Thesis for The Degree of Licentiate of Engineering

Performance Analysis and Enhancements of Memory Systems for Multi-Chiplet NUMA Architectures

Neethu Bal Mallya

Division of Computer and Network Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Performance Analysis and Enhancements of Memory Systems for Multi-Chiplet NUMA Architectures

Neethu Bal Mallya

Advisor:	Ioannis Sourdis, Chalmers University of Technology
Co-Advisors:	Bhavishya Goel, Chalmers University of Technology
	Evangelos Vasilakis, ZeroPoint Technologies
Examiner:	Per Stenström, Chalmers University of Technology
Discussion Leader:	Cristina Silvano, Politecnico di Milano

Department of Computer Science & Engineering
Division of Computer and Network Systems
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2025.

Abstract

As the semiconductor industry is struggling with the slowdown in Moore's Law and the challenges of increased design complexity on a single chip, multi-chiplet systems have become a promising alternative to large monolithic systems, offering improved yields and lower costs. However, these chiplet-based architectures are susceptible to non-uniform memory access (NUMA) inefficiencies, where remote memory access significantly hinders the system's performance. These performance bottlenecks are augmented by the high latency and limited bandwidth of the inter-chiplet communication, thus compromising the overall performance of multi-chiplet systems. While prior studies have thoroughly examined the yield and cost benefits of multi-chiplet chips, their performance relative to monolithic counterparts remains unexplored. This thesis delves into a comprehensive performance analysis of multi-chiplet systems, comparing them to traditional monolithic designs and evaluating their cost-performance trade-offs. While multi-chiplet systems can drastically reduce recurring engineering costs by nearly half, our analysis reveals that they may suffer performance losses of up to one-third compared to monolithic systems due to these NUMA-related overheads. To address the performance overheads, this thesis introduces MEMPLEX, a novel memory system explicitly designed for multi-chiplet NUMA architectures. MEMPLEX combines data replication and migration strategies to optimize data placement and improve data locality within the multi-chiplet memory hierarchy. By allocating a portion of each memory node as a DRAM cache and enabling migration based on access patterns and memory traffic, MEMPLEX reduces the frequency of costly remote memory accesses, mitigates performance overheads, and delivers substantial energy savings. The evaluation on multi-programmed workloads from different benchmark suites demonstrated that, compared to a multi-chiplet system with NUMA-aware data placement and no support for DRAM caching or migration, MEMPLEX reduces remote memory traffic by 80%, leading to a significant 44% dynamic memory energy consumption. MEMPLEX also delivers up to 7% speedup (5% on average) when $\frac{1}{16}$ of each HBM is dedicated for caching in a 4-chiplet system, with performance gains increasing up to 15% (10% on average) in 16-chiplet systems. Overall, this thesis provides insights into the design and optimization of multi-chiplet architectures, paving the way for scalable and efficient systems in the post-Moore's Law era.

Keywords: Chiplets, Non-Uniform Memory Access, Caching, Migration

Acknowledgments

I am immensely grateful to my advisor, Ioannis, for his unwavering dedication, patience, and firm guidance throughout this journey. I am deeply thankful to my co-advisor, Bhavishya, for his profound expertise and steadfast support. Thank you both for your confidence in me and for strengthening my mindset of perseverance.

I also extend my gratitude to my former co-advisor, Evangelos, for his invaluable understanding and insights, especially regarding simulations during the early stages of this work. A heartfelt thanks to my collaborators, Panagiotis and Ahsen, for sharing their valuable insights and enriching my research.

I am truly grateful to my fellow PhD students, Magnus and Panagiotis, for all the insightful discussions, good-natured support and camaraderie along the way. I also extend my heartfelt thanks to my examiner, Per, for his thoughtful feedback and guidance. I sincerely appreciate Arne, Elad, and Miquel for their timely support with processing power and Monica for her remarkable administrative assistance.

Lastly, I sincerely thank all my past and present colleagues at Chalmers: Prajith, Stavroula, Lars, Madhavan, Mehrzad, Sonia, Jing, Minyu, Hari, Qi, Piyumal, Konsta, Fareed, Mateo, Pedro, Mo, and many others, for fostering an inspiring and supportive environment that has been instrumental in my journey.

This work is supported by the Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] Neethu Bal Mallya, Panagiotis Strikos, Bhavishya Goel, Ahsen Ejaz, and Ioannis Sourdis
"A Performance Analysis of Chiplet-Based Systems"
To appear in 2025 Design, Automation and Test in Europe Conference and Exhibition (DATE), Lyon, France, Mar 2025.

[B] Neethu Bal Mallya, Bhavishya Goel, and Ioannis Sourdis
"MEMPLEX: A Multi-Chiplet NUMA Architecture with Data Replication and Migration"
Under review.

Contents

Abstract iii

Acknowledgments v

List of Publications vii

1 Introduction 1
- 1.1 Problem Statement . . . 2
  - 1.1.1 Cost-Performance Trade-offs of Multi-Chiplet Systems Compared to Monolithic Systems . . . 3
  - 1.1.2 NUMA Challenges in Multi-Chiplet Systems . . . 3
- 1.2 Thesis Objectives and Contributions . . . 3
  - 1.2.1 Evaluating Cost-Performance Trade-offs of Multi-Chiplet Systems Compared to Monolithic Systems . . . 4
  - 1.2.2 Mitigating NUMA Challenges in Multi-Chiplet Systems 5
- 1.3 Thesis Outline . . . 7

2 A Performance Analysis of Chiplet-Based Systems 9
- 2.1 Introduction . . . 10
- 2.2 Overview of Chiplet-Based Architectures . . . 11
  - 2.2.1 Chiplet-based NoCs . . . 12
  - 2.2.2 NUMA-aware Memory Allocation . . . 13
  - 2.2.3 Last Level Cache Organization . . . 13
  - 2.2.4 Yield and Cost . . . 14
- 2.3 Experimental Methodology . . . 15
  - 2.3.1 System Configuration . . . 15
  - 2.3.2 Simulation Setup . . . 16
  - 2.3.3 Workloads . . . 17
  - 2.3.4 Evaluated Systems . . . 18
- 2.4 Performance Evaluation . . . 18
- 2.5 Conclusions . . . 22

3 MEMPLEX: A Memory System with Replication and Migration of Data for Multi-Chiplet NUMA Architectures 23
- 3.1 Introduction . . . 24
- 3.2 Related Work . . . 25
  - 3.2.1 Non-Uniform Memory Access in Shared Memory Systems 25
  - 3.2.2 Hybrid Memory Systems . . . 26
  - 3.2.3 Software support in NUMA machines . . . 27

3.3 MEMPLEX Design 28
3.3.1 MEMPLEX System Overview 28
3.3.2 DRAM Cache Controller 29
3.3.3 Memory Layout & Metadata 31
3.3.4 Memory Access Path 32
3.3.5 Allocating a Sector in Local Memory 34
3.3.6 DRAM Cache Evictions 35
3.3.7 Migration Decision and Traffic Regulation 36
3.3.8 An Example Illustration 37
3.3.9 Cache Coherence 38
3.4 Experimental Setup 39
3.4.1 System Configuration 39
3.4.2 Simulation Setup 40
3.4.3 Workloads 41
3.4.4 Evaluated Systems 41
3.5 Evaluation 42
3.5.1 Performance 42
3.5.2 Memory Traffic 44
3.5.3 Energy Consumption 45
3.5.4 Sensitivity analysis on System Size 45
3.5.5 Sensitivity analysis on DRAM Cache Size 46
3.6 Conclusions 46

Bibliography **47**

Appendix **53**

A Additional Results On Cost Analysis of Chiplet-Based Systems **53**

Chapter 1

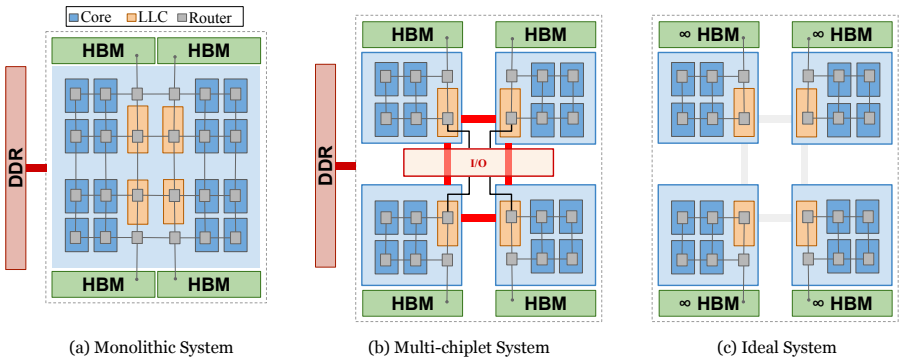
Introduction

In the multicore era, performance scaling effectively relies on integrating more resources onto a chip. The shift occurred because Dennard scaling, which had enabled frequency scaling, was constrained by power limitations. The deceleration of Moore's Law has intensified technology scaling challenges. Large monolithic chips suffer from low yields and high costs, making them less viable. Consequently, building chips from multiple smaller chiplets offers a better cost-effective alternative, providing higher yields and better scalability [1–4].

3D-stacking technology initially involved stacking memory chips, such as High Bandwidth Memory (HBM), and placing them closer to processing die like GPUs [5] or vector engines [6]. This technology has evolved beyond simple memory stacking to enable more complex and efficient chiplet systems, as exemplified in AMD's EPYC and RYZEN architectures [1]. However, despite offering significant advantages in yield and cost, multi-chiplet systems introduce new performance challenges. The size of multi-chiplet systems often necessitates multiple non-uniform access memory (NUMA) nodes, leading to performance bottlenecks that arise from varying memory access latencies. For instance, early AMD EPYC and RYZEN chips connected CPU chiplets to DRAM via a single I/O die, resulting in access latencies that varied by tens of nanoseconds, depending on the DRAM controller being accessed [1]. Multi-chiplet systems have advanced, incorporating more chiplets and complex memory systems, such as in AMD's MI300 [7] and Intel's Sapphire Rapids [8], but the NUMA-related inefficiencies remain a persistent challenge.

This thesis studies the overheads of remote memory access and inter-chiplet communication in multi-chiplet systems and evaluates their impact on system performance compared to traditional monolithic systems. It also proposes solutions to mitigate these challenges in these increasingly prevalent architectures.

The remainder of the introductory chapter is structured as follows: Section 1.1 outlines the problem statement, followed by a discussion of the thesis objectives and contributions in Section 1.2.



(a) Monolithic System (b) Multi-chiplet System (c) Ideal System

Figure 1.1: Hypothetical System Configurations with 16 cores: Monolithic *vs.* Multi-Chiplet *vs.* Ideal

1.1 Problem Statement

In recent years, chiplet architectures have gained attention as a solution to the scaling challenges of traditional monolithic architectures. Multi-chiplet systems offer a low-cost alternative, delivering higher yields and better scalability compared to their monolithic counterparts [1–4]. However, the benefits of chiplet-based systems come with trade-offs, particularly in terms of performance overheads related to remote memory access and inter-chiplet communication.

Without loss of generality and to better understand the trade-offs, Figure 1.1 illustrates typical system configurations for a 16-core multicore architecture, comparing three design paradigms: (a) a monolithic system, (b) a multi-chiplet system, and (c) a hypothetical ideal system. The chip organization and floorplan, inspired by the AMD Zen families [9, 10], consists of compute tiles composed of the core with private L1 and L2 caches and a shared Last-Level Cache (LLC) partitioned in slices. The memory hierarchy incorporates HBM and external DDR DRAM interfaces, with all components interconnected via a Network-on-Chip (NoC). In the monolithic chip, a 2D mesh NoC is employed, with the LLC slices located in the middle columns of the 2D mesh and the HBM and external DDR interfaces placed at the chip's edges. The multi-chiplet system divides the architecture into several chiplets, each containing a subset of tiles, one or more LLC slices, and HBM interfaces, while a separate I/O chiplet manages the external DDR access. Each chiplet utilizes a 2D mesh NoC topology, and inter-chiplet communication is constrained by the chiplet's microbumps budget, as detailed in prior work [11,12]. The high latency and limited bandwidth of the inter-chiplet communication add extra latency compared to the monolithic system. The ideal system serves as a theoretical baseline, assuming infinite capacity in the local HBMs, thus eliminating remote memory accesses and the need for communication across chiplet boundaries.

Remote memory accesses in a NUMA environment increase access latency, while the constrained bandwidth of inter-chiplet links exacerbates communication delays. Consequently, multi-chiplet systems face two primary challenges that limit their performance: (i) NUMA inefficiencies impacting the remote accesses and (ii) inter-chiplet communication overheads. These challenges form the core of this thesis and are discussed in detail in the following sections.

1.1.1 Cost-Performance Trade-offs of Multi-Chiplet Systems Compared to Monolithic Systems

Although the economic effectiveness of multi-chiplet chips has been thoroughly analyzed [2], there is a notable gap in understanding the performance implications of multi-chiplet architectures compared to monolithic designs. Multi-chiplet systems exhibit performance overheads due to the inherent non-uniform memory architecture and inter-chiplet communication. These overheads may negate the yield and cost advantages, making it crucial to evaluate the impact systematically.

This thesis aims to comprehensively evaluate the performance overheads of multi-chiplet systems relative to monolithic architectures and analyze the associated cost-performance trade-offs. The findings will provide valuable insights into the feasibility of multi-chiplet architectures as a scalable and cost-effective alternative to monolithic chips while identifying avenues to mitigate their performance limitations.

1.1.2 NUMA Challenges in Multi-Chiplet Systems

While software can optimize data placement in multi-chiplet NUMA systems, there are currently no hardware mechanisms to improve data placement in DRAM distributed across chiplet nodes at runtime. Our experiments reveal that even with NUMA-aware memory allocation—where data is placed in the closest available memory node relative to the processing node—system performance remains significantly lower than that of an ideal system, which always retrieves data from its local memory node.

While the monolithic system also lags behind the ideal in terms of system performance, the chiplet-based system experiences an even greater slowdown, emphasizing a significant performance gap that must be addressed. This gap arises from the latency and bandwidth penalties associated with remote memory requests, which are inherent to multi-chiplet NUMA architectures. These inefficiencies pose a significant barrier to achieving optimal performance in such systems. Therefore, there is a pressing need for mechanisms that mitigate the performance impact of remote memory accesses.

Together, these two problems—understanding the cost-performance trade-offs of multi-chiplet architectures and addressing NUMA-related inefficiencies—constitute the primary focus of this thesis.

1.2 Thesis Objectives and Contributions

The overarching goal of this thesis is to investigate and optimize the performance of multi-chiplet systems by addressing their inherent NUMA challenges. This involves exploring methods to reduce performance overheads and improve memory access efficiency, specifically by evaluating the impact of different architectural choices and proposing memory system techniques to overcome the existing performance bottlenecks. Below, we outline the specific objectives that drive this thesis and describe the approach taken to achieve these objectives, along with key contributions of the thesis.

1.2.1 Evaluating Cost-Performance Trade-offs of Multi-Chiplet Systems Compared to Monolithic Systems

Objective: The first objective of the thesis is to analyze the performance overheads of chiplet-based systems and evaluate their cost-performance trade-offs compared to monolithic chips. This includes a detailed examination of key design parameters that influence both system performance and cost.

Related Work: The benefits of multi-chiplet systems, particularly in terms of yield and cost, have been thoroughly analyzed in the prior studies [2]. However, these systems are not without performance penalties, primarily due to the need for multiple NUMA nodes and the associated communication latencies. For example, early AMD EPYC and Ryzen processors connected their CPU chiplets to DRAM through a single I/O die, leading to varying access latencies depending on the DRAM controller being accessed [1]. These latencies, often differing by tens of nanoseconds, highlight the inefficiencies in memory access that multi-chiplet systems must overcome.

Thesis Approach: This thesis focuses on evaluating the cost-performance trade-offs of multi-chiplet systems by examining various design parameters, including (i) system size, (ii) chiplet size, (iii) LLC organization, (iv) NoC datapath width, and (v) silicon interposer type (passive vs. active). By analyzing these parameters, the thesis aims to identify configurations that balance performance and cost in multi-chiplet systems. To achieve this, a microarchitectural simulation setup was developed to model large-scale chiplet architectures, and the key design choices and technological factors influencing performance were examined. The performance overheads of chiplet-based systems are evaluated across different design alternatives, with insights drawn from several interconnect and memory system metrics. To complement this analysis, the thesis evaluates the associated costs using the Feng-Ma chiplet actuary model [2], incorporating the parameters of the evaluated systems in our study.

Thesis Contributions: In line with the objective, **Paper A** examined the performance overheads of multi-chiplet systems, which had not been thoroughly analyzed in prior studies, and made the following contributions:

- Developed a microarchitectural simulation setup to model large-scale chiplet-based architectures, with detailed models of their memory system and interconnection networks, capable of simulating about a billion instructions per hour.
- Examined technological factors influencing inter-chiplet link delays, microbump budgets, and the yield and cost of chiplet-based versus monolithic chips, confirming that chiplet-based designs can reduce recurring engineering costs by nearly half.
- Conducted the first systematic performance analysis of chiplet-based systems against monolithic chips, showing that chiplet-based systems

suffer a performance loss of about one-third compared to monolithic. Compared to monolithic, chiplet-based systems:

  - Achieve only 43% to 75% of monolithic performance, averaging 58%, while an ideal system, where LLC misses always go to the closest HBM, is 24% faster than monolithic.
  - Suffer a 40% higher average memory access time, contributing significantly to the performance gap.
  - Experience 3.7× longer average packet latency due to inter-chiplet communication overheads.

