Design and Performance Evaluation of Cache Coherence Protocol for Heterogeneous Chiplet Systems

JiaCheng Mei
NanJing University of Posts and Telecommunications, China
Email: mj1468856453@gamil.com

***Abstract*—In this paper, we comprehensively explore the design space of cache coherency for systems based on Chiplet technology, and analyze the cache coherency protocol bottlenecks, architectural design bottlenecks, and on-chip network topology bottlenecks that may be encountered in the design. We propose two solutions for the protocol bottleneck: aggregating the states of several blocks into a single region state and delivering the data required by the GPU generated by the CPU directly to the GPU's cache. For the architectural bottleneck, we propose a solution of dividing the directory and the memory controller to improve the parallel processing of the directory. We then combine the partitioning solution and the on-chip network topology of the system to propose a further improvement: separate routers for each partitioned directory and memory controller to address possible on-chip network bottlenecks. We designed and validated the latter two schemes experimentally by gem5 simulator. The experimental results are analyzed, compared, and synthesized from both Speedup and VPC perspectives. The experimental results show that 2-way partitioning of the directory and memory controllers and setting up a separate router for each partitioned directory and memory controller is the best solution, which can improve the overall system performance by 1.5% and requires less hardware support compared to 4-way partitioning and 8-way partitioning.**

***Index Terms*—Cache coherence, Chiplet, Heterogeneous system, On-chip Networks**

I. Introduction

Multi-Chiplet systems often require data sharing between different Chiplets to promote parallel computing and task collaboration, so that multiple Chiplets can exchange information and assign tasks through data sharing, improve parallel computing efficiency and cooperation effects, and then improve the overall performance of the system. However, the key prerequisite for achieving this goal is the need to ensure the consistency and correctness of the data in the Chiplets in each module as well as in the memory module, which means that in a multi-Chiplet system, when different Chiplets access the shared data at the same time, it is necessary to ensure that the data seen by all the Chiplets is the same, which avoids the errors and abnormal behaviors caused by the data inconsistencies, and ensures the correctness of the data. The mechanism to ensure this correct data sharing is the cache coherence protocol.

The main objective of this paper is to explore the cache coherence design space for heterogeneous Chiplet systems and design relevant experiments to analyze and verify the functionality as well as the performance of the relevant schemes. The main research contents are as follows: (1) Analyze the possible performance bottlenecks in heterogeneous integrated CPU-GPU systems, including the possible bottlenecks in protocols, architectures, and on-chip networks. (2) The performance impact of the number of directories on heterogeneous integrated CPU-GPU systems is analyzed. Dividing directories is actually a parallelization technique. Allowing each directory to manage a certain range of address space blocks separately can effectively alleviate the possibility of directory access conflicts. Originally, a directory can only receive requests serially, whereas after partitioning, multiple requests can be processed in parallel without the delay caused by contention for a message cache queue. (3) The performance impact of on-chip networks on heterogeneous integrated CPU-GPU systems is explored. For heterogeneous systems, the important data communication is very much dependent on the on-chip network system, and thus the on-chip network can easily affect the performance of the whole system. This paper mainly compares the impact of two different network topologies on the system.

II. Related Works

The cache coherence problem of Chiplet is essentially the cache coherence problem of heterogeneous systems, so it can be drawn from the research on cache coherence for heterogeneous systems as well as multicore systems. Moshovos et al. [1] proposed JETTY, which filters incoming cache coherency requests based on the contiguous region property. taeweon Suh et al. [2] proposed two novel integration techniques, bypass and bookkeeping, to solve the cache coherency problem for non-shared-bus heterogeneous systems. Cache coherency compatibility for Multiprocessor System-On-Chip (MPSoC).J.F. Cantin et al. [3] proposed coarse-grained coherency tracking for filtering a large number of consecutively identical requests from the core.N. D. Enright Jerger et al. [4] proposed a Virtual Tree Coherence (VTC), a coherence protocol that relies on virtual ordered interconnections, to simplify the coherence protocol and provide reliable bandwidth.

