Exploring Network on Chip Architectures Using GEM5

Arthita Ghosh, Ankita Sinha and Nancy
Former Students, CSE Department
Heritage Institute of Technology
Kolkata, India
e-mail: ghosharthita95@gmail.com,
ankita.sinha4894@gmail.com,
nainsyjaiswal403@gmail.com

Arindam Chatterjee
Assistant Professor, CSE Department
Heritage Institute of Technology,
Kolkata, India
Email- Arindam.Chatterjee@heritageit.edu

***Abstract*—The number of computing resources in a SoC (System on a Chip) has been steadily increasing to meet greater computational requirements in a shorter time. However, associated problems that have cropped up with these increased number of resources also need to be addressed. In this work, we have made a comparative study of simple and Garnet networks on these SoCs using both mesh and crossbar topologies in each. We have seen that mesh is better than crossbar for both simple and garnet networks, but in mesh performance degrades as the size of network increases. Furthermore, a detailed study of standard cache coherence protocols has also been made with the help of MI and MESI_Two_Level and MOESI_CMP protocols. The simulator we have used for this study is GEM5 and the benchmark used is SPLASH2. We have observed that performance measured in form of computation time has increased from MI to MESI_Two_Level to MOESI_CMP.**

Keywords—NoC, Simple network, Garnet Network, Cache coherence, Snoopy protocol, Directory based protocol, GEM5.

I. INTRODUCTION

The number of computing resources on single-chip has enormously increased to meet the growing need of high-performance systems. With the addition of computing resources like CPUs, specific IPs (Intellect Properties) etc. to build a system in System-on-Chip, determination of the kind of interconnections required between these resources has become a challenging issue.

The basic idea of NoCs or Network-On-Chips came from traditional large-scale multiprocessors and distributed computing networks. NoCs are basically communication subsystems on an integrated circuit (commonly called a "chip"), typically between IP cores in a SoC.

Scaling NoCs to accommodate tens or hundreds of core is not efficient and may introduce bottleneck. Hence, a detailed study of different kinds of interconnection networks implemented on these NoCs is required to determine an efficient network which will reduce bottleneck and increase efficiency with increase in the number of computing resources.

In this work, we have presented a simulation based comparative study of performance parameters of certain NoC topologies. The simulation software have used is GEM5.

The GEM5 simulator is a modular platform for computer-system architecture research. GEM5 is open-source microarchitectural simulator having the capability of simulating NoCs. Some of the key features of GEM5 which has been of special use in this work are-

Multiple interchangeable CPU models.-
GEM5 provides four interpretation-based CPU models
Event-driven memory system
Homogeneous and heterogeneous multi-core
Full-system capability

In this work, a simulation based comparison and a detailed analysis of simple and Garnet networks with Mesh-XY and Crossbar topologies has been done. Along with this, a comparison between Snoopy and Directory based protocols to address the cache coherence problem is performed. The later is done using the SPLASH2 benchmark programs like FFT, OCEAN, RADIX etc.

Section II below describes some related works in this field. Section III describes in detail our methodology using GEM5 to carry out various sets of experiments. Section IV represents in tabular form various results we have obtained after performing sets of experiments as is described in Section III. Section V analyzes the results, makes a few conclusions from the analysis and gives an outline for future scope of work.

II. BACKGROUND

Different architectures and topologies have been proposed for proper optimization of the resources on NoCs. For example, in [1] authors describe a hierarchical crossbar switch that logically and hierarchically separates the control logic to increase performance and reduce area consumption. In [2], authors proposed a decomposed crossbar to reduce contention and thereby achieve energy-efficient architectures. The RoCo work in [3] introduced the idea of decoupling the NoC router operation into two functionally independent modules, each with its own compact 2×2 crossbar. The work in [4] uses the NoC router to aid cache coherence in CMPs. By keeping track of cache accesses within each router, the on-chip network now becomes an integral part of the cache coherence protocol. In [5], authors use comparisons on various directory based protocols with varying injection rates to show how the network latency and

978-1-5386-2924-6/17 $31.00 © 2017 IEEE
DOI 10.1109/ICIT.2017.16

power consumption values vary across coherence protocols. Here they show that the MOESI_CMP protocol is most demanding. In [6], the author simulates and analyzes several coherence protocols and their associated memory hierarchies to highlight the performance impact of finer-designed protocols and their reaction faced to qualitative and quantitative changes into the hierarchy.