- Investigated design parameters such as system size, chiplet size, LLC organization, NoC datapath width, and silicon interposer type, and analyzed their impact on system performance. The key observations include:

  - As the system size increases, the performance gap between monolithic and multi-chiplet systems remains relatively stable.
  - For a constant system size, the performance reduces as the system is disintegrated into more, smaller chiplets.
  - Private LLC improves the system performance by about 30% over Sliced LLC in multi-chiplet systems.
  - Wider intra-chiplet NoC links improve the network throughput, reduce average memory access time, and therefore improve overall system performance.
  - Active interposers effectively recover most of the performance overhead seen in chiplet-based systems with passive interposers but at a 61% higher cost.

Appendix A provide additional results on the cost analysis of various design options for chiplet-based systems, which were not included in **Paper A** due to space constraints. This supplementary material offers further insights and supports the findings presented in **Paper A**.

1.2.2 Mitigating NUMA Challenges in Multi-Chiplet Systems

Objective: The second objective of this thesis is to alleviate this performance degradation in NUMA-based multi-chiplet architectures and bridge the performance gap to ideal systems. The **key insights** behind this work are:

- *Remote memory access in multi-chiplet NUMA systems leads to significant performance degradation due to the high latency and limited bandwidth associated with accessing data located on remote memory nodes.*
- *Caching remote data within the DRAM cache of the local HBM node enhances data locality, thereby reducing the frequency of remote memory accesses.*
- *Migrating data upon the DRAM cache eviction ensures that the most frequently accessed data stays close to the processing chiplet.*

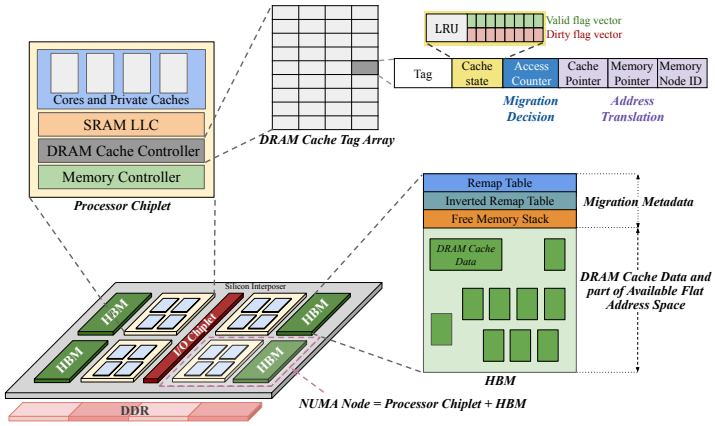


Figure 1.2: MEMPLEX System Overview

Related Work: The performance of multi-chiplet systems is primarily limited by two key challenges: the NUMA-related inefficiencies in memory access and the overheads associated with inter-chiplet communication.

NUMA-related inefficiencies have long been a challenge in computing systems. Early solutions, such as Cache Only Memory Architectures (COMA) [13] and Cache Coherent NUMA (CC-NUMA) [14] memory systems, enhanced the performance of NUMA multi-socket machines by replicating (caching) and/or migrating data closer to the processor chip. More recently, hybrid memory systems, which combine smaller High Bandwidth Memory (HBM) with larger, lower bandwidth external DRAM, have adopted similar techniques to reduce memory access latencies. These hybrid systems use HBM as part of a flat address space, relying on Operating System (OS) support [15] or hardware migration mechanisms [16–21] to place data closer to the processing chiplet, or they use the entire HBM as a DRAM cache for external DDR memory, often wasting valuable main memory capacity [22–31], or employ a combination of both [32].

Another performance overhead in multi-chiplet chips, which exacerbates NUMA effects, stems from the increased latency and bandwidth constraints in inter-chiplet communication. The necessity of traversing longer links through microbumps and silicon interposers introduces additional latency. Moreover, the chiplet size and density limitations restrict the number of available microbumps, capping the available off-chiplet bandwidth and further impacting performance.

Thesis Approach: To address these challenges, this thesis proposes MEMPLEX, a novel memory system designed for multi-chiplet architectures, which incorporates the above insights and techniques to minimize the performance degradation in NUMA-based multi-chiplet architectures. MEMPLEX is a novel memory system that enhances data locality by replicating and migrating data across memory nodes in multi-chiplet systems. MEMPLEX is tailored for chiplet-based architectures, comprising multiple processor chiplets and HBMs integrated on a silicon interposer as well as external DDR memory accessed

via an IO chiplet, as illustrated in Figure 1.2. MEMPLEX increases the number of accesses to the closest memory node for each processing chiplet, thereby minimizing remote memory requests. By leveraging a small fraction of each HBM node as a DRAM cache and intelligently deciding whether to migrate data upon eviction, MEMPLEX reduces average memory access times, improves overall system performance, and provides substantial energy savings.

Thesis Contributions: In line with the objective, **Paper B** proposed memory system enhancements to overcome the existing performance overheads, and made the following contributions:

- Investigated the performance bottlenecks in multi-chiplet NUMA systems, revealing performance losses of 26% and 31% in 4- and 16-chiplet configurations, respectively, compared to an ideal system.
- Proposed MEMPLEX, the first multi-chiplet NUMA architecture that combines replication and migration of data across multiple memory nodes, offering:
  - Most of the HBM capacity as a shared flat address space, unlike designs that use it entirely as a DRAM cache.
  - Superior performance to existing software solutions offering NUMA-aware data placement and designs using HBM exclusively as a cache.
- Evaluated MEMPLEX on multi-programmed workloads from different benchmark suites (detailed in Section 3.4.3) and demonstrated that, compared to a multi-chiplet system with NUMA-aware data placement and no support for DRAM caching or migration, MEMPLEX:
  - Eliminates 80% of remote memory traffic, leading to a 44% decrease in dynamic memory energy consumption in a 4-chiplet system.
  - Achieves up to 7% speedup (5% on average) when $\frac{1}{16}$ of each HBM is dedicated for caching in a 4-chiplet system, with performance gains increasing up to 15% (10% on average) in 16-chiplet systems.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents **Paper A**, "*A Performance Analysis of Chiplet-based Systems*," which addresses the first problem, and Chapter 3 presents **Paper B**, "*MEMPLEX: A Memory System with Replication and Migration of Data for Multi-Chiplet NUMA Architectures*," which addresses the second problem of the thesis.

Paper A

A Performance Analysis of Chiplet-Based Systems

Neethu Bal Mallya, Panagiotis Strikos, Bhavishya Goel, Ahsen Ejaz, and Ioannis Sourdis

2025 Design, Automation and Test in Europe Conference and Exhibition (DATE), Lyon, France, Mar 2025.

Chapter 2

A Performance Analysis of Chiplet-Based Systems

Abstract

As the semiconductor industry struggles to keep Moore's law alive and integrate more functionality on a chip, multi-chiplet chips offer a lower cost alternative to large monolithic chips due to their higher yield. However, chiplet-based chips are naturally Non-Uniform Memory Access (NUMA) systems and therefore suffer from slow remote accesses. NUMA overheads are exacerbated by the limited throughput and higher latency of inter-chiplet communication. This paper offers a comprehensive analysis of chiplet-based systems with different design parameters measuring their performance overheads compared to traditional monolithic multicore designs and their scalability to system and chiplet size. Several design alternatives pertaining to the memory hierarchy, interconnects, and technology aspects are studied. Our analysis shows that although chiplet-based chips can cut (recurring engineering) costs to half, they may give away over a third of the monolithic performance. Part of this performance overhead can be regained with specific design choices.

2.1 Introduction

In the multicore era, integrating more resources on a chip is evermore important for the performance scaling of processors. In the past couple of decades, frequency scaling has been limited by power density, and therefore, delivering performance speedup relies primarily on fitting more cores on a chip. However, technology scaling has become more difficult, and large monolithic chips have low yields and, thus, excessive costs. Building chips out of multiple smaller chiplets is a cheaper, higher yield alternative [1–4].

Die stacking technology has enabled multi-chiplet chips. It was first used for building 3D stacked DRAM chips such as High Bandwidth Memory (HBM) and bringing it closer to processing units, e.g., to a GPU [5] or a vector engine [6]. Later it was employed for disintegrating processors to multiple chiplets, e.g., AMD EPYC and RYZEN architectures, improving yield [1]. Currently, large chips, such as the AMD MI300 [7] and Intel Sapphire Rapids [8], are composed of many CPU and/or GPU chiplets, as well as HBM nodes combining high processing throughput with fast, high-bandwidth memory access.

Despite their improved yield, multi-chiplet chips come with performance overheads. Due to their large size, such systems inevitably use multiple non-uniform access memory nodes. Even early AMD EPYC and RYZEN chips, which provide DRAM access to their CPU chiplets via a single IO die, have a varying access latency by tens of nanoseconds depending on the accessed DRAM controller [1]. AMD MI300 and Intel Sapphire Rapids have even more complex, heterogeneous memory systems composed of multiple HBM nodes and external DRAM. Non-uniform Memory Access (NUMA) machines entail the performance pitfall of long latency remote accesses. Although in the past Cache Only Memory Architectures (COMA) [13] and Cache Coherent NUMA (CC-NUMA) [14] approaches improved data locality and performance of NUMA multi-socket machines, current multi-chiplet chips rely mainly on code optimizations to improve data placement when operating in a flat "HBM + external DDR" mode, or otherwise sacrifice HBM capacity to cache data.

Another performance overhead in multi-chiplet chips, which exacerbates the NUMA effects, is related to the inter-chiplet communication. As opposed to networks on monolithic chips, inter-chiplet connections suffer latency and bandwidth overheads. Exchanging messages with another chiplet requires traversing longer links via microbumps and a silicon interposer, adding extra latency. In addition, the number of available microbumps is limited by the chiplet size and their density constraints, putting a cap on available off-chiplet bandwidth.

Although the yield and cost benefits of multi-chiplet chips have been thoroughly analyzed [2], to the best of our knowledge, the performance with respect to their monolithic counterparts has not been studied. This work fills this gap by evaluating the aforementioned performance overheads of multi-chiplet chips compared to monolithic ones and analyzing the cost-performance trade-off they offer. We explore the following design points in this study: (i) system size, (ii) chiplet size, (iii) Network-on-Chip (NoC) bandwidth, (iv) Last-Level-Cache (LLC) organization, and (v) silicon interposer type (passive vs. active). We measure the performance overheads of chiplet-based chips varying the above design alternatives and analyze them based on the insight provided by several interconnect and memory system metrics.

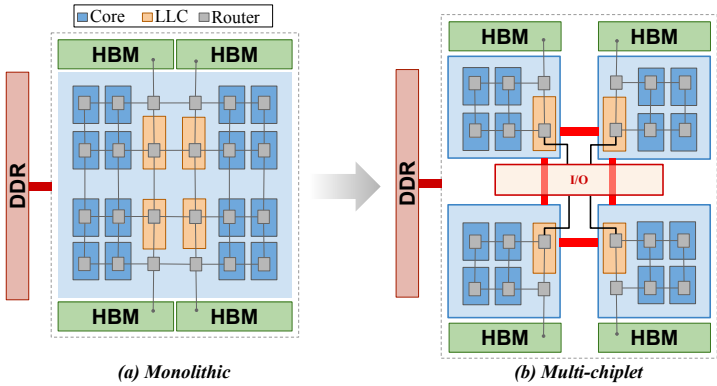


Figure 2.1: Monolithic vs. Multi-Chiplet chips.

Concisely, the contributions of this paper are the following:

- A microarchitectural simulation setup to model large-scale chiplet-based architectures, including detailed models of their memory system and interconnection network;
- The first thorough analysis of performance overheads and cost-performance trade-offs of chiplet-based chips in comparison to monolithic chips, showing that despite their large cost benefit, chiplet-based designs incur a significant impact on system performance;
- An analysis of various technological aspects that determine specific system parameters such as the length and delay of inter-chiplet links, microbumps budget, yield and cost of chiplet-based and monolithic chips;
- Some design choices are identified to regain some of the performance overheads of chiplet-based chips.

The remainder of this paper is organized as follows: Section 2.2 describes the design alternatives analyzed for chiplet-based architectures. Section 2.3 explains our experimental methodology. Section 2.4 presents our evaluation results. Finally, Section 2.5 summarizes our conclusions.

2.2 Overview of Chiplet-Based Architectures

The microarchitecture of the chiplet-based chips studied in this paper as well as the monolithic chips used as baselines, are described next. Without loss of generality, the multicore systems are organized in tiles composed of a core with its private L1 and L2 caches, a shared Last-Level Cache (LLC) partitioned in slices, each being closer to a subset of tiles, as well as HBM and external DDR DRAM interfaces. The above are interconnected via a Network-on-chip (NoC), which dedicates a network router for each tile, LLC slice, HBM node, and external DDR controller. The organization and floorplan of the chips illustrated in Figure 2.1 are inspired by the AMD Zen families [9, 10]. A monolithic chip uses a 2D mesh NoC, has its LLC slices in the middle columns of the 2D mesh, and the HBM and external DDR interfaces at the edges of the chip.

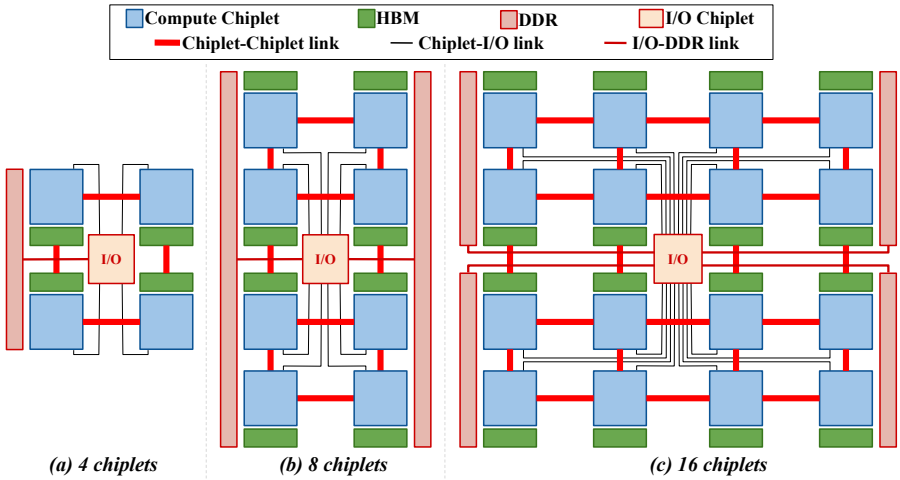


Figure 2.2: Different sizes of multi-chiplet chips.

The chiplet-based counterpart uses chiplets, which contain a subset of tiles, one or multiple LLC slices and HBM interfaces, while access to external DDR is provided via a separate IO chiplet. The NoC topology within the chiplet remains a 2D mesh, and inter-chiplet links are reduced to a number that can be supported by the microbumps budget of the chiplet, similar to previous work [11,12].

Figure 2.2 illustrates the above multi-chiplet organization for different system sizes, i.e., different number of chiplets. It can be observed that the number of HBM nodes scales linearly to the number of chiplets, i.e., one HBM node per chiplet placed next to it. Moreover, the external DDR size and the number of channels also scale linearly to the number of chips.

In the rest of the section, details are provided for system aspects that affect the potential performance overheads of chiplet-based chips. In particular, it describes (i) the design of the interconnects with a focus on inter-chiplet communication and the choice of silicon interposer, (ii) the memory allocation that dictates data placement on the Non-Uniform Memory Access system, and (iii) the choice of the LLC organization. Finally, the yield and cost of the chiplet-based chips are estimated with respect to their monolithic counterparts.

2.2.1 Chiplet-based NoCs

One of the first design choices involves deciding whether to use a passive or active silicon interposer for integrating the chiplets. A passive silicon interposer is cheaper as it requires fewer fabrication steps and offers a higher yield, but it offers slower inter-chiplet links because it does not include buffers. An active interposer offers higher throughput because it includes active components to pipeline the links and potentially lower link latency. It may also include network routers offering more advanced topologies. Additionally, an active interposer benefits from lower clock skew/jitter due to repeaters and easier clock synchronization. Minimally active silicon interposers have been shown to have a small cost overhead compared to passive ones [33,34]. Another design

choice studied in previous work is the placement of inter-chiplet links, showing the benefits of concentrating inter-chiplet links to a few edge NoC routers of a chiplet [11, 12].

In our performance analysis, passive silicon interposers as well as minimally active, i.e., with pipelined links, are explored. The total budget of microbumps per chiplet is estimated based on technology parameters, and inter-chiplet links are concentrated to fewer NoC edge routers, adjusting their width accordingly. Finally, various intra-chiplet NoC datapath widths are explored, offering different intra-chiplet communication bandwidths.

2.2.2 NUMA-aware Memory Allocation

The placement of data in DRAM (HBM and external DDR) is critical for the performance of a NUMA system. Modern operating systems widely support Non-Uniform Memory Access (NUMA) architectures through various mechanisms. For instance, operating systems like Linux [35], Windows [36], and FreeBSD [37] implement NUMA-aware scheduling algorithms that place processes and threads closer to the memory nodes. This approach helps to reduce access latency by optimizing memory locality. Additionally, these operating systems provide APIs that allow user applications to discover the NUMA topology, request memory from specific nodes, and set process affinity, enhancing performance for NUMA-enabled systems.

In this study, we use a Distance-aware Memory Allocation policy. This policy allocates physical memory pages to the memory node closest to the processor core that first accesses the memory. This approach can significantly improve performance by ensuring that memory is allocated in proximity to the accessing core, reducing the latency of memory access.

2.2.3 Last Level Cache Organization

In multi-chiplet systems, the Last-Level Cache (LLC) can be organized through two primary designs:

Sliced LLC: The Sliced LLC architecture, pioneered by Intel starting with the Sandy Bridge microarchitecture, distributes the LLC into multiple "slices" [38]. Each slice acts as an independent cache, but all the slices together form a single logical cache. The physical memory address determines the slice into which data is loaded, effectively distributing memory addresses across slices, and thereby enhancing effective memory bandwidth. In our study, we assign one LLC slice per chiplet and evenly divide the address space among the LLC slices, corresponding to the number of chiplets or High Bandwidth Memory (HBM) nodes associated with a chiplet. The addresses mapped to the external DDR are also divided and assigned into these slices. Accesses from core private caches mapped to the local LLC slice exhibit lower latency, whereas accesses mapped to remote LLC slices have to traverse inter-chiplet links, resulting in higher access latency.