Isaac Gelado et al. [5] proposed a new programming model for heterogeneous computing, called Asymmetric Distributed Shared Memory (Asymmetric Distributed Shared Memory, ADSM), which maintains a shared logical memory space

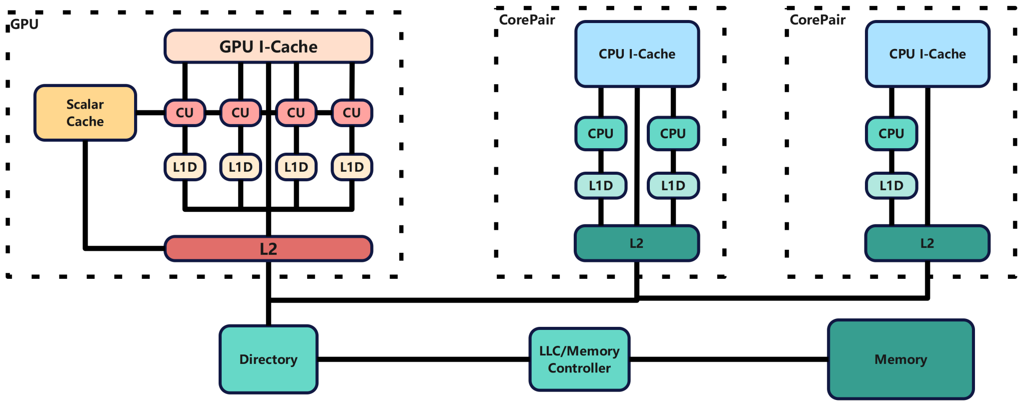

Fig. 1: Our baseline system.

for the CPU, allowing the CPU to access objects in the physical memory of the gas pedal, and vice versa.J. Lee et al. [6] proposed a hardware and software combination of data prefetching technique MT- prefetching, which combines traditional stride prefetching with hardware-based inter-threading techniques, where each thread prefetches data for threads located in other thread groups.H. Hoffmann et al. [7] proposed a memory model RSP (Remote store programming) for multi-core, which is similar to the Cache Coherent Memory System (Cache Coherent Shared Memory (CCSM) or Direct Memory Access (DMA), which requires less hardware support. Because most programs are more tolerant of write memory (store) latency than read memory (load) latency, RSP sacrifices write memory latency for low read memory latency.

III. Bottleneck analysis

A. Bottlenecks on Cache Coherence Protocols

1) Protocol Overhead From Data Access Patterns: As an example, the system studied in this paper is mainly oriented to computationally intensive applications, in which the CPU is mainly responsible for data initialization, while the GPU is responsible for specific parallel computing tasks. In terms of cache coherence, the system uses a write-invalidated directory-type protocol. For compute-sensitive applications, GPUs require a large number of cache accesses as well as memory accesses compared to CPUs due to their high throughput requirements, and thus most of the cache coherence messages are generated due to GPU accesses. In addition, GPUs and CPUs have different access modes to memory. GPUs use the Single Instruction Multiple Threads (SIMT) model, and therefore exhibit high spatial locality and low temporal locality in memory accesses. Analyzed from the perspective of the cache coherence protocol, the access requirements of GPUs to these large amounts of contiguous address data are basically the same, as shown in Figure 2, GPUs need to send a write operation coherence request GetM when accessing block 1, and similarly, GPUs need to send a GetM request when accessing blocks 2, 3, and beyond.

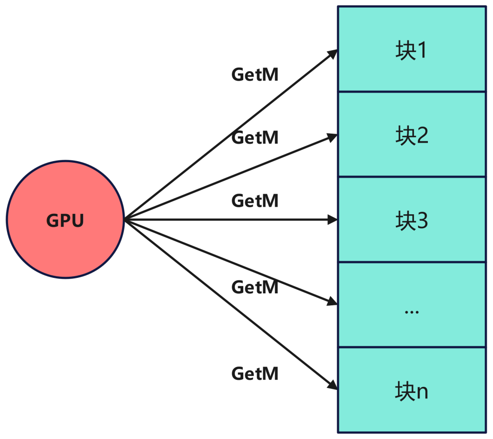

Fig. 2: GPU access pattern.

These operations are almost always the same, but each access generates a large number of duplicate cache coherence messages, which usually cause a lot of stuttering for the GPU, making it impossible to fully utilize the performance of the compute unit, especially for GPU systems with high bandwidth, and this bottleneck is even more significant. Another problem caused by such repeated accesses is high bandwidth and resource usage. For each GPU request, the system's directory has to allocate a Miss Status Handing Register (MSHR) entry to resolve the competition with the CPU-related request, and this register resource is very limited, so when the directory cannot handle these requests fast enough, the GPU will be unable to handle them. requests, the GPU incurs stalls.