In [7], authors develop a hybrid cache coherence protocol for Chip Multiprocessors (CMPs), referred to as MESCIF (Modified Exclusive Shared Clean Invalid Forward). This combines the advantages of both directory-based and broadcasting protocols. They have introduced a small directory based cache (DB-CACHE) and cache-coherence bus (CC-BUS) into existing CMP architecture to realizes MESCIF which overcomes the shortcomings of existing coherence protocols. In [8], authors describe a virtual computing group (VCG) model to improve the utilization of the computing resources on NoC-based many-core processor. Each VCG can be reconfigured into different size and topology before the program starts. The token protocol for cache coherence is adopted to improve the performance of memory accessing. Modifications to Token protocol are made to support cache coherence in the local VCG only, which lightens the communication penalty on a large NoC. Authors have shown via simulation in GEM5 that both Network Latency and Network Interface Queue lengths decreases with the adoption of the new protocol.

The topic cache coherence is a very critical issue in the multi-core system and assumes a pivotal place in our work. Cache coherence is defined as the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system. Cache coherence protocols applied to deal with and resolve the problem of coherence inside shared-memory multiprocessors.

Snoopy and Directory protocols are the standard protocols which have been used to maintain cache coherence in multi core systems. We can see the way protocols have evolved previously to achieve high performance irrespective of design constraint in [2] and [3]. It has been seen that Snoopy is not good for large scalable multi-core system and Directory method has directory storage overhead and high cache-to cache transfer latency. Therefore, protocols have evolved to address these issues from MI to MSI to MESI to MOESI. However, they all have their drawbacks and advantages as well.[1].

The MI protocol is easy to implement and the cache controller is simple to design. It does not however distinguish between a shared and a modified block. The last problem is eliminated in the MSI protocol, but it results in unnecessary broadcast of invalidation messages. The MESI protocol takes care of this unnecessary broadcast problem but results in extra hardware logic. In the MOESI protocol, a distinct state Owned is proposed, which can be obtained by only one cache (with modified abilities) with other processors may be in shared state at the same time; this improves the overall performance. But this protocol suffers from the disadvantage that Snoopy implementation of MOESI, when a read request arrives on bus, every sharer of the block sends requested data; this can create message contention on the bus [9].

[1] (**M : Modified; I: Invalidated; E: Exclusive; O:Owned; S: Shared** states)

III. OUR WORK

In our work we have provided a comparative analysis of simple and Garnet networks using Crossbar and Mesh topologies.

We have made a detailed study of existing cache coherence methods, such as Snoopy and Directory coherence technique. Snoopy coherence protocol is studied with the help of MOESI coherence protocol. Directory coherence protocol is studied with the help of MESI and MOESI coherence protocols. The GEM5 [10] simulator and Splash-2 benchmark programs like FFT, Ocean, Radix, LU and FMM have been used to compare their performance.

Syscall Emulation mode and Full System mode

GEM5 provides us capability to run programs in two modes- system emulation and full system. For this work, we have only used system emulation mode. It is easy to run programs in this mode.

In system emulation mode, we only need to specify the binaries to be simulated. These binary file can be statically or dynamically linked. Program used for configuring and running simulations in syscall emulation mode is *configs/example/se.py*. The binary file is specified with option –c. Full system mode simulates a complete system providing OS based simulation environment (this mode is not used in this work).

Simple Network Invocation

In this work we are using the Ruby memory model which supports two network models- **simple** and **garnet 2.0** [11].

Simple network model is the default network model in Ruby. It models hop by hop network traversal. The flow control depends on available buffer and available bandwidth in output link.

Garnet 2.0 is another network model in Ruby. It is a more detailed interconnection network and is in active development stage. New features are periodically pushed into it. It's older version is Garnet network and this model is built on top of it. It provides cycle-accurate micro-architectural implementation.