Private LLC: Unlike the sliced LLC architecture where the logical LLC cache is distributed across chiplets, in the private LLC architecture, each chiplet is assigned its own private LLC cache, and the entire address range is mapped to that private LLC. As a result, all the accesses from the core private

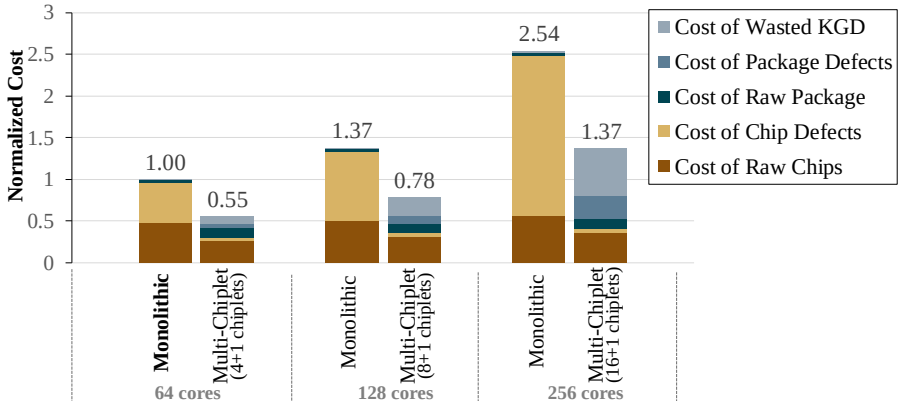


Figure 2.3: Cost analysis and comparison of RE cost of monolithic and multi-chiplet chips for different system sizes. Cost is normalized to that of the smallest monolithic.

caches first go to the local LLC, and only in the case of LLC miss may be required to traverse the inter-chiplet communication link (depending on where the data is mapped in the memory). This design results in reduced inter-chiplet communication overhead compared to the sliced LLC architecture. However, since the same address can now be present in different LLCs across chiplets, LLCs need to be kept coherent, requiring a complex inter-chiplet coherence mechanism. Despite these challenges, the Private LLC approach can offer performance benefits in scenarios where the coherence overhead is manageable or inter-chiplet communication latency is critical.

2.2.4 Yield and Cost

The performance analysis of the above multi-chiplet design alternatives needs to be complemented with an evaluation of their cost with respect to their monolithic alternative. This is performed using the Feng-Ma chiplet actuary model [2] with the parameters of the evaluated systems in our study.

More precisely, our work considers monolithic and multi-chiplet chips which are based on AMD EPYC microarchitecture composed of Zen4/Zen4c CPU chiplets manufactured at 5 nm and an IO chiplet at 14 nm. The chiplet area is estimated, considering 16-core Zen4 chiplets after scaling the L2 and L3 cache sizes to what is used in our performance evaluation, as depicted in Table 3.1. That results in a chiplet size of 66 mm^2, which is similar to the AMD Zen4 chiplets and slightly smaller than the AMD Zen4C chiplet. In addition, each multi-chiplet chip includes an IO chiplet of 400 mm^2, as estimated from AMD EPYC chips of similar technology. Finally, a passive interposer of 65 nm technology is considered.

Based on the above parameters, the above cost model was used to derive the costs of 64-, 128-, and 256-core systems, which are divided in the case of chiplet-based chips to 4+1, 8+1, and 16+1 chiplets, respectively, including the IO chiplet [2]. Figure 2.3 presents a breakdown of the detailed recursive engineering (RE) cost. The total cost is the sum of the following: (i) cost of raw chips which includes among others the cost of the silicon and the processing during the wafer fabrication, (ii) cost of raw package which covers the materials

that are necessary for assembling and packaging the chip as well as testing and verification, (iii) cost of wasted known good dies (KGD) that encapsulates the cost derived from dies that already have been tested to ensure their correct functionality, but still fail, (iv) cost of chip defects, that covers the cost of defects that occur during the wafer fabrication process, and (v) cost of package defects which includes costs related to the packaging process. The total cost is then normalized to that of the smallest monolithic chip.

Overall, the cost of chiplet-based chips is about 55% of that of monolithic chips. The gap between the monolithic and multi-chiplet costs seems to only slightly increase because of the conservative estimation of the model regarding the bonding and packaging multi-chiplet yield. Nevertheless, the model confirms the significant savings of chiplet-based approaches and puts our performance analysis in perspective.

2.3 Experimental Methodology

2.3.1 System Configuration

Our microarchitectural simulation offers detailed modeling of the memory subsystem and interconnection network, as explained in Section 2.3.2, and therefore is computationally intensive for large systems. In order to keep the simulation times of our experiments within affordable bounds (tens of hours per simulation point), the modeled systems are scaled down to a quarter of a real one. A full-scale AMD Zen4C chiplet contains 16 cores, as many considered in our cost analysis of Section 2.2.4[1]. As a consequence, our performance analysis considers chiplets scaled to be a quarter of a chiplet AMD Zen4C or Intel Sapphire Rapids chiplet and as such they contain a quarter of the number of cores and connect to a quarter of HBM channels, as shown in Table 2.1. Moreover, the L2 and L3 caches are undersized in order to put more pressure on the memory system and increase LLC misses per kilo instructions (MPKI), which is otherwise difficult to achieve when simulating systems for only a few billion instructions.

Based on the scaled down chiplet size ($16.5\,\mathrm{mm}^2$)[2], the microbump budget is calculated to be proportional to the number of cores it includes. In addition, the following parameters were taken into account for calculating the microbump budget: (i) a microbump pitch of $45\,\mu\mathrm{m}$, (ii) reserving 40% of the microbumps for power. Then, the number of microbumps available for data were allocated for (i) connecting to the HBM channels, (ii) one bidirectional link to the IO chiplet, (iii) multiple bidirectional links to the other CPU chiplets. Then, the width of the links to IO and CPU chiplets, as well as the total number of links to other CPU chiplets, were adjusted to fit the microbump budget. Finally, the latency of the inter-chiplet links was measured to be 2 or 3 (NoC) clock cycles considering the chiplet's dimensions and the latency of the links on the silicon interposer similar to [3, 39].

[1] A yield and cost analysis of a scaled down chiplet would not make sense as the size of the chiplets would be small, and so would be the size of the monolithic chip making it too small to break down into chiplets.

[2] Calculated based on Zen4 after scaling down L2 and L3 sizes proportional to the capacity indicated in Table 2.1.

Table 2.1: System Configuration[1]

System	
Chiplets	4 chiplets[1]
Cores and Caches	
Cores	4 cores[1] / chiplet, out-of-order, 3.2 GHz
TLB	I-TLB: 512-entry, 4-way, 1 cycle latency D-TLB: 512-entry, 4-way, 1 cycle latency
L1 Cache	L1-I: Private, 32KB, 4-way, 2 cycle access latency L1-D: Private, 32KB, 4-way, 2 cycle access latency
L2 Cache	Private, 256KB, 8-way, 4 cycle access latency
L3 Cache	Shared, 1MB/core, 16-way, 12 cycle access latency[2]
Main Memory	
HBM2	1 GB/chiplet, 2 GHz, 4 channels, 128 bits per channel, tCAS-tRCD-tRP: 14-14-14 ns
DDR4	4 GB, 3.2 GHz, 1 channel, 64 bits per channel, tCAS-tRCD-tRP: 22-22-22 ns
Network	
Intra-chiplet	2 GHz, 3-stage router (VA/SA, ST, LT), 2x3 Mesh, 4 VCs per port, credit-based flow control, 256 bit link for data, 154 bit link for control (coherence) traffic, 5 flit buffers, XY Routing [40]
Inter-chiplet	2 GHz, 3-stage router (VA/SA, ST, LT), 2x2 Mesh, passive interposer, 2 to 3 cycle link latency[3], 7 to 9 flit buffers [3]

[1] This configuration is the default setting. The parameter adjustments are detailed in the respective evaluation sections of the sensitivity studies.
[2] L3 access latency is 8 cycles for 2MB, 12 for 4MB, and 15 for 8MB.
[3] Depending on the maximum inter-chiplet link length [3].

2.3.2 Simulation Setup

BZSim simulator was used for our experiments [41], extended to model memory system and interconnects of chiplet-based chips. BZSim is based on ZSim simulator [42] integrated with BookSim2 [43] for cycle-accurate intra- and inter-chiplet network modeling, enhanced with a technique to detect and skip simulation of low contention traffic in order to speed up simulation times. BZSim offers microarchitectural simulations with detailed (cycle-accurate) interconnect modeling at an order of magnitude faster simulation speeds compared to GEM5, enabling multi-billion instruction experiments within reasonable times [41]. DRAMSim3 [44] was used for cycle-accurate DRAM modeling and CACTI [45] for estimating cache access times.

The system treats all HBM and external DDR memory as part of a unified flat address space. The virtual memory system was implemented based on HSCC [46]. The cores are configurable with translation lookaside buffers (TLBs) for both instructions and data, as well as with page table walkers (PTWs). Additionally, the memory management modules include a distance-aware allocation policy. This policy allocates pages to the HBM in the chiplet where they are first accessed. If pages are unavailable in the nearest HBM, they are allocated in the next neighboring HBM or in the external DDR.

Table 2.2: Workload Characteristics

Benchmark	Label	Input	LLC MPKI	Footprint (GB)	Assigned to Mixes mix-id$^{\#of instances}$
LLC MPKI 20-40					
pageRank[2]	PRL2	LDBC (100k)	37.41	0.84	1^3,2^3,3^2,4^3,5^2,6^3,7^3,8^1
mcf[1]	MCF	Default	34.01	0.45	2^2,6^3,8^2
graphColoring[2]	GCL2	LDBC (100k)	30.70	0.45	1^1,2^1,4^1,5^2,7^1,8^2
graphColoring[2]	GCL3	LDBC (10k)	21.26	0.09	1^2,2^1,3^2,6^2,8^1
Random Access Workload[3]	RAND	N=30, M=1000, chunk=1024	20.83	0.70	1^1,2^3,3^1,4^2,6^2,7^1,8^1
LLC MPKI 10-20					
connectedComp[2]	CCL3	LDBC (10k)	19.33	0.09	1^3,2^2,3^1,4^3,5^1,6^2,7^2,8^1
lbm[1]	LBM	Default	18.19	0.40	3^1,7^2,8^1
BFS[2]	BFSCR	CA RoadNet	17.25	0.64	1^1,2^1,3^1,4^2,5^3,7^2,8^1
fotonik3d[1]	FOTO	Default	17.07	0.59	1^1,4^1
pageRank[2]	PRL3	LDBC (10k)	13.96	0.09	2^1,4^1,5^1
xalancbmk[1]	XAL	Default	13.62	0.16	1^1,2^1,3^2,4^1,6^2,7^3,8^1
blender[1]	BLEN	Default	12.78	0.08	2^1,4^1,5^1
shortestPath[2]	SPCR	CA RoadNet	12.30	0.64	1^1,3^2,5^1,7^1,8^1
XSBench[4]	XSB	XXL	11.11	0.37	5^1,8^1
graphColoring[2]	GCCR	CA RoadNet	10.69	0.63	1^1,3^1,5^2,6^1,8^1
LLC MPKI 0-10					
parest[1]	PAR	Default	8.54	0.05	3^1
roms[1]	ROMS	Default	7.58	0.25	8^1
triangleCount[2]	TCL2	LDBC (100k)	6.24	0.55	6^1,8^1
graphColoring[2]	GCL1	LDBC (1000k)	5.92	0.29	1^1,4^1,7^1
pageRank[2]	PRKR	Knowledge Repo	4.56	0.30	3^1,5^1
omnetpp[1]	OMN	Default	4.53	0.16	5^1
BFS[2]	BFSL1	LDBC (1000k)	2.71	0.98	3^1

[1] SPEC CPU 2017 [47], [2] GraphBIG [48], [3] GUPS [49], [4] XSBench [50]

2.3.3 Workloads

We use mixes of multi-programmed workloads from the SPEC CPU2017 benchmark suite [47] (the eight with the highest MPKI), GraphBIG [48], Random access workload from the GUPS suite [49] and XSBench [50] in our experiments. For the SPEC CPU2017 and GraphBIG benchmarks, we use Simpoints [51] to select a representative slice of one billion instructions. We have chosen 22 different workloads, detailed in Table 2.2, and created random multi-programmed mixes of 16 applications designed to run on a system with 16 cores. Each mix of applications has a minimum total memory footprint of 7 GB and a geometric mean LLC MPKI of at least 11. To scale these mixes for systems with 32 or 64 cores, we replicate the 16-application mix twice for the 32-core system and four times for the 64-core system. All experiments run with an average of 125 million instructions per core warm-up period, where memory allocation is enabled, followed by an average of 250 million instructions per core of detailed simulation.

2.3.4 Evaluated Systems

We evaluate three distinct systems as follows:

1. **Chiplet-based System (CS)**: A multi-chiplet system with sliced LLC is the focus of our evaluation. The default configuration (depicted in Table 2.1) consists of 4 chiplets, each with 4 cores, integrated on a passive interposer with one LLC slice per chiplet, 256 bit NoC data-links, 4 HBM channels per chiplet, one link to IO chiplet, and one channel to external DDR. The above parameters change in the various sensitivity analyses of the evaluation. One variation of this design is to use chiplets with private, rather than sliced, LLC, denoted as **CP**.

2. **Monolithic (MN)**: A monolithic multicore matching the CS characteristics. Similarly, the default monolithic configuration is a 16-core system with sliced LLC in 4 parts and 256 bit NoC data-links, 16 HBM channels and one external DDR channel.

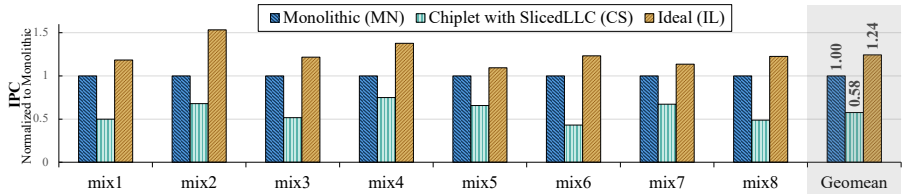
3. **Ideal (IL)**: An ideal chiplet-based system with the ideal scenario where an LLC miss is always served by the closest HBM channel, assuming the local HBM has infinite capacity, so memory allocation occurs solely within this local HBM. As a result, all memory requests remain local to the chiplet, eliminating the additional latency associated with accessing remote HBM or external DDR.

2.4 Performance Evaluation

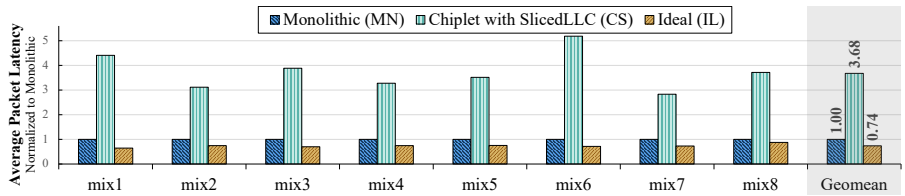
The performance of chiplet-based systems is evaluated and their overheads with respect to monolithic counterparts are measured. System performance is measured in terms of Instructions Per Cycle (IPC). The Average Memory Access Time (AMAT) is also measured and broken down to: (i) the access time for each cache level, (ii) the Network-on-Chip (NoC) latency between each level (L2-L3 and L3-DRAM), and (iii) the DRAM access time. Furthermore, the Average Packet Latency (APL) and the percentage of accesses to local and remote HBM nodes, as well as to external DDR are reported.

The performance evaluation is structured as follows: first, the default chiplet-based system is compared against the monolithic and ideal systems. Subsequently, a sensitivity analysis of the system size is conducted to examine how performance scales as the number of chiplets increases. Next, the impact of chiplet granularity is explored by analyzing different chiplet sizes (i.e., cores per chiplet) while keeping the system size fixed. Then, the performance of an alternative LLC organization for chiplet-based designs (private LLC per chiplet) is evaluated in comparison with the default sliced LLC. Next, a sensitivity analysis of the intra-chiplet NoC data bandwidth (datapath) is performed. Finally, the impact of passive versus minimally active interposer is measured.