2) Efficiency Issues in Data Sharing: The second issue is the efficiency problem caused by the data sharing model. the CPU is the generator of data and the GPU is the corresponding consumer. the CPU generates the data needed by the GPU and then writes this data to the shared memory. the GPU reads the data when it needs it. however, since the CPU always writes the GPU's data to the shared memory, the first time the

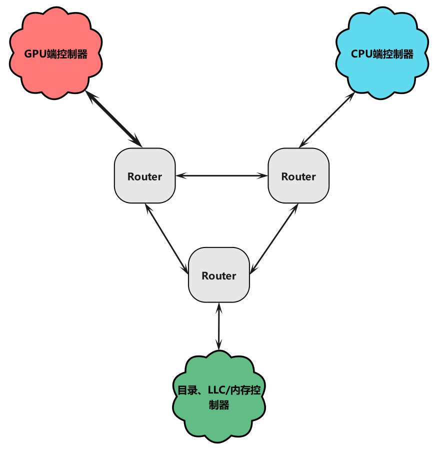

Fig. 3: Our baseline system's network on chip.

GPU reads the data it has to make an access to the shared memory instead of its L1 as well as L2 caches. This leads to a significant increase in mandatory misses in the GPU cache, which results in a certain amount of system stuttering, and the efficiency of GPU-CPU cooperation is greatly reduced.

B. On-chip Network Topology Analysis

The on-chip network topology of the system studied in this paper is shown in Figure 3. This is a relatively simple on-chip network, where Router stands for router, henceforth referred to as R. R1 mainly connects to the consistency controller on the CPU side, R0 connects to the directory/memory controller, and R2 connects to the controller on the GPU side. Since this system is oriented to compute-intensive applications, the communication between GPU and memory is much more than that between CPU and memory, which is reflected in the network that requires a large amount of bandwidth support between GPU and memory, which is represented by a thick line in the figure. As can be seen from the figure, all GPU controllers can only interact with R0, which is connected to the catalog and LLC/memory controllers, through R2, and there is only one bi-directional link connecting R0 and R2. When the GPU sends out a large number of memory access requests, the resources of R0 as well as R2 will be rapidly exhausted, and the connection between R0 and R2 will become highly congested, thus causing the GPU to stall.

C. Bottlenecks in architectural design

Based on the analysis of the on-chip network in the previous subsection, we can find another architectural bottleneck in this system: there is only one directory and LLC/memory controller at R0 to maintain all blocks. All requests from CPUs and GPUs can only be processed serially through this one directory, and GPU requests will take up a lot of resources of the directory and the LLC/memory controller, thus causing subsequent GPU stalls.

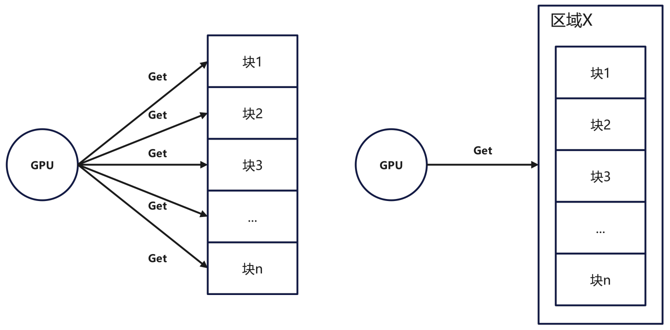

Fig. 4: Aggregate the state of several blocks into the state of a region.

IV. Heterogeneous system cache system optimization

A. Cache Coherence Design Space Exploration

Before describing the specific design and exploration, we briefly describe the cache coherence protocol used by the benchmark system in this paper, as shown in Figure 1.

