesh power and area. On the other hand, the flattened butterfly appears

to scales worse, on all accounts, except for latency. However, simulations are run at a near-saturating injection rate, which is much higher for are the flattened butterfly, thereby achieving throughputs $6\times$ and $2.4\times$ than the comparative mesh topologies.

4.2.4 gem5 SE-mode x86 simulations

After tedious efforts, the FFT benchmark of the Splash2 suite [39] was able to run in gem5 x86 full-system (FS) mode. In addition to the simulator's large compute and memory overhead for many-core simulations, even a quad-core simulation resulted in extensive simulation times. This made FS-mode not a viable option for x86 simulations. In system-call emulation (SE) mode, the FFT benchmark achieved acceptable simulation times. SE-mode emulates applications on the host OS by trapping system calls made to the host and emulating them on the simulated system. Unlike FS-mode, SE-mode does not take overheads of the OS and background applications into account, which leads to reduced accuracy for SE-mode. The FFT benchmark in Sniper simulations detailed in Section 4.1.1 on page 42 proved to not stress the interconnection network sufficiently to compare scalability of different interconnects. Due to time constraints, further exploration of gem5 x86 experiments was abandoned.

CHAPTER 5

Conclusions

5.1 Effects of the evolving performance, power and area relation

Up to present-day (2018), CPU technology advancement has been adhering to Moore's law. Since 2006, CPUs have been gradually approaching power and thermal limits, leading to continued changes in the performance, power and area relation. In an effort to sustain performance scaling proportional to Moore's law, technology trends have been to increase the number of cores. Due to limits of transistor scaling, power consumption, as well as the proportion of power consumption due to leakage, is increasing exponentially with technology evolution on the nanoscale. This results in a continuous growth in CPU core count and dark silicon. Feature sets of heterogeneous core operation and memory hierarchies are becoming ever more complex. This begs the need for an efficient and scalable interconnection network.

Udipi et al. have proposed an alternative to widespread NoC interconnects: the segmented filtered hierarchical bus. Compared to the flattened butterfly network, their 64-core model achieves average energy reductions of 13× and 2.5× in the address and data network, respectively. This comes at the cost of a resulting 46% increase in execution time compared to the flattened butterfly network. It is unknown how well this model performs in practise or how well it scales beyond 64 cores. The imminent many-core era is well served by developing energy reduction techniques for the proven-scalable NoC interconnect.

5.2 Experiment evaluation summary

Computer architecture simulators are useful tools for exploration of architecture scalability. This thesis' experiments with the Sniper computer architecture simulator and its associated McPAT power and area modeling tool proved these tools to be inadequate for evaluation of interconnects, as well as future chip technology, due to McPAT's limitation to 22 nm technology. Conclusions are drawn that Sniper's implementation of McPAT is both outdated (2015) and deficient in modeling interconnect power and area. Additionally, the Splash2 FFT benchmark, which were observed to highlight the greatest discrepancies in performance, power and area models out of various Splash2 and PARSEC benchmarks, induced far too low amounts of coherency traffic for evaluating interconnect scalability.

A multitude of extensions to the gem5/Garnet2.0 simulator were presented. Garnet2.0 includes the mesh topology, which was employed in synthetic traffic simulations along with the newly created line, fully connected, ring, hierarchical ring and flattened butterfly topologies. Garnet2.0 was extended for use with the DSENT power and area modeling tool, with an additional newly proposed model for die area.

Synthetic traffic simulations at near-saturation injection rate show that for 16 cores, the fully connected topology shows performance equal to the flattened butterfly, both in terms of latency and throughput, due to having more links. Leakage power is the dominant factor in total power in this topology, at 11 nm technology, due to its large amount of links and buffers. The mesh topology draws less power and encompasses less area than the flattened butterfly, at the cost of a higher latency and lower throughput. Simulated mesh topologies scale in accordance with the theoretical asymptotic cost of $\mathcal{O}(n)$ for power and area. The flattened butterfly achieves substantially higher near-saturating injection rates than the mesh, thereby achieving throughputs 6× and 2.4× at 128 cores versus comparative mesh topologies,

with concentration factors of 1 and 4, respectively.