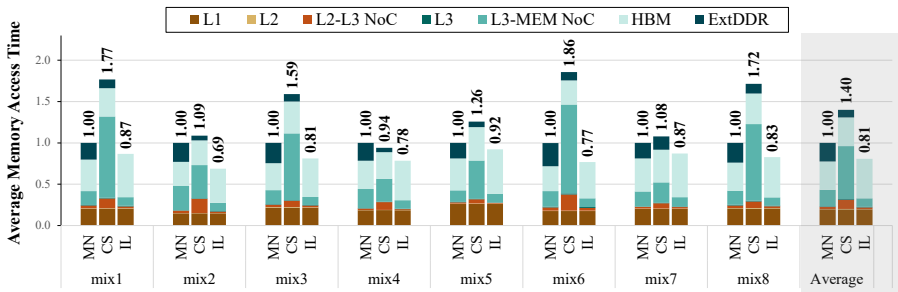
Multi-Chiplet vs. Monolithic vs. Ideal: Figure 2.4 shows, per mix of programs, the average performance (IPC), average packet latency, average memory access time, and the breakdown of DRAM accesses for the default configuration of a 4-chiplet system as well as for the equivalent 16-core monolithic and ideal systems. The chiplet-based system is able to maintain only



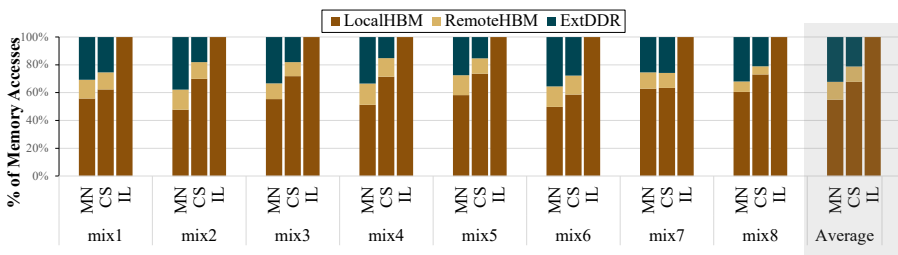
(a) IPC (Normalized to Monolithic)



(b) Average Packet Latency (Normalized to Monolithic)



(c) Average Memory Access Time (AMAT) (Normalized to Monolithic)



(d) Memory Accesses

Figure 2.4: Multi-Chiplet vs. Monolithic vs. Ideal

43%-75% of the monolithic performance and on average 58%, as shown in Figure 2.4(a). Compared to the ideal system, which is 24% faster than the monolithic, and its LLC misses always go to the closest HBM, the chiplet-based system is 54% slower. The performance overhead of the chiplet-based system versus the monolithic is not explained by just observing AMAT, which is on average 40% higher than the monolithic. A more detailed look in Figure 2.4(c) reveals that the longer monolithic AMAT is due to slow external DDR accesses, rather than longer NoC and cache access latency, which is more performance critical and hence puts a heavier toll on chiplet-based performance. In fact, Figure 2.4(b) confirms the 3.7× longer packet latency of chiplet-based systems compared to monolithic. Finally, it is interesting to analyze the primary source

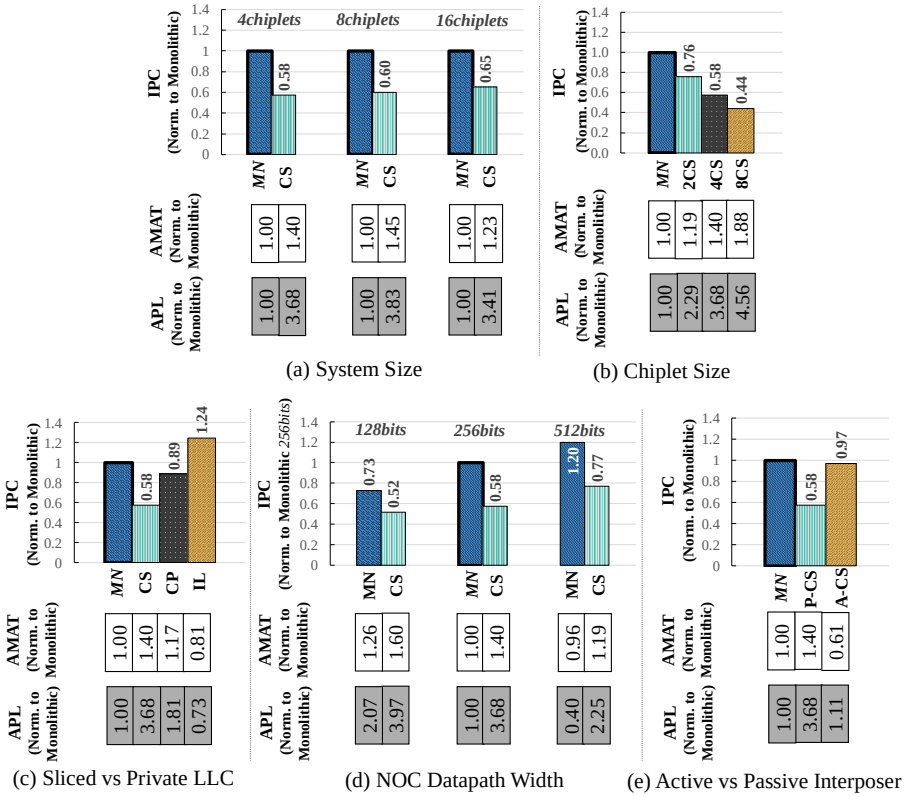


Figure 2.5: Sensitivity analyses* on system size, chiplet size, LLC organization, NoC datapath width, and interposer type.

* Geomean values of 8 mixes are shown, normalized to **MN**, as indicated by the thicker outline of the bars; **AMAT**: Average Memory Access Time; **APL**: Average Packet Latency; **MN**: Monolothic, **CS** or **CP**: Chiplet-based with Sliced or Private LLC, **IL**: Ideal, **P-CS** or **A-CS**: Chiplet-based with Passive or Active Interposer.

of the performance overheads in the chiplet-based system, which is a fraction of accesses to data placed remotely. Figure 2.4(d) shows that on average 19% of the data accesses are to the external DDR and 10% of the accesses are to a remote HBM node. For a chiplet-based system, as opposed to a monolithic one, all these accesses involve slow (due to limited bandwidth and longer latency) inter-chiplet communication.

Sensitivity Analysis on System Size: An interesting sensitivity analysis is with respect to the system size. Chiplet-based chips are meant to scale better to larger systems in terms of cost. However, it is unclear how their performance overheads change when system size increases while keeping the chiplet size constant. The performance of systems with 4, 8, and 16 CPU chiplets, i.e., 16, 32, and 64 cores, respectively, are evaluated. As shown in Figure 2.5(a), as the system size increases, the performance gap between chiplet-based systems and their respective monolithic of the same size remains stable or even slightly reduces, ranging from 58% to 65%. This is attributed primarily to the distance-aware data placement, which allows the distance of remote accesses to remain relatively stable.

Sensitivity Analysis on Chiplet Size: Next, a sensitivity analysis on the chiplet size, i.e., the number of cores included in a chiplet, is performed for chiplets of 2 (2CS), 4 (4CS), and 8 cores (8CS) per chiplet on a 16-core system. Figure 2.5(b) shows the performance of these systems. As expected, performance reduces as the system is disintegrated to a larger number of chiplets. More precisely, 2, 4, and 8 chiplet systems offer 76%, 58%, and 44% of the monolithic IPC, respectively, which is reflected in their AMAT and APL measurements.

LLC Configuration Analysis - Sliced vs. Private: A chiplet-based system with sliced LLC would need to go to a remote chiplet to serve an L2 miss accessing a remote LLC slice if the memory address is mapped to the memory region of another chiplet. An interesting question arises as of the benefit of designing chiplets each with a private LLC. Such private LLC would store cache lines from the entire address space, regardless of whether the memory is mapped to local or remote DRAM (HBM and external DDR). Figure 2.5(c) depicts the results of this comparison. Chiplets with sliced LLC reduce IPC versus monolithic by 42%, while chiplets with private LLC reduce it by only 11%. A significant reduction in average packet latency of about 50% is achieved by using private LLC because, with this organization, L2 misses do not need to go off-chiplet, as opposed to LLC misses, which are fewer, and may need remote accesses. On the contrary, chiplets with sliced LLC may need remote accesses to serve L2 misses but not for LLC misses. This is also reflected in the AMAT, which is improved by 16% when using chiplets with private LLC compared to chiplets with sliced LLC.

Sensitivity Analysis on NoC Datapath Width: The performance impact of the NoC data-link bandwidth is analyzed for chiplet-based and monolithic systems. The intra-chiplet and monolithic NoC datapath varies from a quarter of a cache line (128 bits), to a full cache line (512 bits). It is worth noting that the width of inter-chiplet links remains constant as defined by the available microbump budget of the chiplets (64 bits). Figure 2.5(d) shows the IPC, AMAT and average packet latency of the three design points for chiplet-based and monolithic chips normalized to the 256-bit monolithic. It can be observed that wider (intra-chiplet) NoC links improve performance, although the gap with the respective monolithic does not follow a specific trend. Finally, as expected, AMAT improves for wider NoC datapaths, while at the same time, the gap between monolithic and chiplet-based average packet latency increases because the bottleneck of narrower inter-chiplet links is exacerbated.

Sensitivity Analysis on Active vs. Passive Interposer: The last design parameter explored in our study is the use of minimally active (A-CS) versus passive interposer (P-CS). As illustrated in Figure 2.5(e), a minimally active interposer increases both throughput and latency of inter-chiplet links because it can pipeline them. The performance impact of this is significant as it recovers most of the performance overhead on chiplet-based systems. That comes, however, at a higher system cost due to the lower yield of active interposers.

2.5 Conclusions

Semiconductor technology has difficulty scaling the integrated resources on a single die because monolithic chips have poor yield, leading to excessive costs. Multi-chiplet chips offer a cheaper alternative as they have a higher yield, but come with certain performance overheads stemming from their NUMA memory system and inter-chiplet interconnection bottlenecks. This paper analyzed these performance overheads. In particular, our study reveals that although chiplet-based chips reduce system costs by almost half compared to monolithic, they give away about a third of the monolithic performance. Our work further showed that part of this performance overhead can be regained with specific design choices. More specifically, designing chiplets with a private LLC improves performance by about 30%. Moreover, chiplet-based systems with active interposers are only 3% shy of the monolithic performance.

Paper B

MEMPLEX: A Multi-Chiplet NUMA Architecture with Data Replication and Migration

Neethu Bal Mallya, Bhavishya Goel, and Ioannis Sourdis

Under review.

Chapter 3

MEMPLEX: A Memory System with Replication and Migration of Data for Multi-Chiplet NUMA Architectures

Abstract

As the semiconductor industry struggles with the diminishing returns of Moore's law and explores innovative solutions for integrating more resources on a chip, multi-chiplet chips offer a cost-efficient alternative to large monolithic chips due to their higher yield. However, chiplet-based systems inherently exhibit Non-Uniform Memory Access (NUMA) characteristics and, therefore, suffer from slow remote accesses. Although data placement in multi-chiplet NUMA systems can be optimized in software, currently, there are no hardware mechanisms to dynamically improve data placement in DRAM distributed across chiplet nodes. Our experiments show that this leads to wasting a significant fraction of system performance compared to a hypothetical system with ideal data placement. Our work addresses this problem by introducing MEMPLEX, a novel memory system for multi-chiplet NUMA architectures, which offers data replication and migration in the memory nodes of a multi-chiplet system. MEMPLEX allocates a small fraction of each memory node to construct a DRAM cache and offers their remaining capacity to a shared flat address space with hardware migration. In a nutshell, MEMPLEX DRAM cache attracts data of the working set to the local memory node and decides whether to migrate them upon eviction based on their usage in the cache. Thereby, MEMPLEX improves data locality, regains a large fraction of the above performance overhead, and offers substantial energy savings.

3.1 Introduction

Fitting more resources onto a chip has always been a key aspect of enhancing the performance of processor chips. This became more critical in the multicore era after Dennard scaling could no longer deliver higher frequencies due to power limitations. However, with Moore's law running out of steam, technology scaling is increasingly challenging, and integrating more resources on a single monolithic chip has become too expensive. Building larger chips out of multiple smaller chiplets offers higher yield and is thus a lower cost alternative [2, 4].

Multi-chip integration technologies were initially employed to build High Bandwidth Memory (HBM) and position it closer to a processing die, such as a GPU [5] or a vector engine [6]. It was soon expanded to disintegrate processors to multiple chiplets, as seen in AMD's EPYC and RYZEN architectures, providing access to multiple memory nodes with non-uniform access latencies that vary by tens of nanoseconds [1]. Currently, multi-chiplet chips, such as AMD MI300 [7] and Intel Sapphire Rapids [8], integrate multiple CPU and/or GPU chiplets along with HBM nodes, forming part of a complex and less uniform memory system.

Non-uniform Memory Access (NUMA) machines entail the performance pitfall of long latency remote accesses, but also offer opportunities for performance optimizations, if data locality is maximized. In the 1990s, Cache Only Memory Architectures (COMA) [13] and Cache Coherent NUMA (CC-NUMA) [14] memory systems improved data locality and, consequently, the performance of NUMA multi-socket machines by replicating (caching) and/or migrating data close to the processor chip. More recently, hybrid memory systems composed of nodes with heterogeneous characteristics, such as smaller HBM and larger but lower bandwidth external DRAM, have employed similar techniques to reduce memory access times, including, DRAM caching [22–31], data migration [15–21], or a combination of both [32].

The focus of this work is on NUMA architectures composed of multiple CPU chiplets and HBM nodes, such as the AMD MI300 [7] or Intel Sapphire Rapids [8]. Currently, such systems use HBM as part of a flat address space and rely on Operating System (OS) support or user optimizations to place data closer to the processing chiplet, or they use the entire HBM as a DRAM cache of an external DDR memory, thereby wasting valuable main memory capacity [8]. This paper demonstrates that even when memory allocation is NUMA-aware, placing data in the closest available memory node relative to the processing node, system performance is still significantly reduced compared to an ideal system that always finds data in its local memory node. Our aim is to alleviate this performance loss by reducing remote memory accesses.

To this end, we propose MEMPLEX, a novel memory system that offers replication and migration of data across the memory nodes of a multi-chiplet chip in order to enhance data locality. As a result, the number of accesses to the closest memory node for each processing chiplet is increased, while accesses to the remote memory nodes are minimized. This reduces the average memory access time, thereby improving system performance. The proposed memory system uses a small fraction of each HBM node as a DRAM cache and decides whether to migrate data upon eviction from that cache based on the usage of the evicted blocks.

Concisely, this paper makes the following contributions:

- Investigates the performance bottlenecks in a multi-chiplet NUMA system revealing a performance loss of 26% and 31% in 4- and 16-chiplet configurations, respectively, compared to an ideal system.
- Introduces MEMPLEX, the first multi-chiplet NUMA architecture, which combines replication and migration of data across multiple memory nodes. As a result, it:
  - Offers most of the capacity of the HBM nodes as shared flat address space, as opposed to designs that use them entirely as DRAM cache.
  - Outperforms existing software solutions that offer NUMA-aware data placement on a flat address space, as well as designs that use HBM exclusively as DRAM cache.
- Evaluates MEMPLEX on multi-programmed mixes of workloads from different benchmark suites (detailed in Section 3.4.3), and shows that, compared to a multi-chiplet system with NUMA-aware data placement and no support for DRAM caching or migration, MEMPLEX:
  - Eliminates 80% of the remote memory traffic, resulting in a 44% reduction in dynamic memory energy consumption in a 4-chiplet system.
  - Achieves up to 7% speedup (5% on average) when $\frac{1}{16}$ of each HBM is dedicated for caching in a 4-chiplet system, with performance gains increasing up to 15% (10% on average) in 16-chiplet systems.

The remainder of this paper is organized as follows: Section 3.2 discusses related work, Section 3.3 presents the MEMPLEX architecture, Section 3.4 outlines our experimental setup, Section 3.5 presents our evaluation results, and Section 3.6 concludes with a summary of our findings.

3.2 Related Work

This section reviews existing solutions for data replication and/or migration in memory systems, with a focus on (i) traditional multi-socket NUMA shared memory systems, (ii) hybrid memory systems composed of HBM and external DRAM, and (iii) the software support available for NUMA architectures.

3.2.1 Non-Uniform Memory Access in Shared Memory Systems

Distributed shared memory (DSM) systems inherently deal with the problem of incurring higher delays when retrieving data mapped to a remote memory node compared to its local memory, resulting in non-uniform memory access. Optimizing latency for remote data access in DSMs has been an extensive topic of research in computer architecture for many decades. Cache-Coherent NUMA (ccNUMA) machines address this challenge by allowing the remote data to be cached in the local node's cache hierarchy [14, 52–54]. References to the remote data that miss the local node's cache hierarchy are sent to the *home node* of the referenced page. The home node is responsible for the initial allocation of

the page and is in charge of maintaining the consistency and coherence of that page across the system.

Due to the relatively smaller size of the remote cache, the ccNUMA systems exhibit high sensitivity to data placement. This sensitivity can be mitigated to some extent by strategies such as caching remote data on DRAM [55, 56] or leveraging OS support for dynamic page migration to local memory [57]. Cache-only memory architecture (COMA) machines address this problem by allowing the remote pages to be freely migrated to the local memory, improving the chances of the referenced data being available locally. Since there is no concept of a home node in traditional COMA systems, block localization in case of a miss in the local memory can be challenging and time-consuming. FLAT-COMA [58] resolves this problem by assigning a fixed home node for each page. In this scheme, the pages are free to migrate to remote nodes, but the location of the directory remains fixed. COMA systems implement block replacement and relocation mechanisms in hardware, resulting in increased hardware complexity. Simple-COMA (S-COMA) [59] systems simplify the hardware implementation by offloading some of this complexity to the operating system. When a remote page is first referenced, it results in a page fault. The operating system allocates a page frame in the local memory for the remote page and fetches the remote block into this newly allocated page frame. Subsequent references to the same block get mapped directly to the local memory. Since the physical addresses in the local memory are handled independently by the local MMU, identical blocks residing in different nodes can have different physical addresses. Consequently, nodes need a global identifier for migrated pages for inter-node communication. Hence, each node maintains a translation table responsible for converting local addresses to global addresses and vice versa.

Reactive NUMA (R-NUMA) [60] aims to combine the performance benefits of ccNUMA and S-COMA. This scheme initially allocates the remote block in the remote cache to achieve a low initial overhead cost of ccNUMA. The system keeps track of block refetching due to conflict and capacity misses to remote cache and initiates S-COMA page allocation process when refetch count exceeds a certain threshold.

3.2.2 Hybrid Memory Systems

DRAM-based hybrid memory systems combine two types of memory to balance performance and capacity. The first type is High Bandwidth Memory (HBM), which offers high data transfer rates but has limited capacity due to heat dissipation issues, increased cost, and stacking efficiency. To complement the HBM, conventional lower-bandwidth external DRAM is used to expand the system's overall memory capacity. Ideally, the goal is to design a memory system that seamlessly integrates the high bandwidth of HBM with the larger capacity of off-chip external DRAM, providing an efficient balance of both performance and storage.