1) State Aggregation: Combined with the above analysis, a large number of access requests from GPUs will result in a lot of duplicate coherency requests, which will consume a large number of MSHR registers in the directory, and at the same time cause a large number of unnecessary communications in the on-chip network. This problem can be solved by increasing the cache state granularity. Specifically, whereas previously it was the case that each block needed to cache coherence state information, since these blocks may get the same cache coherence requests, they can actually share a single state. Thus, the state of these blocks can be represented uniformly by a regional cache coherence state, as shown in Figure 4.Whereas the original directory was a block-level directory-operating on cache coherence state on a block-by-block basis, with re-regionalization, the directory becomes a region-level directory--operates on cache coherence state on a region-by-region basis. When the GPU makes its first access to a region, it can obtain the corresponding read and write permissions for all blocks from the regional directory, and if the GPU is the only requester for that region, then for the next accesses to the region's data, the GPU can just fetch it directly from the corresponding memory without sending cache coherency requests. Similar to the idea of building a cache, we can also implement a regional directory cache on top of such regional directories, which in turn improves the speed of these cache consistency operations.

2) Data Prefetching: For the data transfer problem, in the original system, the data generated by the CPU must be written to the shared memory first, and then read by the GPU, which leads to an increase in mandatory L1 and L2 misses in the GPU. The solution to this can be found by making a simple change to the architecture. When the CPU generates data, it should first determine if this data will be used by the GPU, and if so, it can put this data directly into the GPU's L1 and L2

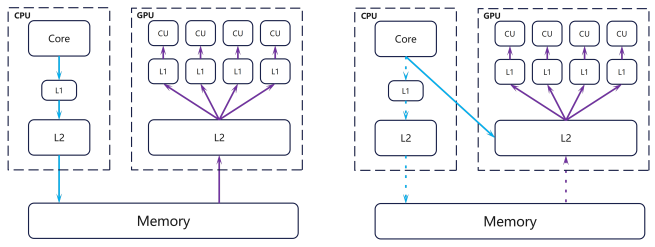

Fig. 5: Data prefetching for the GPU

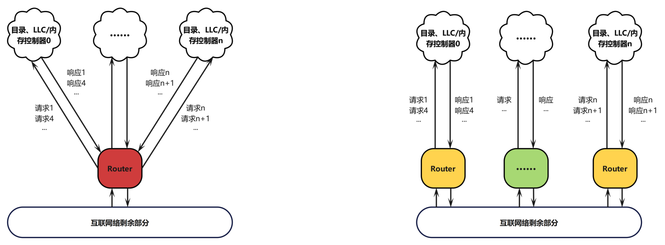

Fig. 7: Division of Diretoy and LLC/memory controllers

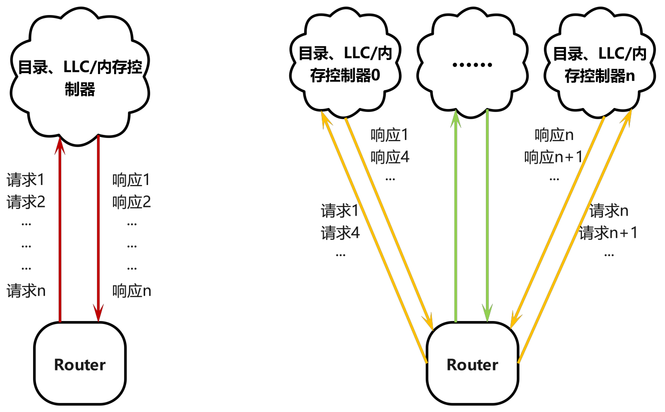

Fig. 6: Division of Diretoy and LLC/memory controllers

caches. However, this comes at the cost of additional dedicated on-chip network connections, as well as some changes to cache coherency to allow the GPU to recognize this data and store it in its own cache.The process is shown in Figure 5.

B. *Directory and LLC/Memory Controller Divisions*

According to the analysis in Section III-C, a single directory as well as LLC/memory controller may create a performance bottleneck because the directory's requests from GPUs and CPUs are processed serially, not in parallel. We can achieve truly parallelized operations by dividing the directories and LLC/memory controllers so that each directory and LLC/controller is responsible for only a portion of the address space. As shown in Figure 6, with only one directory and LLC/memory controller, the directory's operations on these data blocks are serialized and not parallelized; however, when multiple directories as well as memory controllers are set up, it is possible for these data blocks to be assigned to different controllers, each of which can process the data independently. The price of this parallelization is higher on-chip network requirements: each controller that has been partitioned has to be connected to R0.