In this work, we will focus only on simple network model. This model is simpler than garnet 2.0 model. We invoked a simple network and analyzed certain performance parameters of the network. For doing this, different topologies like **Crossbar, Mesh_XY, CrossbarGarnet** etc. are available. The default topology is Crossbar and number of CPUs in it is 16. We can change these parameters either in program or through command line option. The Ruby Network Tester provides a framework for simulating the interconnection network with controlled inputs. This is useful for network testing/debugging. The tester works in

conjunction with the network_test coherence protocol. We tested the network for Crossbar and Mesh topology, and analyzed different performance parameters like number of read/write requests, average bandwidth, cumulative time taken etc by changing the configuration of the network.

The two topologies we are going to use are- Crossbar and Mesh. In Crossbar, each controller is connected to a simple switch and each switch is connected to a central switch. But in Mesh_XY, each router has one cache controller and one directory controller and hence number of CPUs is always equal to number of directories. All X-direction links are used first then Y-direction links are used.

Cache Coherence

In any memory model, we have several processors with on-chip caches sharing common memory. There should be uniformity in shared resource data stored in multiple local caches. But the problem may arise due to incoherent data. Each processor has a separate cache memory, so each shared data may be stored in multiple caches. When multiple processors with separate caches share a common memory, it is necessary to keep the caches in a state of coherence by ensuring that any shared operand that is changed in any cache is changed throughout the entire system. The coherence of data can be maintained in either of two ways- through Directory based or Snooping based protocols. Both of these protocols have their own benefits and drawbacks. We will discuss about each of them one by one.

Snoopy Based Protocol

Snooping is a method in which each cache will monitor address line for the access to memory location they have cached. If change is made in data of any one cache, then all other copies are invalidated or updated. There are mainly two types of snooping protocol:-write invalidate and write update. Snooping based protocol is fast if large bandwidth is available. It can be considered more like a shared bus based system. For small number of nodes, it can do well. But as the count of nodes keep growing, problems may arise. Hence Snoopy protocol is not scalable. The default protocol in GEM5 is Snoopy. The earlier program of simple network invocation follows Snoopy based protocol. The second program that we tried is '*memtest.py'*. Both of these programs use MOESI protocol.

Program using Snoopy protocol('memtest.py')

```
gem5.debug  configs/example/memtest.py  -c 4:1  -t 1:0:2  -l
500000
```

In the above command, -c denotes the **associativity of cache**, -t gives the tester configuration and –l specifies the number of loads after which program is going to stop. We ran this command for different cache associativity by **varying L1 cache size and cache block size**. L1 cache size and block size is changed by changing its value in the program. Default L1 size is 32kb and block size is 64byte. We tried to make both of them 64 and 128 with cache associativity 2:2:1(default), 4:3:1 and 4:1. When cache associativity is 4:1, we need to change the associativity of tester as 1:0:2 because in this case we need only tester present at three levels, not four. For other two cases there is no need to mention about tester as its default value (1:1:0:2) will be taken.

Directory Based Protocol

Another mechanism to handle cache coherence is directory based system. It uses a special directory to serve instead of

shared bus. Coherence is maintained by placing data in a common directory between caches. An entry from primary memory is loaded to the cache of a processor by taking permission from directory. When an entry is changed, the directory will then either update or invalidate the other caches with that entry. As compared to snooping protocol, directory based system is scalable and requires much less bandwidth.

The directory based protocol is always implemented for ruby memory model only. MESI_Two_level and MESI_Three_ level are directory based protocols. Whenever we want to change the protocol, we need to configure it by command:

Scons RUBY=TRUE PROTOCOL=MESI_TWO_Level build/ARM/gem5.opt –j8

After changing the protocol, we again ran simple network with different configurations. Command is same as given above.

Simple Network (for Mesh_XY topology)

```
gem5.debug  configs/example/ruby_random_test.py \
 --stats-file=m16256.txt          --num-cpus=16  \
          --num-dirs=16           --l1d_size="256B" \
          --network=simple        --topology=Mesh_XY \
           --mesh_rows=4
```

This program **ruby_random_test.py** runs for both MESI two level as well as three level cache. The output is discussed in later section of this work.

Ruby memory model

There are two types of memory model in gem5-classic and ruby. In this paper, we have used ruby memory model in all the programs. A detailed simulation model for memory subsystem is implemented by ruby. It models inclusive/exclusive cache hierarchies with various replacement policies, coherence protocol implementations, interconnection networks, DMA and memory controllers, various sequencers that initiate memory requests and handle responses.