Currently, there are two primary approaches for organizing hybrid memory systems. The first approach is to use HBM and off-chip DRAM as part of the same hybrid main memory system, with a migration mechanism that brings the "hottest" data to the high-bandwidth 3D-stacked DRAM [15–21]. Some of these designs rely on OS to select and migrate data [15] and, although simpler,

have a slow response to working set changes. Other data migration solutions are implemented in hardware, offering a faster response, but need to handle address remapping and keep it transparent to the OS [16–21]. The second approach uses HBM as a DRAM cache of the external DRAM [22–31], with the primary challenge being the overheads associated with the management of metadata (tags).

A hybrid of these two approaches has also been proposed in the Hybrid^2 design, which reserves a fraction of the HBM for caching and offers its remaining capacity to the main memory [32]. Hybrid^2 targets systems with a single processor chip and 2-level hybrid memory, i.e. HBM and external DDR. In contrast, MEMPLEX extends the concept of combining replication and migration to chiplet-based systems featuring multiple processing chiplets, multiple HBM nodes, and external DDR. Unlike Hybrid^2, MEMPLEX faces more complex challenges, such as (i) the remap information needs to be fragmented and scattered to the various NUMA nodes, (ii) data allocation and migration decisions are intricate due to the varying distances between memory nodes, and (iii) introducing multiple DRAM caches (one per HBM node) calls for compatibility with directory-based cache coherence protocols [55]. MEMPLEX addresses these challenges and provides an innovative solution that offers data replication and migration in the memory system of a multi-chiplet processor, improving the performance and energy efficiency of chiplet-based systems and demonstrating scalability to larger systems with more chiplets.

Existing commercial multi-chiplet processors with HBM, such as Intel Sapphire Rapids [8], do not combine replication and migration. Their HBM nodes are either used entirely as DRAM cache wasting capacity, or as part of a flat address space with no hardware support for migration. In the latter case, software techniques can be employed to alleviate NUMA overheads, as explained next.

3.2.3 Software support in NUMA machines

Modern operating systems widely support NUMA architectures through various mechanisms. Operating systems like Linux [35], Windows [36], and FreeBSD [37] implement NUMA-aware scheduling algorithms to place processes and threads closer to the memory nodes, minimizing access latency. Additionally, operating systems allow user applications to discover NUMA topology, request memory from specific nodes, and set process affinity through NUMA APIs. In addition, Linux also facilitates manual page migration [61] from the remote NUMA node to the one currently running the process. The automatic NUMA balancing mechanism in Linux [62] enables periodic unmapping of process memory, NUMA hinting faults, migration-on-fault, and automatic placement of tasks closer to the memory. These operating system features can make applications NUMA-aware and improve performance on NUMA machines mostly by selecting a static mapping that places data close to the consuming threads. However, even with dynamic migration support, this approach often requires programmer intervention and has a slower response to working set changes compared to hardware migration solutions [32]. In contrast, MEMPLEX aims to transparently improve performance on NUMA systems without putting an additional burden on application programmers.

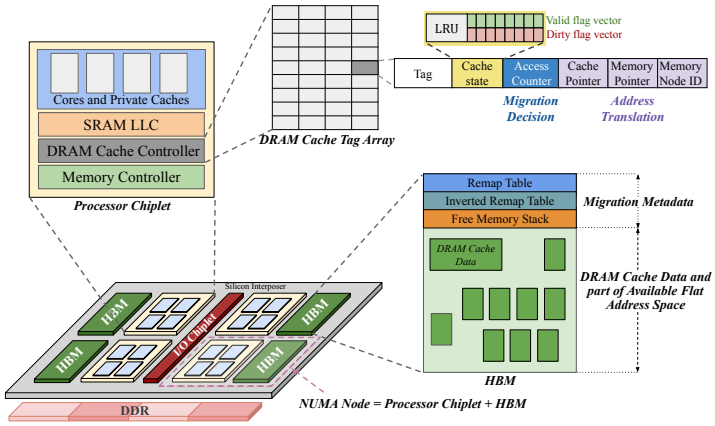


Figure 3.1: MEMPLEX System Overview

3.3 MEMPLEX Design

MEMPLEX is a memory system for chiplet-based architectures, comprising multiple processor chiplets and HBMs integrated on a silicon interposer as well as external DDR memory accessed via an IO chiplet, as illustrated in Figure 3.1. Without loss of generality, the system is organized in NUMA nodes composed of a processor chiplet and an HBM. Each processor chiplet has a high bandwidth connection to its nearby, local HBM, henceforth denoted as Local Memory (LM), and can gradually reach larger parts of the shared memory at the cost of lower bandwidth, by connecting first to the remote HBMs of other nodes on the chip and second to the off-chip external DDR, collectively referred to as Remote Memory (RM). On a system with such trade-offs between memory bandwidth and memory capacity, MEMPLEX improves data locality by employing a DRAM cache and a migration scheme on the shared flat address space. It allocates a small portion of each HBM in the system as the data array of a sectored DRAM cache, private to the node, and utilizes its remaining capacity to form, combined with the external DDR, a shared flat address space offering hardware support for data migration across the shared HBMs and the external DDR.

3.3.1 MEMPLEX System Overview

The MEMPLEX system combines data replication and migration across the HBMs of a multi-chiplet chip as well as the external DDR. A fraction of each HBM is allocated to store the data array of a sectored DRAM cache, which attracts data frequently used by the cores on the local chiplet. The rest of its capacity is part of the flat address space. The DRAM Cache Tag Array (DCTA) is maintained in SRAM locally at a reasonable cost. Migration decisions are made per sector upon its eviction from the DRAM cache.

In MEMPLEX, the data management operates at distinct granularities. The data blocks in the DRAM cache are fetched at the cache line granularity (64 Bytes). The DRAM cache tags are maintained at the sector granularity,

which for simplicity is equal to an OS page (4 KBytes). After a cache miss in the SRAM LLC, DCTA is the first point of reference for determining whether the requested cache line is available within the DRAM cache. The requested cache line may reside either in the LM or in RM. In the event of a tag array miss, a new entry for the missing sector is allocated in the DCTA regardless of where the requested cache line resides. However, a new data array entry for the sector is allocated in LM only if the requested cache line resides in the RM. Otherwise, if the sector is already located in the main memory part of LM, the added DCTA entry would point to the existing location of the sector in LM and mark all cache lines as dirty to ensure a writeback after eviction. Thereby, replication of data that already reside in LM is avoided, while DCTA acts as a cache of the address remap information.

An HBM is logically, rather than physically, partitioned between the DRAM cache and the flat address space, and the partitioning is facilitated by pointers maintained in the DCTA. This allows for a seamless link of sectors already present in the LM to the DRAM cache tags. Moreover, it enables cached sectors from RM to be migrated into LM without relocating the already fetched cache lines.

The sectored DRAM cache allows the tags to be kept entirely on the processor chiplet without significant SRAM cost due to its small size. This induces minimal latency to the critical memory access path as all the memory requests go through the DCTA. The tag array also contains additional information to facilitate the data migration within the shared memory. Besides the tag and cache state, each entry in the on-chip tag array stores the remapped address of the sector, serving as a cache of the migration metadata, which effectively reduces the overhead of address remapping. Section 3.3.2 elaborates on the DCTA structure.

When a sector is evicted from the DRAM cache, the migration mechanism decides whether to migrate it to the LM or evict it back to its current location in RM. The migration decision is based on the cost of migration in terms of the memory traffic and the number of accesses to the sector while in the DRAM cache. By deferring the migration decision until a DRAM cache eviction, the management of migration-related metadata is moved off the critical path, thereby minimizing its impact on performance. Additionally, the RM traffic generated by migrations is dynamically adjusted according to the workload behavior.

3.3.2 DRAM Cache Controller

Each processor chiplet in the system features a DRAM Cache Controller (DCC) responsible for high-level block management tasks. This includes handling requests from the processor, accessing the on-chip tags, fetching cache blocks on misses, evicting blocks, and generating writeback traffic to the main memory for dirty blocks. Additionally, the DCC manages sector migrations between the local and remote memory. This involves translating the addresses of remapped sectors, selecting sectors for migration to LM, and making migration decisions based on data usage (while in DC) and migration overhead considerations. Section 3.3.7 discusses how DCC manages migrations. All memory requests go through the DCC, which communicates with the memory controller to access the HBMs and external DDR.

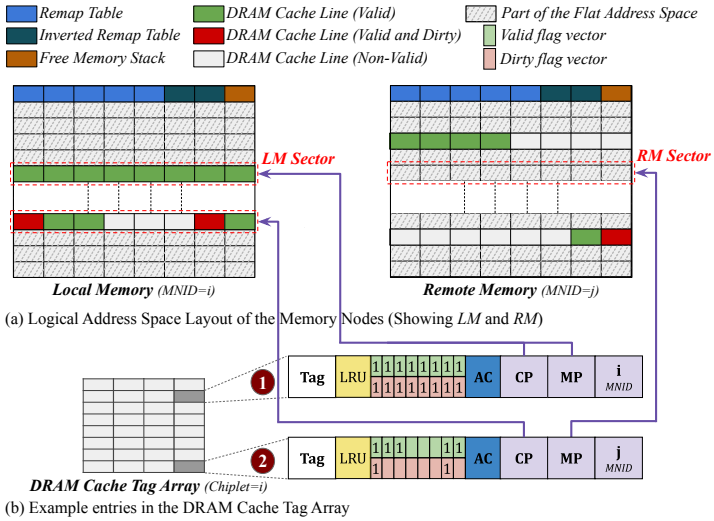


Figure 3.2: (a) Logical Address Space Layout of the Memory System and (b) Example entries in the DCTA

The DCC manages the DRAM Cache Tag Array (DCTA), which stores all tags for the DRAM cache in SRAM on the processor chiplet. DCTA is set-associative, with each set containing entries for multiple sectors. Each entry in DCTA, as depicted in Figure 3.1, comprises the sector tag, state bits for each cache line in the sector (including valid and dirty bits), an access counter, two pointers, and a node identifier. The Access Counter (AC) monitors sector accesses and is used upon DRAM cache eviction to decide whether to migrate the sector to LM or evict it to RM. AC is incremented only for non-migrated LM sectors to prevent potential starvation within the cache set, ensuring LM sectors with frequent accesses are not evicted from the DCTA. Additionally, we ignore the sectors whose counters have reached the maximum value to prevent starvation from RM sectors that remain in the cache for prolonged periods. Pointers facilitate address translation of processor physical addresses to sector locations in the memory system. The Cache Pointer (CP) decouples the set and way from the physical location of data in the LM. This indirection allows our design for sector migration to LM without the need to copy data from one LM location to another. The Memory Pointer (MP) points to sector physical locations in RM and helps avoid remap table lookups. MP is same as CP for sectors that belong to the LM or for sectors that have entirely migrated to the LM. The Memory Node Identifier (MNID) is used to identify the node where the cached sector is located in the main memory. For the HBM address space, the MNID is the node ID where the HBM is located. On the other hand, the external DDR is divided into a number of regions equal to the number of nodes in the system, so each region is assigned to a different node, as illustrated in Figure 3.1. When the MNID matches the self-ID of a node, it indicates that the respective node has the sector in its flat address space, either as a result of migration or as the original *Home Node (HN)* location determined by the memory allocation.

3.3.3 Memory Layout & Metadata

Figure 3.2(a) illustrates the logical address space layout of the memory system, depicting the HBM local to a processor chiplet (LM) and another one that is remote (RM). The external DDR memory is not illustrated as it only contains data as part of the address space. On the contrary, each HBM includes a reserved portion containing migration metadata structures used in the MEMPLEX design. The non-reserved portion of the HBM is logically partitioned between DRAM cache data and the available flat address space across the shared memory. This means that the sector corresponding to a DCTA entry can be located anywhere across the flat address space of its respective LM (shown by the lined area in Figure 3.2(a)). DCC uses the pointers maintained in the DCTA to track the location of sectors within its respective LM. Sectors in LM may either fully reside in its DRAM cache (with a corresponding DCTA entry) or not at all. Sectors in RM may be partially or fully cached in the DRAM cache, also with corresponding DCTA entries.

Figure 3.2(b) demonstrates examples of DCTA entries. The first entry (1 in Figure 3.2(b)) corresponds to a sector entirely migrated to its respective LM, as indicated by CP specifying its location. In this scenario, MP is the same as CP, MNID is the node's self ID, and as a convention, all valid and dirty bits are set. The second entry (2 in Figure 3.2(b)) represents a sector partially cached in the DRAM cache, indicating it has not been migrated to the LM. Some cache lines of the sector have been fetched to the LM, as indicated by the valid flag vector of the DCTA entry. The dirty flag vector specifies the cache lines of the sector that were written while in the DRAM cache. The CP and MP pointers indicate the sector location in the LM and RM corresponding to MNID, respectively.

Migration Metadata Structures

Our design allows all-to-all address remapping for pages across the flat address space available in the HBMs and external DDR. To achieve this, we maintain the following structures in each memory node:

- **Remap Table:** Each node in the system maintains a remap table which stores mappings from the processor physical address to the actual memory location of the sector in the memory system. A remap table stores entries of sectors (pages), which natively belong to its node and are migrated elsewhere as well as sectors migrated to its local HBM from other parts of the memory. Unlike the centralized remap table in the Hybrid^2 design, where all remap information is stored in one location, our approach distributes remap information across NUMA nodes. This fragmentation introduces greater complexity in tracking data. To mitigate this complexity, the remap table structure is optimized by implementing it as a hash table that maintains an entry for each of the native LM sectors that have migrated to RMs and for the RM sectors that have migrated to the LM. This means that if a native LM sector gets a miss in the remap table, the sector is in its default, native location. The structure is indexed by the processor physical address and points to the memory location of the sector. On a sector migration, the remap table is updated

to reflect the new address. It is worth noting that the DCTA serves as a cache for remap table entries of sectors currently (partially or fully) in the DRAM cache, facilitated by the pointers illustrated in Figure 3.2(b). This distributed yet optimized design enables effective memory location tracking across NUMA nodes.

- **Inverted Remap Table:** This table contains processor physical addresses corresponding to all locations within the respective memory node. The table also includes a bit map representing the sharers of the address if the sector is cached. This table is employed during the migration of blocks out of the memory node. Further details on its usage are provided in Section 3.3.5.

- **Free Memory Stack:** Each node maintains a stack of a minimum number of its own free locations, which currently hold no valid data and are available for use. A predefined number of entries from this stack are given exclusively to each other memory node for migrating data. Thus, in addition to its own free locations, each memory node maintains a stack of free locations reserved for use on all other memory nodes. Furthermore, when a node exhausts its available free locations, it requests additional entries to replenish its stack. The stack size is bound to the number of sectors that can fit within the DRAM cache. The stack pointer and a set number of top entries of the stack per node are stored on-chip within the DCC to minimize LM access.

MEMPLEX overheads are primarily related to (1) added logic in the memory controller (for supporting migration and DRAM caching), which is similar to the overheads imposed by existing hardware migration and DRAM caching approaches, (2) SRAM cost for storing DCTA, and (3) DRAM space for metadata. The space allocated for all the above metadata is small (even when considering the full remap table) and in our implementations constitutes only 0.5% of a memory node capacity.

3.3.4 Memory Access Path

When a memory request arrives in the DCC of a requesting node due to an LLC miss, the DCTA is indexed with the (physical) address to determine if the requested sector and the specific cache line is available within the DRAM cache. This operation can result in one of four possible outcomes, as illustrated in Figure 3.3.

❶ ***DCTA Miss:*** In this scenario, the DCTA does not contain an entry corresponding to the requested sector. The requested sector may reside either in the LM or in any of the RM locations. Regardless of where the requested sector resides (LM or RM), an entry is allocated in the DCTA for that sector. Section 3.3.6 elaborates on the allocation of a new entry in the DCTA and the eviction process in the DRAM cache if necessary during this allocation.

The address remap table in the LM is accessed using the sector's physical address to determine the sector's location in the memory system. If the remap table in LM does not contain the updated location of the requested

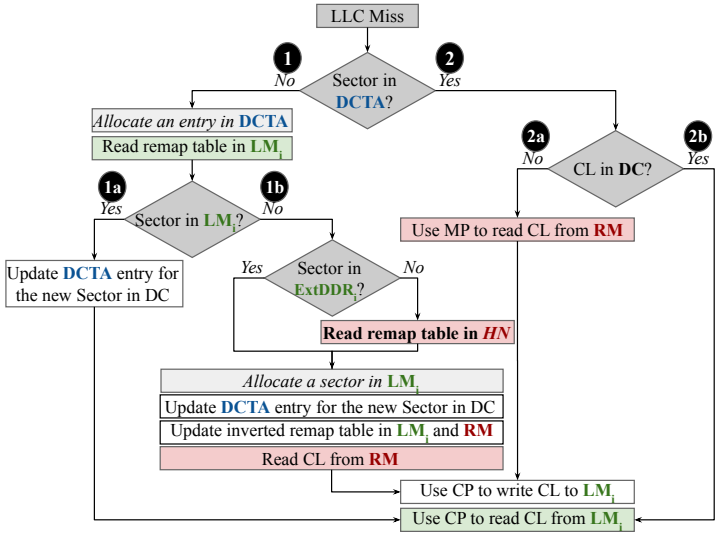


Figure 3.3: Memory Access Path

physical address, the system defaults to accessing the remap table in the *Home Node (HN)*, decoded from the higher-order bits of the address. This is typical for the first access to a sector allocated in any of the RMs. Subsequent requests from the same chiplet would be served from its DCTA.

1a ***Requested Sector in LM:*** If the sector is located in the LM, then all cache lines associated with that sector are already present in the LM. Consequently, the entry in the DCTA is updated accordingly. The CP and MP are set to point to the LM location of the sector. Additionally, the MNID is set to the self-ID, and all cache lines are flagged as valid and dirty.