C. *On-chip Network Topology Optimization*

The previous vignette mainly addresses bottlenecks related to the catalog and the LLC/memory controller. On this basis, data processing at the R0 end should be faster. However, the bottleneck from R0 to R2 still exists, even though all data requests are eventually assigned to different directories and LLC/memory controllers, they actually pass through R0 as a router, and thus the requests and responses to these data are still limited by the resources of R0 - the capacity of each buffer queue in R0 is limited . If there are enough requests from other routers and response messages from the directory and LLC/memory controllers, then the capacity of each buffer queue in R0 is rapidly depleted, and requests/responses that do not make it into the buffer queue must wait until the buffer queue has space available. To solve this problem, we can assign a router to each directory and LLC/memory controller as shown in Figure 7.

The optimized topology has both advantages and disadvantages. The advantage is that each controller can communicate with the router on the GPU or CPU side through its own router without having to compete for the only router resource. This ”triage” operation can help alleviate contention in the buffer queue, thereby reducing the wait time for request or response messages and reducing congestion in the network. The disadvantage of this design is that the hardware overhead required for reconfiguration increases rapidly as the number of divisions increases. Specifically, each subcontroller requires a separate router and link, and since the chip area is limited, too many routers as well as links take up a large amount of the chip area and pose a very difficult layout problem.

V. Evaluation

In this paper, we focus on the simulation of heterogeneous CPU-GPU systems using the apu configuration in gem5 and we evaluate our design using the DNNMark test set.DNNMark is a benchmarking framework for measuring deep neural networks based on their performance.The specific parameters of the simulation system are shown in Table I

A. *Directory and LLC/Memory Controller Division Experiments*

We have selected three divisions: 2-division, 4-division, and 8-division. Figure 8 shows the system structure of the baseline system after the three divisions. The execution time acceleration of the three divisions normalized to the baseline system is shown in Figure 9. From the Figure 9, we can see that all three divisions bring some performance improvement to the system, and more divisions bring more performance improvement. In terms of individual divisions: 2 divisions bring an average performance improvement of 1.4%, up to 7.1%; 4 divisions bring an average performance improvement of 1.9%, up to 8.0%;

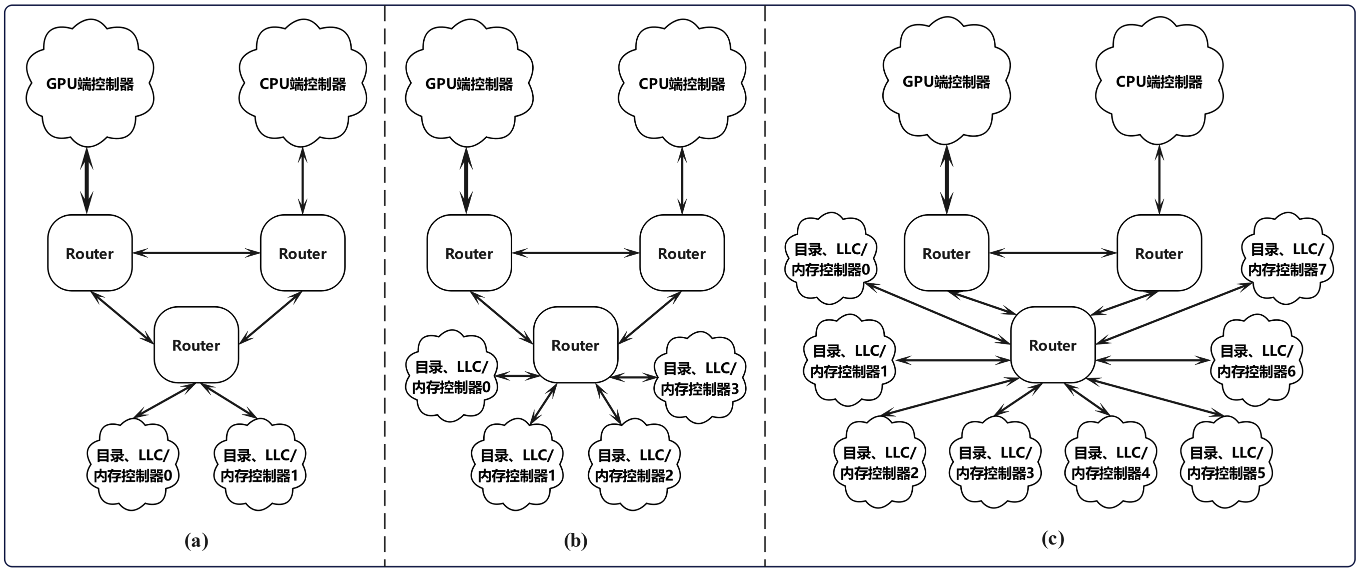

Fig. 8: System structure after various divisions. Figures (a), (b), and (c) represent the structure of the system after 2, 4, and 8 divisions, respectively.