5.3 Future work

Garnet2.0 provides a robust and well-documented framework for detailed NoC simulation. Released in 2016, it is still lacking many features, such as adaptive routing techniques and optimization techniques, such Dynamic Voltage and Frequency Scaling (DVFS). Garnet2.0's code is well factored for out-of-the-box operation, which has its disadvantages. For example, the `RoutingUnit` class, in which routing algorithms are implemented, is not VC-aware. It contains a pointer to its parent `Router`, which in turn references its `InputUnit` and `OutputUnit` classes holding the VC states for this router's ports, as well as a pointer to the `GarnetNetwork` class, which references all routers in the network. By default, Garnet2.0 only supports shortest path routing, which is leveraged from its parent ruby `Topology` and `Routing` infrastructures. Thus, implementing sophisticated adaptive routing techniques require a lot of code extensions and possibly code refactoring, which can interfere with other network models depending on the leveraged parent modules. Routers and links have a cycle-accurate latency, but each link is assumed to be a straight edge of equal length, apparently tailored to grid-based topologies.

Adaptive routing techniques such as deflection routing should be considered for hierarchical ring topologies. Deflection routing eliminates deadlock, while decreasing design complexity at equal or better performance and better energy efficiency. To evaluate interconnect scalability for future technologies at the nanoscale with DSENT, models for technologies beyond 11 nm should be added.

References

[1] Shekhar Borkar and Andrew A Chien. “The future of microprocessors”. In: *Communications of the ACM* 54.5 (2011), pp. 67–77.

[2] Bernd Hoefflinger. “ITRS: The international technology roadmap for semiconductors”. In: *Chips 2020*. Springer, 2011, pp. 161–174.

[3] Robert H Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268.

[4] Hadi Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE. 2011, pp. 365–376.

[5] Linda Wilson. “International technology roadmap for semiconductors (ITRS)”. In: *Semiconductor Industry Association* (2013).

[6] Shekhar Borkar. “The exascale challenge”. In: *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*. IEEE. 2010, pp. 2–3.

[7] Anil Kanduri et al. “A perspective on dark silicon”. In: *The Dark Side of Silicon*. Springer, 2017, pp. 3–20.

[8] Qing Xie et al. “Performance comparisons between 7-nm FinFET and conventional bulk CMOS standard cell libraries”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 62.8 (2015), pp. 761–765.

[9] Shoaib Akram et al. “A workload-adaptive and reconfigurable bus architecture for multicore processors”. In: *International Journal of Reconfigurable Computing* 2010 (2010), p. 2.

[10] Aniruddha N Udipi, Naveen Muralimanohar, and Rajeev Balasubramonian. “Towards scalable, energy-efficient, bus-based on-chip networks”. In: *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE. 2010, pp. 1–12.

[11] Rasmus Ulfsnes. “Design of a Snoop Filter for Snoop Based Cache Coherency Protocols”. MA thesis. Institutt for elektronikk og telekommunikasjon, 2013.

[12] Aanjhan Ranganathan et al. “Counting stream registers: An efficient and effective snoop filter architecture”. In: *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE. 2012, pp. 120–127.

[13] Bin-feng Qian and Li-min Yan. “The research of the inclusive cache used in multi-core processor”. In: *Electronic Packaging Technology & High Density Packaging, 2008. ICEPT-HDP 2008. International Conference on*. IEEE. 2008, pp. 1–4.

[14] James Goodman and HHJ Hum. “MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects (2004)”. In: (2004).

[15] Xing Liu et al. “Efficient sparse matrix-vector multiplication on x86-based many-core processors”. In: *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM. 2013, pp. 273–282.

[16] Avinash Sodani et al. “Knights landing: Second-generation intel xeon phi product”. In: *Ieee micro* 36.2 (2016), pp. 34–46.

[17] Avinash Sodani and Senior Principal Engineer. “Knights landing intel xeon phi cpu: Path to parallelism with general purpose programming”. In: *Keynote Address HPCA* (2016).

[18] Evgeny Bolotin et al. “Cost considerations in network on chip”. In: *INTEGRATION, the VLSI journal* 38.1 (2004), pp. 19–42.