1b ***Requested Sector in RM:*** If the sector is not located in the LM, but is in the region of external DDR that belongs to the requesting node, then the location of the sector is already known form step 1a. Otherwise, the remap table in the RM node, i.e., the node of the physical address is accessed to get the updated location of the requested physical address. Next, space is allocated in the LM for caching the new sector in the DC, and the requested cache line needs to be fetched from RM to the newly allocated location in LM. Section 3.3.5 elaborates on the allocation process followed by a memory node. Subsequently, the DCTA is updated with the new sector. The CP is set to point to the newly allocated LM location of the sector. The MP is set to the RM location of the sector, and the MNID is assigned the ID of the RM node. The valid flag is set only for the fetched cache line, while the dirty flag depends on the request type. Additionally, the inverted remap table in the requesting node is updated with the physical address of the sector, even though this sector has not yet been migrated to LM. This is done to ensure correctness during LM allocation, as explained in Section 3.3.5.

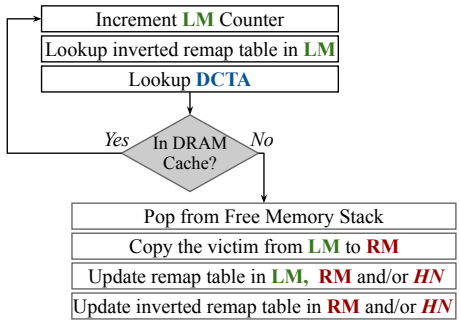


Figure 3.4: Allocating a sector in Local Memory

❷ ***DCTA Hit:*** In this scenario, the DCTA contains an entry matching the requested sector. However, even though there is an entry for the sector in the DCTA, the requested cache line might be located in the LM or not.

❷a ***Requested Cache Line not in DRAM Cache:*** In this scenario, an entry for the sector exists in the DCTA, but the specific cache line is not valid. This indicates that the sector is located in the RM, and only certain cache lines of the sector have been fetched to the DRAM cache. Subsequently, the MP pointer is utilized to retrieve the requested cache line from the RM, while the CP pointer is employed to write the cache line to the appropriate location in the LM.

❷b ***Requested Cache Line in DRAM Cache:*** In this scenario, the requested cache line is located in the DRAM Cache. The sector can be located either in the LM or the RM. In either scenario, the requested cache line is accessible in the LM through the CP pointer of the DCTA entry.

3.3.5 Allocating a Sector in Local Memory

When a DCTA miss occurs and is indicated that the requested sector resides in any of the RM (❶b in Figure 3.3), a new sector must be allocated in LM. To make space for this new sector, another sector must be migrated away to any RMs. When the cache is initially empty at boot, we employ a simple counter to allocate space for the cache within LM. Figure 3.4 illustrates the sector allocation process in the LM. During this process, the DCC (i) identifies the victim sector in the LM, (ii) locates a free sector in the nearest RM for allocation from the Free Memory Stack, (iii) copies the data from the victim sector in the LM to the free sector in RM, and (iv) after the data is copied, the mapping structures are updated to reflect the new location of the sectors in the LM and RM.

3.3.5.1 Finding victim sector in LM

A FIFO policy is employed to identify a victim sector in LM. A Local FIFO counter, wrapping around all the available LM locations, is incremented each time a new location in LM is needed. However, the sector corresponding to the counter may currently be assigned to the DRAM cache (indicated by CP

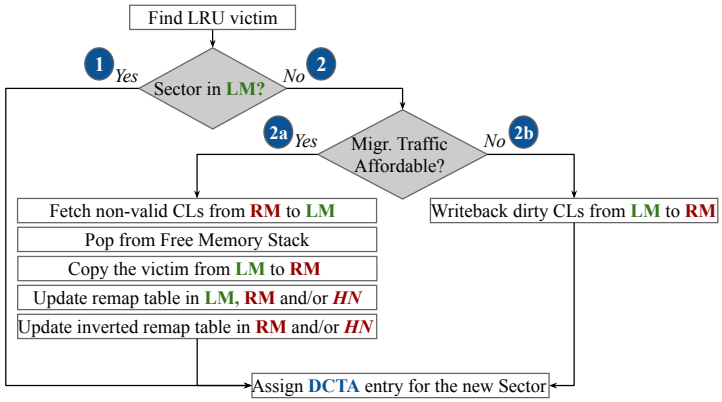


Figure 3.5: Allocating an entry in DCTA

in the DCTA entry) or cached in any of the RMs. To handle this, the inverted remap table is indexed with the counter to obtain the sector's physical address. Then, look up the DCTA using the physical address of the sector. If the sector is in the DCTA, we proceed to the next one until finding an available sector. This ensures correctness, as a sector in the DRAM cache must not be migrated to RM. Furthermore, this approach yields a better replacement decision than FIFO alone, as sectors frequently accessed are more likely to reside in the DRAM cache and avoid migration to RM. To minimize the latency of this step, which is in the critical path of an access, each DCC maintains a buffer of a few (e.g., two) spare, unused DC data entries ready to be used as victim sectors.

3.3.5.2 Finding free sector in RM

To locate a free sector in RM, we utilize the free entries of RM stored in the Free Memory Stack of the node. When a sector is migrated from RM to LM, its original RM location is pushed onto the Free Memory Stack of the RM, making it available to be overwritten.

3.3.6 DRAM Cache Evictions

Figure 3.5 depicts the DRAM cache eviction logic, where the DCC employs the LRU algorithm to determine which sector to evict from the DRAM cache. The DRAM cache can contain (i) Sectors already in the LM, (ii) Sectors that have migrated to the LM, or (iii) Sectors located in the RM, with some or all cache lines already fetched to the LM.

3.3.6.1 Evicting sectors already in LM

For sectors in cases (i) and (ii) involving data already in the LM or migrated to it (1 in Figure 3.5), no data movement is necessary. The remap table has been updated with the evicted sector's location during migration to LM, and the inverted remap table has been updated with the physical address of the evicted sector when first fetched in the DRAM cache. Thus, the corresponding DCTA entry can be reassigned.

3.3.6.2 Evicting sectors located in RM

For sectors in case (iii) located in any of the RMs, the DCC determines whether to migrate the sector to the LM or evict it back to the RM. Migration to LM (2a in Figure 3.5) involves fetching all non-valid cache lines of the sector from the RM and updating migration structures. In contrast, eviction (2b in Figure 3.5) involves writing back all dirty cache lines of the sector to the RM, and no remapping data structures need to be updated. The algorithm for deciding between migration and eviction is detailed in the following section (Section 3.3.7).

3.3.7 Migration Decision and Traffic Regulation

This section discusses the mechanism employed to regulate the migration traffic overheads and the process of deciding between migration and eviction.

3.3.7.1 Migration Traffic Overheads

When evicting a sector from the DRAM cache that has not been migrated to the LM, DCC has two choices: Either (i) evict the sector back to the RM, requiring the writeback of all the dirty cache lines (2b in Figure 3.5), or (ii) migrate the sector to the LM by fetching the non-valid cache lines of the sector from the RM (2a in Figure 3.5). The choice between the two is made based on the migration overhead, which is calculated in terms of the number of RM accesses caused by a migration decision and, in essence, indicate memory traffic cost. The number of RM accesses depends on the number of valid and dirty cache lines within the sector in the DRAM cache.

In the case of eviction, the RM accesses (E_{RM}) correspond to the number of dirty cache lines (N_{dirty}) that must be written back to RM. However, in the case of migration, the RM accesses (M_{RM}) are determined by two factors: firstly, the number of cache lines that need to be fetched from RM, calculated by subtracting the number of valid cache lines (N_{valid}) already present in the sector from the total number of cache lines per sector (N_{all}); and secondly, the cost of swapping out the evicted sector from LM to accommodate the new one, which necessitates N_{all} writebacks to RM.

$$E_{\text{RM}} = N_{\text{dirty}} \tag{3.1}$$

$$M_{\text{RM}} = (N_{\text{all}} - N_{\text{valid}}) + N_{\text{all}} \tag{3.2}$$

Thus, the overhead incurred in migrating a sector in terms of RM accesses (O_{m}) is given by the equation:

$$O_{\text{m}} = E_{\text{RM}} - M_{\text{RM}} + 1 = 2 \times N_{\text{all}} - N_{\text{valid}} - N_{\text{dirty}} + 1 \tag{3.3}$$

where the constant "1" is added as a minimum overhead. The O_{m} can range from 1, indicating all cache lines of a sector are valid and dirty, to $2 \times N_{\text{all}}$, which occurs when only one cache line of a sector is valid and clean upon eviction from the DRAM cache. Nevertheless, this latency overhead does not impact the critical path of memory access, as migration decisions are made only during evictions.

3.3.7.2 Balancing Migration and Processors Traffic

DCC maintains a remote access counter to monitor RM accesses, distinguishing between migration and processor requests. The counter is incremented for every DRAM cache miss that must be fetched from the RM. When a sector is migrated, the counter is decremented by its migration overhead (O_{m}). Besides monitoring the RM accesses, DCC also checks the number of sector accesses (The field AC in DCTA). If enabled, this check ensures that the value of AC of a sector is greater than AC of other sectors in the same cache set. The conjecture is that if this check is successful, the sector is likely to be reused again and thus worth keeping in LM.

When deciding on a sector for migration, its O_{m} is compared with the remote access counter. If O_{m} is smaller than the remote access counter and the above check is met, the sector is considered for migration. Essentially, the remote access counter acts as an upper bound on the number of RM accesses for migration and is periodically reset (every 100K cycles) to adapt to workload phase changes. The check on AC regulate eligible sectors for migration, striking a balance between data "hotness" (usage) and migration cost to optimize system performance. These checks occur during eviction and hence do not affect the critical path of a memory access.

3.3.8 An Example Illustration

Figure 3.6 illustrates how MEMPLEX handles migration metadata. The system has four nodes, each displaying certain memory entries, a portion of the address remap table, and the free memory stack. For simplicity, regions of external DDR are omitted. The physical address of the sector is denoted by **A** (in bold), and the actual address (location within the memory node) is denoted by *A* (in italics). The entire address range includes **A**, **A+1**, ..., **A+n**, **B**, ..., **B+n**, **C**, ..., **C+n**, **D**, ..., **D+n**, divided across 4 memory nodes. Node 0 serves as the Home Node for physical addresses **A** to **A+n**, containing locations *A* to *A+n*, and so on for the remaining nodes. If the sector migrates, it will have an Owner Node where the sector currently resides.

Various scenarios of sector placement and migration are shown: (i) Sectors in their native location, e.g., sector with physical address PA=**A** placed at AA=*A*; (ii) Cached sectors, e.g., sector with PA=**B+2** from Node 1 cached at AA=*A+4* in Node 0 while maintaining its data at AA=*B+2* in Node 1. The corresponding entry for this cached sector would be in the DCTA (not shown) of Node 0; (iii) Migrated sectors, e.g., sector with PA=**D** from Node 3 migrated to AA=*B* in Node 1. The remap tables in Node 1 and Node 3 reflect this migration, and the entry is also in the DCTA (not shown) of Node 1. Since **D** migrated out of Node 3, location *D* is in the Free Memory Stack.

An interesting scenario involves sectors with PA=**C** and PA=**C+1**, which remain in their native location in Node 2 and are cached in the DRAM cache of the same node as the respective processor chiplet uses them. Another scenario occurs when Node 0 misses in its DRAM cache for a line in sector **D+5**. Since the sector is not found in Node 0's remap table, it checks the remap table of sector **D+5**'s home node, i.e., Node 3, which indicates that the sector has migrated to Node 1 at AA=*B+3*, where the requested line is found after adding the line offset.

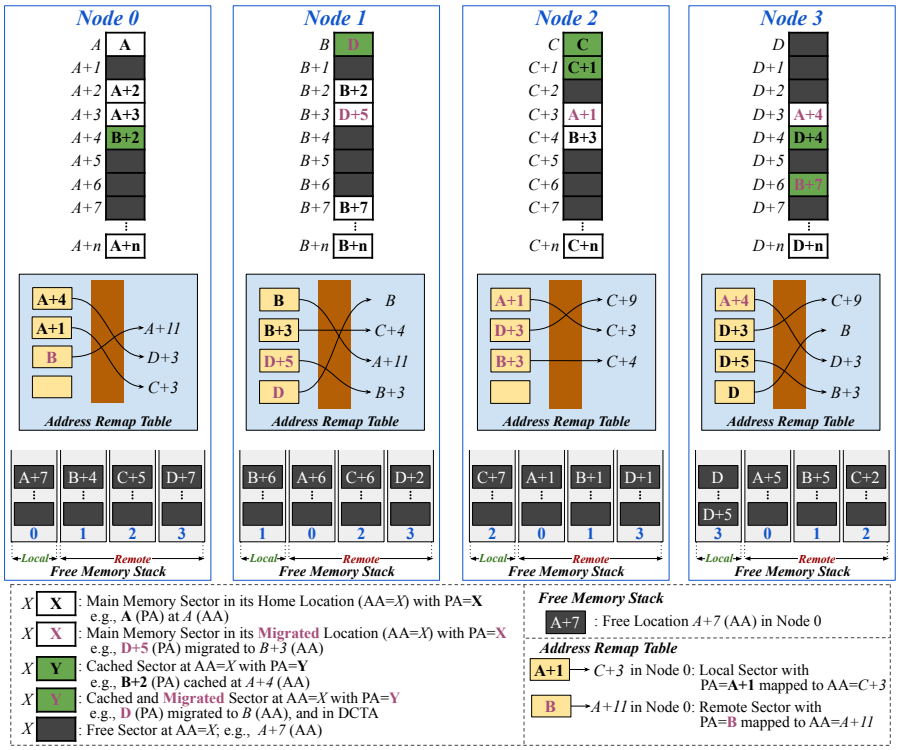


Figure 3.6: Example snapshot of HBM contents and metadata for a 4-node MEMPLEX system. The mapping of sectors with physical addresses (PA - **bold**), to actual addresses (AA - *italics*) of the machine is illustrated, as well as the contents of the remap table and free memory stack for each node.

3.3.9 Cache Coherence

In MEMPLEX, each processor chiplet is paired with a memory node, which allocates a portion of its memory as a private DRAM cache. This DRAM cache can store pages from both local and remote memory nodes. Since a page may reside in multiple DRAM caches simultaneously, a cache coherence protocol is necessary to ensure coherence across the different memory nodes. While the specific cache coherence mechanism for DRAM caches is not detailed in this proposal—leaving room for future optimizations—MEMPLEX is designed to be compatible with a directory-based protocol, such as CANDY [55].

The home node of a cache line can either be statically assigned or dynamically determined by consulting the remap table, which locates the current placement of a sector. The coherence directory is stored in DRAM within each memory node to mitigate the SRAM storage overhead. Additionally, an SRAM cache of the directory is maintained on the processor chiplet to store recently accessed entries. This hybrid approach helps reduce both the substantial storage overhead and the access latency typically associated with coherence directories in the memory nodes. However, the migration policy for shared sectors warrants reconsideration, as sector usage in a single DRAM

cache may not provide enough data for optimal decision-making and could conflict with the migration policies of other nodes. Leveraging the sector's directory to collect information from all sharers could facilitate more efficient migration decisions.
The cache coherence optimizations and their evaluation with multi-threaded workloads are left for future work. Nevertheless, even multi-programmed workloads demonstrate a significant reduction in remote traffic, noticeable performance gain, and substantial energy gains, as detailed in Section 3.5.

3.4 Experimental Setup

3.4.1 System Configuration

Our microarchitectural simulation offers detailed modeling of the memory and interconnects, as outlined in Section 3.4.2, making it computationally intensive for large systems. To keep the simulation times of our experiments within affordable bounds (tens of hours per simulation point), the modeled systems are scaled down to a quarter of a real one. A full-scale AMD Zen4C chiplet consists of 16 cores; therefore, our performance analysis focuses on chiplets scaled to one-quarter of the AMD Zen4C or Intel Sapphire Rapids chiplets. Consequently, these scaled-down chiplets contain only a quarter of the number of cores and connect to a quarter of the HBM channels, as detailed in Table 3.1. Additionally, the L2 and L3 caches are undersized to put more pressure on the memory system and increase LLC misses per kilo instructions (MPKI), which is otherwise difficult to achieve when simulating systems for only a few billion instructions.

Based on the scaled down chiplet size ($16.5\,\mathrm{mm}^2$)[1], the microbump budget is calculated to be proportional to the number of cores it includes. In addition, the following parameters were used for calculating the microbump budget: (i) a microbump pitch of 45 µm, (ii) reserving 40% of the microbumps for power. Then, the number of microbumps available for data were allocated for (i) connecting to the HBM channels, (ii) one bidirectional link to the IO chiplet, (iii) multiple bidirectional links to the other CPU chiplets. Then the width of the links to IO and CPU chiplets, as well as the total number of links to other CPU chiplets were adjusted to fit the microbump budget. Finally, the latency of the inter-chiplet links was measured to be 2 or 3 (NoC) cycles according to the chiplet's dimensions and the latency of the links on a passive silicon interposer similar to [3, 39].

To motivate the use of chiplet-based architectures, we measured the costs of 4, 8, and 16 16-core chiplets, i.e., full scale, with the above configuration compared to their equivalent hypothetical monolithic chips using the chiplet actuary model by Feng and Ma [2]. We considered (i) processor chiplets of size $66\,\mathrm{mm}^2$ manufactured at 5 nm, (ii) a $400\,\mathrm{mm}^2$ IO chiplet at 14 nm, and (iii) passive interposer in 65 nm technology. The analysis showed that the recurring engineering cost of chiplet-based systems were 52-55% of their monolithic counterparts.

[1] Calculated based on Zen4 after scaling down L2 and L3 sizes proportional to the capacity indicated in Table 3.1.