TABLE I: Parameters of the simulation system.

CPU config	3 CPU core ,1 thread per core
CPU L1I Cache	32KB-2way,64B Cache line
CPU L1D Cache	64KB-2way,64B Cache line
CPU L2 Cache	2MB-8way,64B Cache line
LLC(L3) Cache	16MB-16way,64B Cache line
GPU config	4CU per SQC
GPU L1I Cache	32KB-8way,64B Cache line
GPU L1D Cache	256KB-16way,64B Cache line
GPU L2 Cache	16KB-16way,64B Cache line
Dir	1/2/4/8
Memory	8GB

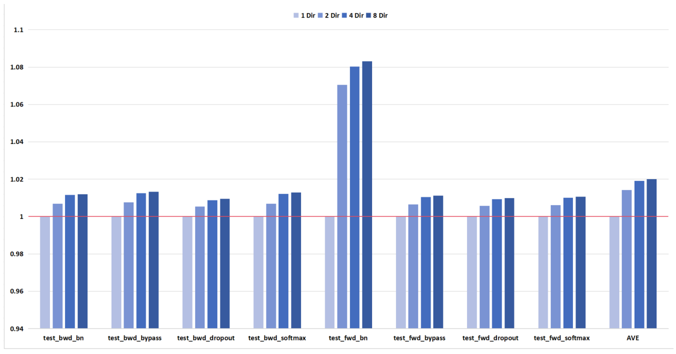

Fig. 9: Speedup of the three divisions normalized to the baseline system

and 8 divisions bring an average performance improvement of 2.0%, up to 8.3%. Between the divisions: 2 divisions gives about 1.4% performance improvement over no divisions, 4 divisions gives less than 1% improvement over 2 divisions, and 8 divisions gives almost no improvement over 4 divisions. Because of the additional hardware overhead associated with segmentation, more segmentation of the controller is not better.

B. On-chip Network Topology Design Space Exploration

We reconstructed the on-chip network topology for each division based on the division experiments. The reconfigured on-chip network structure for each division is shown in Figure 10. Figure 11shows the Speedup normalized to the baseline system after changing the network topology for each division based on the division experiment. According to figure 11, connecting the partitioned directory and LLC/memory controller to a router separately can get better results. 2-partitioned topology brings an average performance improvement of 1.5%, 4-partitioned topology brings an average performance improvement of 2.1%, and 8-partitioned topology brings an average performance improvement of 2.2%, which is almost the same as the 4-partitioned topology.

VI. Conclusion

This paper provides a comprehensive exploration of the design space for heterogeneous Chiplet cache coherency, analyzing the cache coherency protocol bottlenecks, architectural design bottlenecks, and on-chip network topology bottlenecks that may be encountered in the design. (1) For the protocol bottleneck, we propose two solutions: aggregating the states of several blocks into a single region state and delivering the data generated by the CPU and needed by the GPU directly to the GPU's cache. (2) For the architectural bottleneck, we propose a solution of dividing the directory and LLC/memory controllers to improve the parallel processing of the directory.