SPLASH2 Benchmarks

There are two benchmark programs widely used in GEM5, viz., SPLASH2 and PARSEC. In this work, we will focus on SPLASH2 benchmark programs. It has four computational kernels and eight complete application programs representing variety of computations in various domains. A brief description about some of these applications and kernels is given here.

FFT (Fast Fourier Transform) is used to minimize inter-processor communication. LU factors large matrix into lower

and upper triangular matrix. Radix is an iterative algorithm and performs one iteration for each digit of keys. In each iteration, processor passes its assigned key and generates a local histogram. Ocean is used to study large-scale ocean movement based on eddy and boundary current. FMM (Fast Multiple Method) simulate interaction in two-dimension using a different hierarchical method.

Compilation of SPLASH2 in ARM:

We have generated the cross-complied file of all apps and kernels. To run programs on multiple cores, we need m5 thread library. We statically compiled our program with the help of M5thread suite and then ran compiled program code on GEM5 simulator.

Running SPLASH2 in SE mode (ARM architecture)

1. Next step is to copy all the cross-complied files in home directory.
2. Now we are ready to run SPLASH2 programs. We ran the program using command given below: *./build/ARM/gem5.opt configs/example/se.py –c /acsir/FFT*

This command is also used to run Ocean, LU and Radix. We only need to change the name of cross-complied file.

Above command will run some apps and kernels of SPLASH2 benchmark in default configuration. Later we will also discuss about the result of these programs for snooping as well as directory based protocol.

Running splash2 for directory based protocol

Running SPLASH2 for directory based protocol means that the default protocol is changed from MOESI to some directory protocol and also ruby memory model is followed. We ran FFT, LU and Radix for directory protocols also. Commands are same as for snoopy.

IV. TEST RESULTS:

Performance parameters are obtained by testing a network (simple / Garnet) with different network configurations (changing number of CPUs and L1 data cache size).

For any network, memory elements are equal to num_dirs i.e. number of directories. For the Tables 1A, 1B and 1C we have aggregated the values across all the controllers while obtaining the cell entries, apart from the Data Bus Utilization parameter, which is the maximum of all controller values recorded here. The output obtained is given in various tables numbered 1A to 1C.

Comparison of Mesh and Crossbar topology

From the output obtained, we can clearly see that mesh performs mostly better than crossbar and this is true for both simple and garnet networks. But in mesh as the size of network increases, performance degrades. We have also seen that on changing L1 data cache size and keeping same number of CPUs hardly affects the results. This means that the native program 'ruby_random_test.py' doesn't take cache size into account.

But ideally there must be less conflict and hence less delay with increasing memory size.

TABLE 1A: SIMPLE NETWORK (DEFAULT MODEL: MI_EXAMPLE)

	Crossbar			Mesh_XY		
	#CPU =16, L1 size = 256B	#CPU =32, L1 size = 256B	#CPU =64, L1 size = 1024 B	#CPU=16, L1 size = 256B	#CPU =32, L1 size = 256B	#CPU= 64, L1 size = 1024B
Number of read requests accepted	673	544	368	668	540	385
Number of write requests accepted	646	496	301	641	496	320
Average queuing delay per DRAM burst	8.85	7.88	7.88	8.81	*7.94*	8.05
Average memory access latency per DRAM burst	27.85	26.88	26.88	27.81	26.94	27.05
Data bus utilization in percentage	1.26	0.64	0.35	1.26	0.63	0.34

The queuing delay and memory latency values decreases as we go from 16 CPU to 32 CPU as we keep the L1 size constant and increases as we go to 64 CPU and vary the L1 size also. This is true for all cases. The reason fro the dip in value from 16 to 32 may be due to the fact that the cache data items are now more widely distributed among more cores. Hence pressure on any one directory controller gets reduced. When we go to 64 cores however, we increase the L1 cache size. This causes more memory caching per cache controller and hence the queue delay and memory latency values again start creeping up.

Comparison of simple and garnet2.0 networks

From above obtained output it is quite clear that simple network is mostly better as compared to garnet2.0. This is evident from mostly smaller average queue lengths and average memory access latency values. Max data bus utilization is however a bit better for the garnet network.