Table 3.1: System Configuration[1]

System	
Chiplets	4 chiplets[1]
Cores and Caches	
Cores	4 cores[1] / chiplet, out-of-order, 3.2 GHz
TLB	I-TLB: 512-entry, 4-way, 1 cycle latency D-TLB: 512-entry, 4-way, 1 cycle latency
L1 Cache	L1-I: Private, 32KB, 4-way, 2 cycle access latency L1-D: Private, 32KB, 4-way, 2 cycle access latency
L2 Cache	Private, 256KB, 8-way, 4 cycle access latency
L3 Cache	Shared, 1MB/core, 16-way, 12 cycle access latency[2]
Main Memory	
HBM2	1 GB/chiplet, 2 GHz, 4 channels, 128 bits per channel, tCAS-tRCD-tRP: 14-14-14 ns, RD/WR+I/O Energy = 6.4 pJ/bit
DDR4	4 GB, 3.2 GHz, 1 channel, 64 bits per channel, tCAS-tRCD-tRP: 22-22-22 ns, RD/WR+I/O Energy = 33 pJ/bit
Network	
Intra-chiplet	2 GHz, 3-stage router (VA/SA, ST, LT), 2x3 Mesh, 4 VCs per port, credit-based flow control, 256 bit link for data, 154 bit link for control (coherence) traffic, 5 flit buffers, XY Routing
Inter-chiplet	2 GHz, 3-stage router (VA/SA, ST, LT), 2x2 Mesh, passive interposer, 2 to 3 cycle link latency[3], 7 to 9 flit buffers [3]

[1] This configuration is the default setting. The parameter adjustments are detailed in the respective evaluation sections of the sensitivity studies.
[2] L3 access latency is 8 cycles for 2MB, 12 for 4MB, and 15 for 8MB.
[3] Depending on the maximum inter-chiplet link length [3].

3.4.2 Simulation Setup

MEMPLEX is evaluated using BZSim [41], which has been extended to model the memory system and interconnects of chiplet-based chips. BZSim is based on the ZSim simulator [42] integrated with BookSim2 [43] for cycle-accurate intra- and inter-chiplet network modeling, enhanced with a technique to detect and skip simulation of low contention traffic to speedup simulation times [41]. BZSim offers microarchitectural simulations with detailed (cycle-accurate) interconnect modeling at an order of magnitude faster simulation speeds compared to GEM5, enabling multi-billion instruction experiments within reasonable times [41]. DRAMSim3 [44] was used for cycle-accurate DRAM modeling and CACTI [45] for estimating cache access times.

The system treats all HBM and external DDR memory as part of a unified flat address space. The virtual memory system was implemented based on HSCC [46]. The cores are configurable with translation lookaside buffers (TLBs) for both instructions and data, as well as with page table walkers (PTWs). Additionally, the memory management modules include a distance-aware allocation policy. This policy allocates pages to the HBM in the chiplet where they are first accessed. If pages are unavailable in the nearest HBM, they are allocated in the next neighboring HBM or in the external DDR.

Table 3.2: Workload Characteristics

Benchmark	Label	Input	LLC MPKI	Footprint (GB)	Assigned to Mixes mix-id$^{\#of instances}$
LLC MPKI 20-40					
pageRank[2]	PRL2	LDBC (100k)	37.41	0.84	1^3,2^3,3^2,4^3,5^3,6^3,7^1
mcf[1]	MCF	Default	34.01	0.45	2^2,5^3,7^2
graphColoring[2]	GCL2	LDBC (100k)	30.70	0.45	1^1,2^1,4^1,6^1,7^2
graphColoring[2]	GCL3	LDBC (10k)	21.26	0.09	1^2,2^1,3^2,5^2,7^1
Random Access Workload[3]	RAND	N=30, M=1000, chunk=1024	20.83	0.70	1^1,2^3,3^1,4^2,5^2,6^1,7^1
LLC MPKI 10-20					
connectedComp[2]	CCL3	LDBC (10k)	19.33	0.09	1^3,2^2,3^1,4^3,5^2,6^2,7^1
lbm[1]	LBM	Default	18.19	0.40	3^1,6^2,7^1
BFS[2]	BFSCR	CA RoadNet	17.25	0.64	1^1,2^1,3^1,4^2,6^2,7^1
fotonik3d[1]	FOTO	Default	17.07	0.59	1^1,4^1
pageRank[2]	PRL3	LDBC (10k)	13.96	0.09	2^1,4^1
xalancbmk[1]	XAL	Default	13.62	0.16	1^1,2^1,3^2,4^1,5^2,6^3,7^1
blender[1]	BLEN	Default	12.78	0.08	2^1,4^1
shortestPath[2]	SPCR	CA RoadNet	12.30	0.64	1^1,3^2,6^1,7^1
XSBench[4]	XSB	XXL	11.11	0.37	7^1
graphColoring[2]	GCCR	CA RoadNet	10.69	0.63	1^1,3^1,5^1,7^1
LLC MPKI 0-10					
parest[1]	PAR	Default	8.54	0.05	3^1
roms[1]	ROMS	Default	7.58	0.25	7^1
triangleCount[2]	TCL2	LDBC (100k)	6.24	0.55	5^1,7^1
graphColoring[2]	GCL1	LDBC (1000k)	5.92	0.29	1^1,4^1,6^1
pageRank[2]	PRKR	Knowledge Repo	4.56	0.30	3^1
BFS[2]	BFSL1	LDBC (1000k)	2.71	0.98	3^1

[1] SPEC CPU 2017 [47], [2] GraphBIG [48], [3] GUPS [49], [4] XSBench [50]

3.4.3 Workloads

Multi-programmed workloads are used in our experiments from the SPEC CPU2017 benchmark suite [47] (the seven with highest MPKI), GraphBIG [48], Random access workload from the GUPS suite [49] and XSBench [50]. For the SPEC CPU2017 and GraphBIG benchmarks, we use Simpoints [51] to select a representative slice of one billion instructions. We have chosen 21 different workloads, detailed in Table 3.2, and created random multi-programmed mixes mapping one benchmark to each core. Each mix of applications has a minimum total memory footprint of 7 GB and a geometric mean LLC MPKI of at least 11. To scale these mixes for systems with 32 or 64 cores, we replicate the 16-application mix twice for the 32-core system and four times for the 64-core system. All experiments run with an average of 125 million instructions per core warm-up period, where memory allocation is enabled, followed by an average of 250 million instructions per core of detailed simulation.

3.4.4 Evaluated Systems

In the evaluation, we consider four distinct systems that offer unique approaches to managing memory resources in a NUMA multi-chiplet architecture.

1. **Baseline (BS)**: A multi-chiplet system with private LLC, NUMA-aware data placement and no support for DRAM caching or migration. The default configuration (depicted in Table 3.1) includes 4 chiplets, each with 4 cores and private LLC, integrated on a passive interposer with 256 bit NoC data-links, 4 HBM channels per chiplet, 1 link to IO chiplet, and 1 channel to external DDR.

2. **DRAM Cache-Only (CO)**: A multi-chiplet system in which each chiplet uses its entire local HBM as a private DRAM cache. The workload mixes are calibrated to fit within a main memory that combines the capacities of HBMs and external DDR. To ensure a fair comparison, the external DDR size in the **CO** is increased to accommodate the workload mix.

3. **MEMPLEX (MP)**: A multi-chiplet system in which a fraction of the HBM is used as a private DRAM cache, while the rest serves as part of the main memory with migration support.

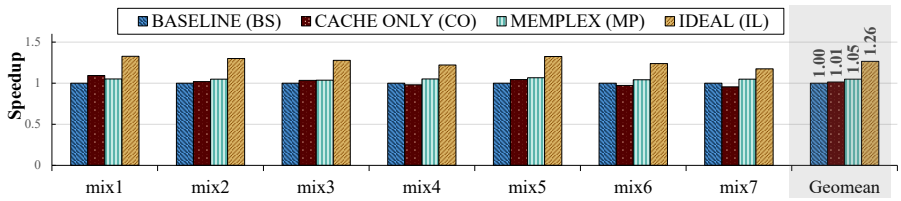
4. **Ideal (IL)**: A multi-chiplet system that operates under the ideal scenario in which an LLC miss is always resolved by the nearest HBM channel, assuming infinite capacity of the local HBM. This setup allows memory allocation solely within the local HBM, ensuring that all memory requests remain local to the chiplet and eliminating the latency associated with accessing remote HBM or external DDR.

3.5 Evaluation

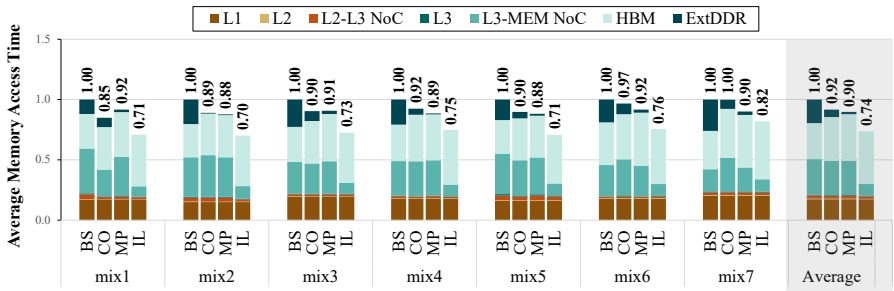
3.5.1 Performance

We evaluate the system performance using Instructions Per Cycle (IPC) as the primary metric. Additionally, Average Memory Access Time (AMAT) is measured and broken down to: (i) the access time for each cache level, (ii) the Network-on-Chip (NoC) latency between each level (L2-L3 and L3-MEM), and (iii) the DRAM access time. Furthermore, the percentage of accesses to local and remote HBM nodes, as well as to external DDR are reported.

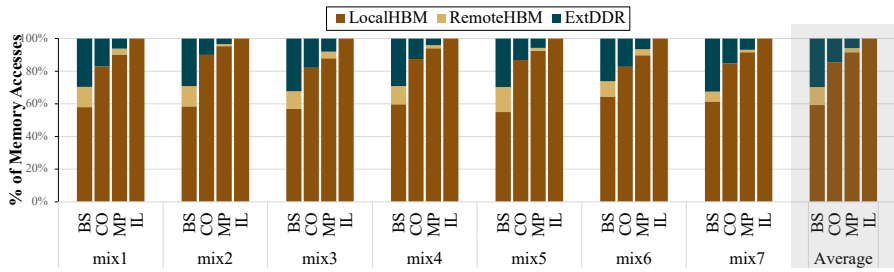
Figure 3.7 illustrates, for each workload mix, the performance speedup (in terms of IPC), the average memory access time, and the distribution of DRAM accesses. These metrics are presented for the default configuration of a 4-chiplet system (**BS**), a DRAM cache-only design (**CO**), the MEMPLEX design with a 1:16 DRAM cache to Main Memory ratio (**MP**), and an ideal system (**IL**). The MEMPLEX system improves baseline performance by 3-7%, with an average improvement of 5%, as shown in Figure 3.7(a). The DRAM cache-only design exhibits a mixed trend across the analyzed workload mixes, with an average performance improvement of 1% over the baseline. This variation is attributed to the differing number of requests, and thus remote traffic to the external DDR for fetching cache lines into the DRAM cache, which is also reflected in the AMAT numbers. In comparison, the ideal system—26% faster than the baseline—achieves this by always directing LLC misses to the nearest HBM. However, MEMPLEX, while 17% slower than the ideal system, still represents a significant improvement over the baseline, effectively reducing the impact of remote requests that are inherent in multi-chiplet NUMA architectures. This



(a) IPC (Normalized to Baseline)



(b) Average Memory Access Time (AMAT) (Normalized to Baseline)

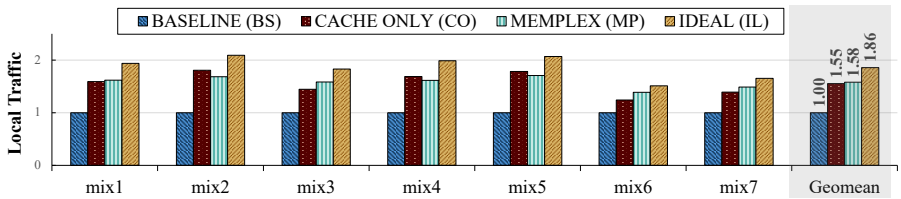


(c) Distribution of Memory Accesses

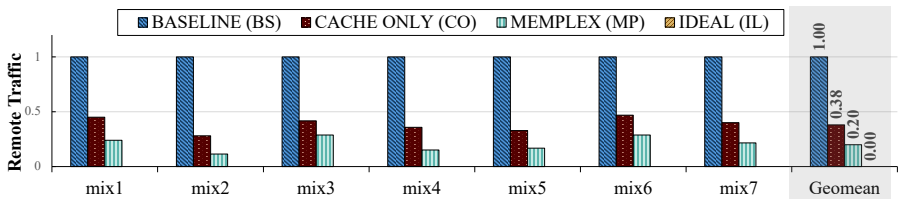
Figure 3.7: Performance, AMAT and distribution of memory access comparison between Baseline, DRAM cache-only, MEMPLEX and Ideal designs.

positions MEMPLEX as a more efficient alternative, narrowing the performance gap toward ideal scenarios.

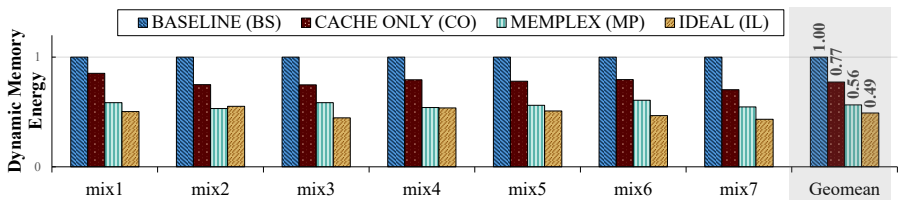
The speedup achieved by MEMPLEX over the baseline is in line with the decrease in AMAT which is on average 10%, as shown in Figure 3.7(b). A closer look reveals that the main source of performance overhead in the chiplet-based system is the significant portion of data accesses placed remotely. As illustrated in Figure 3.7(c), the baseline system experiences an average of 39% of remote data accesses, comprising 10% to remote HBM and 29% to external DDR. Both types of remote accesses involve slow inter-chiplet communication due to limited bandwidth and higher latency. When the entire HBM is dedicated to caching, the DRAM cache-only design reduces remote accesses to just 14%, all of which are directed to external DDR. Meanwhile, MEMPLEX effectively addresses this challenge by bringing 90% of the data within the local HBM, leaving only 10% requiring access to remote memory, thereby significantly reducing the performance impact of remote memory accesses.



(a) Local Traffic (Normalized to Baseline)



(b) Remote Traffic (Normalized to Baseline)



(c) Dynamic Memory Energy (Normalized to Baseline)

Figure 3.8: Local and Remote Memory Traffic, and Dynamic Memory Energy Consumption normalized to the Baseline.

3.5.2 Memory Traffic

Figure 3.8(a) and Figure 3.8(b) show the local and remote memory traffic normalized to the the baseline multi-chiplet system for all workload mixes. The remote memory traffic includes both remote HBM and external DDR accesses. The DRAM cache-only design increases local memory traffic by 55% compared to the baseline. However, this benefit is offset by a 38% increase in remote traffic to the external DDR, highlighting a trade-off that limits its overall performance. In contrast, MEMPLEX delivers a more balanced and effective solution. It generates more local memory traffic compared to the baseline system, with 58% more requests being served from the local HBM. This increase in local traffic is bolstered by an impressive 80% reduction in remote memory traffic, substantially outperforming the 62% reduction achieved by the DRAM cache-only design.

Thus, MEMPLEX effectively combines DRAM caching and data migration to achieve a notable reduction in remote traffic, translating to improved performance and energy savings, as described next. However, the performance gains are not fully maximized, as the MEMPLEX design still involves accesses to the external DDR. Furthermore, unlike the DRAM cache-only design, which dedicates the entire HBM for caching, MEMPLEX sacrifices only $\frac{1}{16}$ of the HBM capacity, demonstrating a far greater resource efficiency.

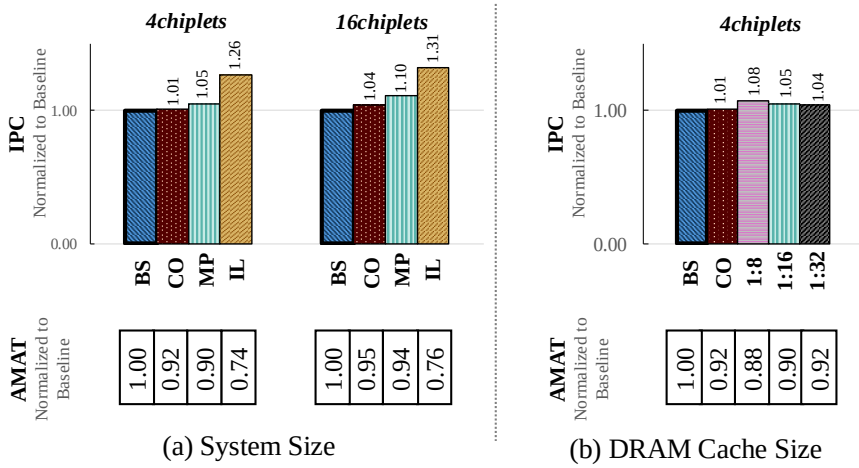


Figure 3.9: Sensitivity analysis on system size and DRAM cache size. All values are the geometric mean of all workload mixes normalized to the Baseline.

3.5.3 Energy Consumption

Figure 3.8(c) presents the dynamic memory system energy consumption normalized to the baseline multi-chiplet system for all workload mixes. The DRAM cache-only design exhibits a 23% reduction in dynamic energy consumption compared to the baseline system. This improvement stems primarily from caching, which significantly reduces remote accesses to external DDR—a major source of energy consumption in the baseline architecture. In comparison, MEMPLEX delivers a remarkable 44% reduction in dynamic memory energy consumption relative to the baseline system. This significant decrease is primarily attributed to lower accesses to remote HBM and external DDR, effectively minimizing high-energy operations and leveraging efficient local memory access. Processor energy and static memory energy (refresh energy) are not reported as these are largely proportional to runtime.