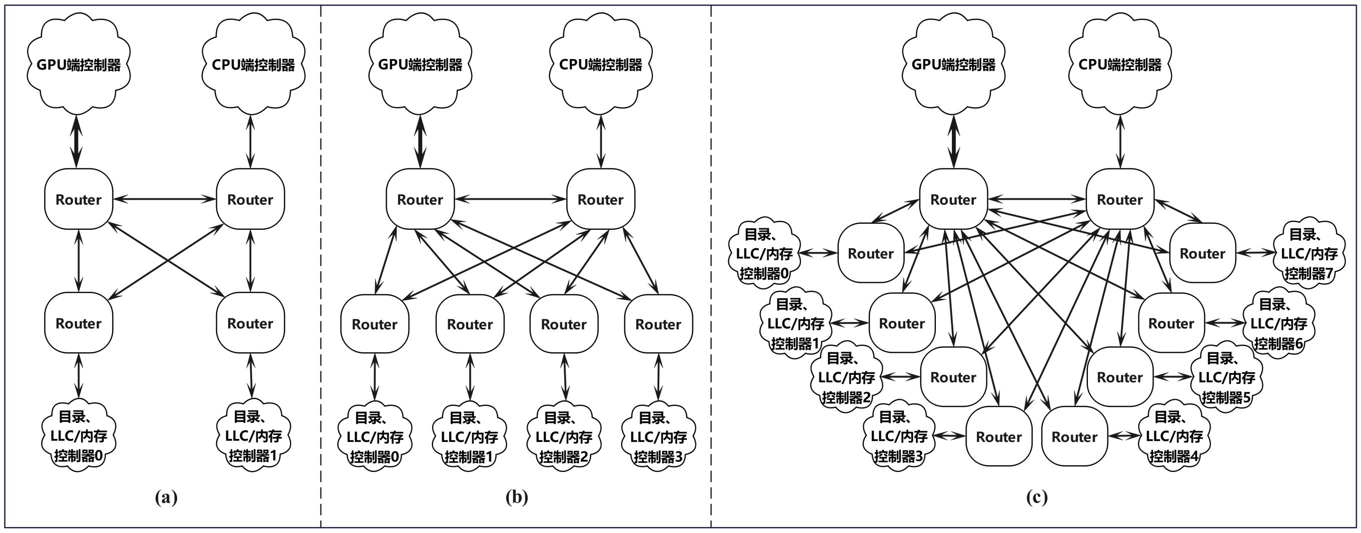

Fig. 10: Reconfiguration diagram of the network on the basis of each division. Figures (a), (b), and (c) represent the graphs after reconfiguring the network topology on the basis of 2 divisions, 4 divisions, and 8 divisions, respectively.

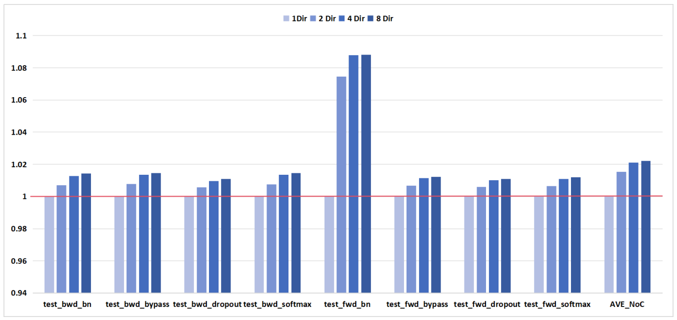

Fig. 11: Speedup of the three divisions normalized to the baseline system

(3) We then combine the partitioning scheme and the on-chip network topology of the system to propose a further improvement: separate routers for each partitioned directory and LLC/memory controller to solve the possible on-chip network bottlenecks. We designed and validated the latter two schemes experimentally by gem5 simulator. The experimental results are also analyzed, compared, and synthesized in terms of speedup ratio. The experimental results show that 2-partitioning the directory and LLC/memory controllers and setting up a separate router for each partitioned directory and LLC/memory controller is the best solution, which can improve the overall system performance by 1.5% and requires less hardware support compared to 4-partitioning and 8-partitioning.

Acknowledgment

Thank NJUPT.

References

[1] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary, "Jetty: Filtering snoops for reduced energy consumption in smp servers," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 85–96.

[2] T. Suh, D. Kim, and H.-H. S. Lee, "Cache coherence support for non-shared bus architecture on heterogeneous mpsocs," in *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 553–558.

[3] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Improving multiprocessor performance with coarse-grain coherence tracking," in *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 2005, pp. 246–257.

[4] N. D. E. Jerger, L.-S. Peh, and M. H. Lipasti, "Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2008, pp. 35–46.

[5] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W.-m. W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," in *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*, 2010, pp. 347–358.

[6] J. Lee, N. B. Lakshminarayana, H. Kim, and R. Vuduc, "Many-thread aware prefetching mechanisms for gpgpu applications," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 213–224.

[7] H. Hoffmann, D. Wentzlaff, and A. Agarwal, "Remote store programming: A memory model for embedded multicore," in *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 2010, pp. 3–17.