Program using Snoopy Protocol – "memtest.py"

We ran memtest.py program. It uses snoopy protocol. Here we will discuss about the variation of snoopy traffic and number of write backs due to change in cache associativity, L1 data cache size and cache block size. We will keep two of these fixed and vary the third, and will analyze the output. Tables containing data are given in 2A through 4B.

TABLE 1B: SIMPLE NETWORK (MESI_TWO_LEVEL)

	Crossbar			Mesh_XY		
	#CPU= 16, L1 size = 256B	#CPU= 32, L1 size = 256B	#CPU= 64, L1 size = 1024B	#CPU= 16, L1 size = 256B	#CPU= 32, L1 size = 256B	#CPU= 64, L1 size = 1024B
Number of read requests accepted	769	748	706	742	670	600
Number of write requests accepted	704	686	646	677	612	554
Average queuing delay per DRAM burst	11.23	8.29	8.37	9.95	8.68	11.22
Average memory access latency per DRAM burst	30.23	27.29	27.37	28.95	27.68	30.22
Data bus utilizatio n (%)	0.66	0.38	0.2	0.64	0.27	0.13

TABLE 1C: GARNET NETWORK (MESI_TWO_LEVEL PROTOCOL)

	Crossbar			Mesh_XY		
	#CPU =16, L1 size = 256B	#CPU =32, L1 size = 256B	#CPU =64, L1 size = 1024 B	#CP U=16 , L1 size = 256B	#CPU =32, L1 size = 256B	#CPU= 64, L1 size = 1024B
Number of read requests	750	673	657	728	631	552
Number of write requests	692	620	609	677	590	513
Average queuing delay per DRAM burst	10.72	10.83	10.25	10.17	9.67	12.05
Average memory access latency	29.72	29.83	29.25	29.17	28.67	31.05
Data bus utilization in percentage	0.45	0.25	0.14	0.43	0.2	0.1

TABLE 2A: IMPACT DUE TO VARIATION OF ASSOCIATIVITY (L1=32KB AND CACHE BLOCK SIZE=64BYTES)

Cache associativity	Snoop Traffic	Write backs
2:2:1	17678196	1946967
4:3:1	63890449	1223231
4:1	15384779	2613350

TABLE 2B: IMPACT DUE TO VARIATION OF ASSOCIATIVITY (L1=32KB AND CACHE BLOCK SIZE=32BYTES)

Cache associativity	Snoop Traffic	Write backs
2:2:1	17607414	1942117
4:3:1	63861928	1241756
4:1	15403853	2621375

From the above two tables 2A and 2B we see that with change in associativity snoop traffic changes. The snoop traffic for cache associativity having more number of cores is large. Due to large number of cores, probability of having copy of a data is large. If that data is requested, then all other copies need to be invalidated or updated.

TABLE 3A: IMPACT DUE TO VARIATION OF CACHE BLOCK SIZE (L1=32KB AND ASSOCIATIVITY=2:2:1)

Cache Block Size	Snoop data	Write backs
32	17607414	1942117
64	17678196	1946967
128	17610164	1926840

TABLE 3B: IMPACT DUE TO VARIATION OF CACHE BLOCK SIZE (L1=32KB AND ASSOCIATIVITY=4:1)

Cache Block Size	Snoop data	Write backs
32	15403853	2621375
64	15382779	2613350
128	15459231	2614283

TABLE 4A: IMPACT DUE TO VARIATION OF L1 SIZE (ASSOCIATIVITY=2:2:1 AND CACHE BLOCK SIZE=64BYTES)

L1 Size	Snoop data	Write backs
32	17678196	1946967
64	19045255	512745
128	19426433	Nil

TABLE 4B: IMPACT DUE TO VARIATION OF L1 SIZE (ASSOCIATIVITY=4:1 AND CACHE BLOCK SIZE=32BYTES)

L1 SIZE	SNOOP DATA	WRITE BACKS
32	15403853	2621375
64	16434913	972329
128	16931407	Nil

With increase in L1 cache size, number of blocks in cache increases and hence more number of copies can be accommodated. In this case, whenever a data is requested, memory hits will be more and so number of invalidation or updation increases. It means that there will be more snoopy traffic with increase in cache block size. This is clear from the above two tables 4A and 4B.