3.5.4 Sensitivity analysis on System Size

An important focus of sensitivity analysis is the effect of system size. Chiplet-based designs are expected to scale more cost-effectively; however, it is still uncertain how their performance overheads change as the system size increases while keeping the number of cores per chiplet constant. To investigate this, in addition to the above evaluated 4-chiplet MEMPLEX system (16-core system), the performance of a 16-chiplet system, which corresponds to 64 cores, is assessed as shown in Figure 3.9(a). As observed above, in a 4-chiplet system, the ideal design would offer 26% higher performance than the baseline and MEMPLEX delivers on average 5% of that speedup and up to 7% for a single mix. As expected, on a 16-chiplet system, the NUMA overheads increase, and so do the MEMPLEX gains. In this case, the ideal design would perform 31% better than the baseline, and MEMPLEX speedup is up to 15% for a single mix and 10% on average.

3.5.5 Sensitivity analysis on DRAM Cache Size

MEMPLEX can be configured with various DRAM cache sizes, and this design choice significantly influences both the performance and the size of the DCTA. Figure 3.9(b) presents the results of a sensitivity analysis exploring different DRAM Cache to Main Memory ratios (1:8, 1:16, 1:32) in comparison to the baseline system without MEMPLEX. In terms of area overhead, the DCTA requires 512 kB, 1 MB, or 2 MB for these respective ratios, assuming an 8-byte entry size. The average performance improvements for the 1:8, 1:16, and 1:32 ratios are 8%, 5%, and 4%, respectively. Notably, the 1:8 ratio can achieve up to a 10% speedup in specific scenarios.

3.6 Conclusions

Multi-chiplet chips provide a cost-effective solution by delivering higher manufacturing yields. However, they encounter performance challenges due to the NUMA memory architecture and inter-chiplet communication bottlenecks. In this study, we analyzed these overheads, showing that an ideal 4- and 16-chiplet system would offer 26% and 31% higher performance, respectively, compared to a baseline with NUMA-aware data placement. To address these challenges, we proposed MEMPLEX, a novel architecture that enables data replication and migration across multiple memory nodes within a multi-chiplet system. MEMPLEX efficiently dedicates a portion of each memory node for a DRAM cache, while the remaining capacity is utilized as a shared flat address space with hardware migration. Thereby, MEMPLEX enhances data locality, bringing frequently accessed data closer to the processor and managing migration based on usage patterns. As a result, MEMPLEX reduces remote memory traffic by 80%, leading to a significant 44% dynamic memory energy consumption. In a 4-chiplet system, MEMPLEX achieves up to 7% speedup and an average of 5% when dedicating $\frac{1}{16}$ of each HBM for caching. When $\frac{1}{8}$ of the HBM capacity is used for caching, the performance gain increases to up to 10%, with an average of 8%. Finally, the performance benefits of MEMPLEX performance are more pronounced in larger systems, with the average speedup improving from 5% to 10% as the system size increases from 4 to 16 chiplets.

Bibliography

[1] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Jun 2021, pp. 57–70.

[2] Y. Feng and K. Ma, "Chiplet Actuary: A Quantitative Cost Model and Multi-Chiplet Architecture Exploration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, Jul 2022, p. 121–126.

[3] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh, "Cost-Effective Design of Scalable High-performance Systems Using Active and Passive Interposers," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 728–735.

[4] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling Interposer-based Disintegration of Multi-core Processors," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 546–558.

[5] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar 2021.

[6] NEC CORPORATION, "SX-Aurora TSUBASA Architecture Guide Revision 1.1," 2018. [Online]. Available: https://sxauroratsubasa.sakura.ne.jp/documents/guide/pdfs/Aurora_ISA_guide.pdf. Last Accessed: 7 Feb 2025

[7] Advanced Micro Devices, Inc., "AMD CDNA™ 3 Architecture," 2023. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf. Last Accessed: 7 Feb 2025

[8] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula, R. Marom, A. M. Kern, B. Bowhill, D. R. Mulvihill, S. Nimmagadda, V. Kalidindi, J. Krause, M. M. Haq, R. Sharma, and K. Duda, "Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2022, pp. 44–46.

[9] T. Burd, N. Beck, S. White, M. Paraschou, N. Kalyanasundharam, G. Donley, A. Smith, L. Hewitt, and S. Naffziger, "Zeppelin: An SoC for Multichip Architectures," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 54, no. 1, pp. 133–143, Oct 2019.

[10] N. Beck, S. White, M. Paraschou, and S. Naffziger, "Zeppelin: An SoC for Multichip Architectures," in *2018 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2018, pp. 40–42.

[11] Y. Feng, D. Xiang, and K. Ma, "A Scalable Methodology for Designing Efficient Interconnection Network of Chiplets," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb 2023, pp. 1059–1071.

[12] Y. Feng, D. Xiang, and K. Ma, "Heterogeneous Die-to-Die Interfaces: Enabling More Flexible Chiplet Interconnection Systems," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2023, p. 930–943.

[13] F. Dahlgren and J. Torrellas, "Cache-Only Memory Architectures," *Computer*, vol. 32, no. 6, pp. 72–79, Jun 1999.

[14] Z. Zhang and J. Torrellas, "Reducing Remote Conflict Misses: NUMA with Remote Cache versus COMA," in *Proceedings Third International Symposium on High-Performance Computer Architecture (HPCA)*, Feb 1997, pp. 272–281.

[15] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 126–136.

[16] A. Kokolis, D. Skarlatos, and J. Torrellas, "PageSeer: Using Page Walks to Trigger Page Swaps in Hybrid Memory Systems," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 596–608.

[17] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "LLC-Guided Data Migration in Hybrid Memory Systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2019, pp. 932–942.

[18] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 433–444.

[19] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent Hardware Management of Stacked DRAM as Part of Memory," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2014, pp. 13–24.

[20] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "CHAMELEON: A Dynamically Reconfigurable Heterogeneous Memory System," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 533–545.

[21] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 349–360.

[22] G. H. Loh and M. D. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011, pp. 454–464.

[23] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2012, pp. 235–246.

[24] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2014, pp. 25–37.

[25] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, Jun 2013, pp. 404–415.

[26] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Decoupled Fused Cache: Fusing a Decoupled LLC with a DRAM Cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, pp. 65:1–65:23, Jan 2019.

[27] C. Chou, A. Jaleel, and M. K. Qureshi, "BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, Jun 2015, pp. 198–210.

[28] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A Fully Associative, Tagless DRAM Cache," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, Jun 2015, pp. 211–222.

[29] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient Footprint Caching for Tagless DRAM Caches," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Mar 2016, pp. 237–248.

[30] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache," in *2014 23rd International*

Conference on Parallel Architecture and Compilation Techniques (PACT), Aug 2014, pp. 51–60.

[31] G. Loh and M. D. Hill, "Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, May 2012.

[32] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Hybrid^2: Combining Caching and Migration in Hybrid Memory Systems," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2020, pp. 649–662.

[33] D. Stow, I. Akgun, and Y. Xie, "Investigation of Cost-Optimal Network-on-Chip for Passive and Active Interposer Systems," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, Jun 2019, pp. 1–8.

[34] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "NoC Architectures for Silicon Interposer Systems: Why pay for more wires when you can get them (from your interposer) for free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2014, pp. 458–470.

[35] K. Sarcar, "What is NUMA?" Nov 1999. [Online]. Available: https://www.kernel.org/doc/html/v5.4/vm/numa.html. Last Accessed: 7 Feb 2025

[36] K. Bridge, "NUMA Support - Win32 Apps," May 2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/procthread/numa-support. Last Accessed: 7 Feb 2025

[37] A. Chadd, "FreeBSD Manual Pages," Oct 2018. [Online]. Available: https://man.freebsd.org/cgi/man.cgi?query=numa&sektion=4&manpath=FreeBSD%2B14.0-RELEASE%2Band%2BPorts. Last Accessed: 7 Feb 2025

[38] Y. Yarom, Q. Ge, F. Liu, R. B. Lee, and G. Heiser, "Mapping the Intel Last-Level Cache," *IACR Cryptology ePrint Archive*, Jan 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:1281871

[39] A. Coskun, F. Eris, A. Joshi, A. B. Kahng, Y. Ma, and V. Srinivas, "A Cross-Layer Methodology for Design and Optimization of Networks in 2.5D Systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

[40] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, USA: Morgan Kaufmann Publishers Inc., 2004.

[41] P. Strikos, A. Ejaz, and I. Sourdis, "BZSim: Fast, Large-Scale Microarchitectural Simulation with Detailed Interconnect Modeling," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, May 2024, pp. 167–178.

[42] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, Jun 2013, p. 475–486.

[43] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-accurate Network-on-Chip Simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr 2013, pp. 86–96.

[44] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters (CAL)*, vol. 19, no. 2, pp. 106–109, Jul 2020.

[45] S. J. Wilton and N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 31, no. 5, pp. 677–688, 1996.

[46] H. Liu, Y. Chen, X. Liao, H. Jin, B. He, L. Zheng, and R. Guo, "Hardware/Software Cooperative Caching for Hybrid DRAM/NVM Memory Architectures," in *Proceedings of the International Conference on Supercomputing (ICS)*, Jun 2017, pp. 26:1–26:10.

[47] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 271–282.

[48] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "GraphBIG: Understanding Graph Computing in the Context of Industrial Solutions," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–12.

[49] S. J. Plimpton, R. Brightwell, C. Vaughan, K. Underwood, and M. Davis, "A Simple Synchronous Distributed-Memory Algorithm for the HPCC RandomAccess Benchmark," in *IEEE International Conference on Cluster Computing (Cluster 2006)*, Sep 2006, pp. 1–7.

[50] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis," in *Proceedings of the International Conference on Physics of Reactors (PHYSOR 2014)*, Sep 2014.

[51] E. Perelman, G. Hamerly, and B. Calder, "Picking Statistically Valid and Early Simulation Points," in *2003 12th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep 2003, pp. 244–255.

[52] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," in *24th Annual International Symposium on Computer Architecture (ISCA)*, Jun 1997, pp. 241–251.

[53] T. Lovett and R. Clapp, "STiNG: A CC-NUMA Computer System for the Commercial Marketplace," in *23rd Annual International Symposium on Computer Architecture (ISCA)*, May 1996, pp. 308–317.

[54] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, "The Stanford Dash Multiprocessor," *Computer*, vol. 25, no. 3, pp. 63–79, Apr 1992.

[55] C. Chou, A. Jaleel, and M. K. Qureshi, "CANDY: Enabling Coherent DRAM Caches for Multi-Node Systems," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[56] C.-C. Huang, R. Kumar, M. Elver, B. Grot, and V. Nagarajan, "C^{3}D: Mitigating the NUMA Bottleneck via Coherent DRAM Caches," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[57] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, "Operating System Support for Improving Data Locality on CC-NUMA Compute Servers," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Sep 1996, p. 279–289.

[58] P. Stenstrom, T. Joe, and A. Gupta, "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures," in *Proceedings the 19th Annual International Symposium on Computer Architecture (ISCA)*, May 1992, pp. 80–91.

[59] E. Hagersten, A. Saulsbury, and A. Landin, "Simple COMA Node Implementations," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences (HICSS)*, vol. 1, 1994, pp. 522–533.

[60] B. Falsafi and D. A. Wood, "Reactive NUMA: A Design For Unifying S-COMA And CC-NUMA," in *24th Annual International Symposium on Computer Architecture (ISCA)*, Jun 1997, pp. 229–240.

[61] C. Lameter and M. Kim, "Page Migration," Mar 2016. [Online]. Available: https://www.kernel.org/doc/Documentation/vm/page_migration. Last Accessed: 7 Feb 2025

[62] J. Herrmann, "Automatic NUMA Balancing Red Hat Enterprise Linux 7," May 2019. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_tuning_and_optimization_guide/sect-virtualization_tuning_optimization_guide-numa-auto_numa_balancing. Last Accessed: 7 Feb 2025

Appendix A

Additional Results On Cost Analysis of Chiplet-Based Systems

This appendix provides additional results on the cost analysis of various design options for chiplet-based systems. This supplementary material offers further insights and supports the findings presented in **Paper A**.

A.1 Cost Analysis

As detailed in Section 2.2.4, this thesis evaluates the associated costs using the Feng-Ma chiplet actuary model [2], incorporating the parameters of the evaluated systems. Figure A.1 presents a detailed breakdown of the recursive engineering (RE) cost for the different design options of chiplet-based systems and their monolithic counterpart. The total RE cost comprises:

(i) **Cost of raw chips**, which includes, among others, the cost of the silicon and the processing involved in wafer fabrication.
(ii) **Cost of chip defects**, which includes the cost of defects that occur during the wafer fabrication process.
(iii) **Cost of raw package**, which covers the materials necessary for assembling and packaging the chip, as well as testing and verification.
(iv) **Cost of package defects**, which includes the costs arising from defects during the packaging process.
(v) **Cost of wasted Known Good Dies (KGD)**, which encapsulates the cost of dies that have already been tested to ensure their correct functionality but still fail.

System Size: Using the cost model and parameters mentioned in Section 2.2.4, we derived the costs of 64-, 128-, and 256-core systems. In chiplet-based systems, these cores are partitioned into 4+1, 8+1, and 16+1 configurations, respectively,

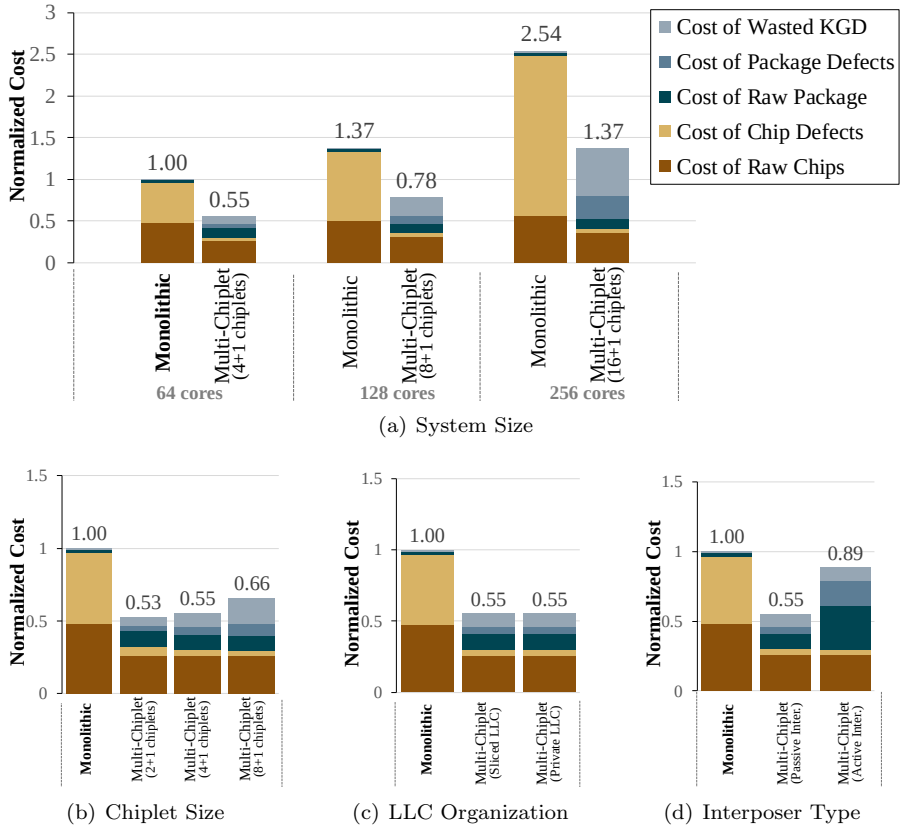


Figure A.1: Cost analysis and comparison of RE costs between monolithic and various chiplet-based systems, with costs normalized to the monolithic.

including an IO chiplet in each case. Figure A.1(a) presents the cost values normalized to the monolithic of the smallest system. The results indicate that the chiplet-based designs reduce costs to approximately 55% of their monolithic counterparts. The cost gap between monolithic and multi-chiplet systems remains relatively stable or increases slightly due to the model's conservative bonding and packaging yield estimates. Nevertheless, the results highlight the substantial cost savings of chiplet-based designs over monolithic designs.

Chiplet Size: We examined the impact of chiplet size, which refers to the number of cores per chiplet, by partitioning a 64-core system into 2+1, 4+1, and 8+1 chiplet configurations, each including an IO chiplet. Figure A.1(b) presents the cost values normalized to the monolithic design. The results show that while the first chiplet-based system has a reasonable cost, increasing the number of chiplets causes a non-linear rise in overall system cost. This can be attributed to the increased package defects and wasted KGD as chiplet size increases, reducing yield and raising costs.

LLC Organization: Another key design choice was the LLC organization in the chiplet-based systems. We evaluated a 64-core system partitioned into a 4+1 chiplet configuration. Figure A.1(c) presents the cost values normalized to the monolithic design. While the LLC organization does not directly influence the cost, Section 2.4 demonstrates that a private LLC significantly outperforms a sliced LLC in terms of system performance.

Interposer Type: The last design choice was the different types of silicon interposers for multi-chiplet systems, evaluated on a 64-core system partitioned into a 4+1 chiplet configuration. Figure A.1(d) presents the cost values normalized to the monolithic design. Unlike passive interposers, active interposers incorporate active logic to pipeline network links, thus improving the network's latency and throughput. While active interposers enhance system performance (as demonstrated in Section 2.4), they come at a 61% higher cost than passive interposers, making their cost comparable to monolithic designs.

Overall, the analysis highlights the cost trade-offs in chiplet-based designs and the significant cost savings they can achieve.