[19] Naveen Muralimanohar and Rajeev Balasubramonian. “Interconnect design considerations for large NUCA caches”. In: *ACM SIGARCH Computer Architecture News*. Vol. 35. 2. ACM. 2007, pp. 369–380.

[20] Lionel M Ni. “Issues in Designing Truly Scalable Interconnection Networks.” In: *ICPP Workshop*. 1996, pp. 74–83.

[21] Cheng Li et al. “LumiNOC: A power-efficient, high-performance, photonic network-on-chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.6 (2014), pp. 826–838.

[22] Rohit Sunkam Ramanujam et al. “Design of a high-throughput distributed shared-buffer NoC router”. In: *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*. IEEE. 2010, pp. 69–78.

[23] Andrew B Kahng et al. “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, 2009. DATE’09.* IEEE. 2009, pp. 423–428.

[24] Amlan Ganguly, Partha Pratim Pande, and Benjamin Belzer. “Crosstalk-aware channel coding schemes for energy efficient and reliable NOC interconnects”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.11 (2009), pp. 1626–1639.

[25] Javad Zarrin, Rui L Aguiar, and João Paulo Barraca. “Manycore simulation for peta-scale system design: Motivation, tools, challenges and prospects”. In: *Simulation Modelling Practice and Theory* 72 (2017), pp. 168–201.

[26] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. “Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Nov. 2011, 52:1–52:12.

[27] Jason E Miller et al. “Graphite: A distributed parallel simulator for multicores”. In: *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE. 2010, pp. 1–12.

[28] Sheng Li et al. “The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 10.1 (2013), p. 5.

[29] Nathan L Binkert et al. “The M5 simulator: Modeling networked systems”. In: *IEEE Micro* 26.4 (2006), pp. 52–60.

[30] Milo MK Martin et al. “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset”. In: *ACM SIGARCH Computer Architecture News* 33.4 (2005), pp. 92–99.

[31] Chen Sun et al. “DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling”. In: *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE. 2012, pp. 201–210.

[32] Anastasiia Butko et al. “Accuracy evaluation of gem5 simulator system”. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*. IEEE. 2012, pp. 1–7.

[33] Pablo Abad et al. “Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers”. In: *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE. 2012, pp. 99–106.

[34] Nathan Binkert et al. “The gem5 simulator”. In: *ACM SIGARCH Computer Architecture News* 39.2 (2011), pp. 1–7.

[35] Yaosheng Fu and David Wentzlaff. “PriME: A parallel and distributed simulator for thousand-core chips”. In: *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE. 2014, pp. 116–125.

[36] Daniel Sanchez and Christos Kozyrakis. “ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems”. In: *SIGARCH Comput. Archit. News* 41.3 (June 2013), pp. 475–486. ISSN: 0163-5964. DOI: 10.1145/2508148.2485963. URL: http://doi.acm.org/10.1145/2508148.2485963.

[37] Niket Agarwal et al. “GARNET: A detailed on-chip network model inside a full-system simulator”. In: *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE. 2009, pp. 33–42.

[38] Changkyu Kim, Doug Burger, and Stephen W Keckler. “An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches”. In: *Acm Sigplan Notices*. Vol. 37. 10. ACM. 2002, pp. 211–222.

[39] Steven Cameron Woo et al. “The SPLASH-2 programs: Characterization and methodological considerations”. In: *ACM SIGARCH computer architecture news*. Vol. 23. 2. ACM. 1995, pp. 24–36.

[40] NJ Dimopoulos and R Sivakumar. “Deadlock-preventing routing in Hypercycles”. In: *Canadian Journal of Electrical and Computer Engineering* 19.4 (1994), pp. 193–199.

[41] John Kim, James Balfour, and William Dally. “Flattened butterfly topology for on-chip networks”. In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2007, pp. 172–182.

[42] Sharifa Al Khanjari and Wim Vanderbauwhede. “The performance of NoCs for very large many-core systems under locality-based traffic”. In: *International Journal of Computing and Digital Systems* 5.2 (2016), pp. 115–124.

[43] Christian Bienia and Kai Li. “Parsec 2.0: A new benchmark suite for chip-multiprocessors”. In: *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*. Vol. 2011. 2009.