SPLASH2 for default protocol-MI_example

TABLE 5: "FFT" SIMULATION

Processor	Computation Time	Transpose Time
0	4833	40
1	4833	40
2	4833	40
3	4833	40
Average	4833	40
Minimum	4833	40
Maximum	4833	40

TABLE 6: "RADIX" SIMULATION WITH N=2048 KEYS

Processor	Total Time	Rank Time	Sort Time
0	49	37	8
1	49	39	9
2	49	37	8
3	49	40	8
Average	49	38	8
Minimum	49	37	8
Maximum	49	40	9

We repeated the above experiments using MESI_Two_Level and MOESI_CMP_directory protocols with other parameters kept constant. We saw very similar results as in the case of MI for all cases. This means this feature is not implemented in ARM, and actually tallies with the conformance matrix given in GEM5 documentation [10]. In order to a full evaluation for ARM, SE mode is not sufficient; we need to use the FS (Full System) mode.

TABLE 7: "LU" SIMULATION FOR 64*64 MATRIX

Processor	Total Time	Diagonal Time	Perimeter Time	Interior Time	Barrier Time
0	2831	231	323	1085	1192
1	2831	0	652	1085	1094
2	2831	230	328	1082	1191
3	2831	0	651	547	1633
Average	2831	115	488	950	1278
Minimum	2831	0	323	547	1094
Maximum	2831	231	652	1085	1633

V. CONCLUSION AND FUTURE WORK

We have studied simple and garnet networks using both crossbar and mesh topologies in each. We have seen that mesh is better than crossbar for both simple and garnet networks, but in mesh performance degrades as the size of network increases. Snoopy protocol has been studied and analyzed by varying cache size and block size. Directory protocol techniques have been studied by using SPLASH-2 and GEM5.We have observed that performance measured in form of computation time remains similar from MI to MESI_TWO_LEVEL to MOESI_CMP_directory.

Below is a list of things which can be done to augment the research findings:

- Design of a queue to further reduce collisions in NoCs can be proposed and further analysis of the same can be done using network calculus.
- We are exploring only 2D chips. 3D version of NoCs can be implemented.
- Better coherence protocol can be implemented which gives good bandwidth, less traffic and takes less time.
- We are using only mesh and crossbar topology. Both have certain limitations. A better topology like hierarchical can be developed in future which will solve these problems.
- Here we have run only FFT, radix and LU. Other apps and kernel programs can also be tested.

REFERENCES

[1] Tanya Shreedhar and Sujay Deb, "Hierarchical Cluster based NoC design using Wireless Interconnects for Coherence Support", VLSI Design Conference India (VLSID '16) 2016.

[2] Jongman Kim, Dongkook Park and Reetuparna Das, "A Novel Dimensionally-Decomposed Router for On-chip Communication in 3D Architectures" ", ACM SIGARCH 2007.

[3] Anoop Tiwari, "Performance Comparison of Cache Coherence Protocol on Multi-Core Architecture", M.Tech. Thesis, NIT Suratkal, 2014.

[4] Vikas B., "Gem5 Splash2 Benchmarks Report", NIT Suratkal.

[5] Babak Aghaei and Negin Zaman-Zadeh, "Evaluation of Cache Coherence Protocols in terms of Power and Latency in Multiprocessors" , 3rd International Conference on Research in Engineering, Science and Technology, 2016.

[6] Anthony Gégo, "Study and performance analysis of cache-coherence protocols in shared-memory multiprocessors", M.Tech. Thesis, EPL, UCL, 2015-'16.

[7] Muthukumar.S and Dhinakaran.K, " Hybrid Cache Coherence Protocol for Multi-Core Processor Architecture", International Journal of Computer Applications, Volume 70– No.14, May 2013.

[8] Xing Han, Jiang Jiang, Yuzhuo Fu, and Chang Wang, " Reconfigurable Many-Core Processor with Cache Coherence", 17th CCF Conference, NCCET 2013, July 2013

[9] Somdip Dey and Mamatha S. Nair, "Design and Implementation of a Simple Cache Simulator in Java to Investigate MESI and MOESI Coherency Protocols.", in International Journal of Computer Applications 87.11 (2014).

[10] GEM5 Simulator, available from web page: http://www.gem5.org/Main_Page

[11] Interconnection networks, available from webpage: http://www.m5sim.org/Interconnection_Network