THE UNIVERSITY OF CHICAGO

MEMORY HIERARCHY DESIGNS FOR TILED HETEROGENEOUS ARCHITECTURES

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
CHEN ZOU

CHICAGO, ILLINOIS
2019.04.01
# TABLE OF CONTENTS

LIST OF FIGURES ......................................................... v

LIST OF TABLES ........................................................ vi

ABSTRACT .............................................................. vii

1 INTRODUCTION ......................................................... 1
   1.1 Motivation ....................................................... 1
   1.2 Tiled heterogeneous Architecture .............................. 3
   1.3 Approach ........................................................ 4
   1.4 Contributions ................................................... 5
   1.5 Thesis organization ............................................ 7

2 BACKGROUND .......................................................... 8
   2.1 OpenMP .......................................................... 8
      2.1.1 Directive based parallel programming model ............... 8
      2.1.2 Fork-join execution model ................................ 8
      2.1.3 Synchronizations ........................................... 10
   2.2 Memory system designs for existing systems ................. 11
      2.2.1 Memory system designs for CPU based systems .......... 11
      2.2.2 Memory system designs for GPU systems ................ 13
      2.2.3 Cache circuit design ..................................... 15

3 EXPERIMENTS ......................................................... 17
   3.1 Methodology .................................................... 17
      3.1.1 Model scope ............................................... 18
      3.1.2 The performance model .................................... 19
      3.1.3 The area and power model ................................ 24
      3.1.4 The application execution model ......................... 25
      3.1.5 Workloads ................................................ 27
      3.1.6 Trace-driven simulation .................................. 27
      3.1.7 Metrics ..................................................... 29
   3.2 Memory Hierarchy Challenges for Tiled Heterogeneous Architectures .................................... 30
      3.2.1 L1 data cache Access Rate Requirements ................ 30
      3.2.2 Coherence traffic in a tile ................................ 32
      3.2.3 LLC and memory bandwidth requirement per tile ......... 34
      3.2.4 LLC bandwidth imbalance from a tile .................... 34
   3.3 Evaluations on proposed cache system improvements ......... 36
      3.3.1 Associativity makes way for multi-banking L1 data caches 37
      3.3.2 Tile-level-shared L2 cache to eliminate LLC bandwidth imbalances 39
   3.4 Summary .......................................................... 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>DESIGN SPACE EXPLORATIONS</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Design Space Explored</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Designing the core-accelerator tile</td>
<td>45</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Scaling up the accelerator speed</td>
<td>45</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Varying L2 cache size and associativity</td>
<td>48</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Design space exploration for core-accelerator tiles</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>Designing the tiled heterogeneous chip</td>
<td>54</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Scaling out optimal tiles</td>
<td>54</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Design space exploration for the tiled heterogeneous chip</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>DISCUSSIONS AND RELATED WORK</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>Reducing memory bandwidth requirements</td>
<td>63</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Effective data sharing</td>
<td>63</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Compressing data on caches or main memory</td>
<td>65</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Adaptive cache access granularity</td>
<td>66</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Design space explorations</td>
<td>66</td>
</tr>
<tr>
<td>5.2</td>
<td>Increasing available bandwidth</td>
<td>67</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Employing new memory technology with higher bandwidth capacity</td>
<td>67</td>
</tr>
<tr>
<td>5.2.2</td>
<td>In-memory processing</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Summary</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>SUMMARY AND FUTURE WORK</td>
<td>69</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary</td>
<td>69</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Work</td>
<td>70</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td>72</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

1.1 System architecture in study and its baseline ........................................ 4
2.1 An example of specifying a parallel loop with a directive ........................ 9
2.2 Parallel construct implementation ......................................................... 10
2.3 The fork-join execution model of OpenMP ............................................. 10
2.4 Example of a barrier based synchronization ............................................ 11
2.5 CPU Memory System ........................................................................... 12
2.6 GPU Memory System ........................................................................... 14
2.7 Cache structure (extracted from cacti 1.0 tech report [52]) ....................... 16
3.1 The performance model for the tiled heterogeneous system and the baseline homogeneous system ................................................................. 19
3.2 Performance throttling with limited LLC bandwidth ................................. 22
3.3 The application execution model for the tiled heterogeneous system and the baseline homogeneous system .......................................................... 26
3.4 Multithreaded OpenMP traces ................................................................ 28
3.5 Online trace-driven simulation based on ZMQ and ZSim ......................... 29
3.6 Tile speedup with variant speed and #L1D banks .................................... 31
3.7 Coherence operations (cross invalidations (light color) or cross downgrades (dark color)) per kilo references ......................................................... 33
3.8 Required LLC and Memory Bandwidth per tile ....................................... 35
3.9 Average LLC bandwidth requirement from the core and the accelerator respectively .......................... 36
3.10 Different types of multi-banking ............................................................ 38
3.11 Sharing a single L2 cache in each tile .................................................... 40
3.12 Global L2 miss rate before and after employing a single tile-level shared L2 cache ................................................................. 40
4.1 Design space explored ............................................................................ 43
4.2 Explorations on the accelerator speed ..................................................... 46
4.3 Explorations on L2 configurations .......................................................... 49
4.4 Tile speedup, normalized area efficiency, normalized power efficiency and L2 cache size choices of optimized designs for different metrics at each accelerator speed .................................................. 52
4.5 Chip performance, area efficiency and power efficiency for three optimal tile design when scaling the number of tiles. X-axises are the number of tiles in a chip. .................................................. 55
4.6 Pareto fronts of the chip designs ............................................................. 58
4.7 Configurations of the designs in the pareto fronts of HBM2 systems ............ 58
4.8 Memory bandwidth requirement for different LLC slice size ................. 59
LIST OF TABLES

3.1 Baseline configurations for caches ........................................... 19
3.2 Area & Power estimation for a conventional core .......................... 24
3.3 Survey on commodity processors and accelerators .......................... 24
3.4 ECP proxy apps and their descriptions ..................................... 27
3.5 Comparsion table between a 32KB 8-way associative cache and a 256KB direct mapped cache ...................................................... 39

4.1 Optimal tiles for three different metrics under two LLC bandwidth throttling scenarios ................................................................. 53
ABSTRACT

Heterogeneous architectures based on accelerators are important paths to high performance. However, acceleration increases the performance demands for memory hierarchy designs. In this study, our focus is tiled heterogeneous architectures that pair accelerators with conventional cores and tile them across the CPU chip.

To understand the memory hierarchy requirements and challenges of such tiled heterogeneous architectures, we study a generic accelerator architecture and different memory system configurations using a trace-driven simulation framework with a set of high-performance computing benchmarks. We assess simulation results via performance and area/power consumption analyses.

Our results highlight the bandwidth challenges of the memory hierarchy that tiled acceleration incurs. Each level of the hierarchy (L1, L2, LLC) must deal with the increased bandwidth requirement, imbalanced off-tile traffic, excessive off-tile and off-chip bandwidth requirement respectively. Together, they can reduce system performance by up to 5.2x.

To distill understandings and design insights, we evaluated a set of designs that vary each level of the memory hierarchy. We propose design insights that address the important challenges and deliver good area and power efficiency. For increased bandwidth requirement at L1 cache, a high-bank-parallelism L1 cache organization is proposed. Large tile-level shared L2 caches are recommended to not only effectively pooling the cache capacity and off-tile bandwidth capacity but also mitigate the off-tile and off-chip bandwidth bottlenecks. Moreover, a balance between scaling up the tile and scaling out the tiles is encouraged to mitigate the main memory bandwidth bottlenecks which ultimately limit the acceleration benefits.

Finally, to show the potential benefits of a tiled heterogeneous architecture, we performed design optimizations in a broad design spaces. Among a set of optimized designs in a Pareto front trading performance with energy efficiency, a tiled heterogeneous chip with 16 tiles,
12x faster accelerator in each tile and a carefully optimized memory hierarchy brings 3.2x performance than a homogeneous chip of 16 baseline tiles. Acceleration alone with only baseline memory hierarchy design brings 2.2x performance. The use of the proposed high-bandwidth L1 caches improves the performance to 2.9x, and other memory hierarchy designs mitigating bandwidth bottlenecks at LLC and main memory brings the system performance to 3.2x.
CHAPTER 1
INTRODUCTION

1.1 Motivation

With the end of Moore’s law, even the multi-core scaling paradigm for continual growth of processor performance is coming close to an end. The reason is that no more transistors are available to build additional cores. Thus, we need new solutions to to continue the increase of performance and energy efficiency of computing systems. One promising proposal among them is accelerators.

By accelerator, we mean any architecture element that increases performance (energy or area efficiency) dramatically relative to a traditional core. We have seen more and more industry adoptions of the idea to bring accelerators into the computing system. Google developed the Tensor Processing Unit (TPU) [34] for accelerated deep learning inference and training iteratively to its third generations. Microsoft’s Catapult acceleration architecture [22] went from a research project into production to accelerate searching, neural networks and software-defined networks. Amazon introduced FPGA instances [1] into its AWS for customers to rent. Intel released a Xeon-FPGA product [14] where the FPGA is in the cache coherent domain of the SkylakeSP cores, bringing acceleration closer to the core.

However, most of these accelerators live on top of completely or partially separate memory hierarchies rather than sharing the one serving conventional CPU cores. It enlarges the latency and decreases the flexibility for sharing data between the heterogeneous computing elements, and demands increased costs of area and power just to build and operate the separate memory hierarchies.

In this study, we focus on the heterogeneous system that features tight coupling of cores and accelerators on the same memory hierarchy as seen in many studies [23, 50, 27]. This organization addresses both the linear and parallel portions of the Hill-Marty model [31]
efficiently by cores and accelerators respectively. Moreover, it features low latency communication between the two computing elements and enables scaling with a single memory hierarchy.

On the other hand, there is a well known ‘memory wall’ problem [53] dated back to decades ago, where the technology process scaling made it possible for single core performance to grow at 50% per year and the performance of the memory chips grew only at 20% per year, creating a huge gap between processor speed and memory speed [30]. The switch to multi-core scaling at the end of the Dennard scaling didn’t make the problem easier to solve. Although the performance of each core almost doesn’t grow while the main memory performance continues to grow, memory traffics grow roughly linearly with the number of cores, maintaining if not aggravating the mismatch between cores and main memory. Now, when it comes to the accelerators era, considering that accelerators are of higher performance, the introduction of them into the computing systems puts even greater stresses on the memory hierarchy to bridge the performance gap between the computing elements and main memory, exacerbating the memory wall problem. As a result, the memory hierarchy design for a accelerated heterogeneous system is of crucial importance to tackle the aggravated memory wall and deliver acceleration benefits.

There are extensive studies on how to avoid being performance limited by the memory wall. Caches are employed as bandwidth filters along with its latency reduction benefits, and get enhanced with methods like non-blocking caches, pipelined caches and multi-banked caches for both single core and multi-core systems [30]. There are also attempts to compress the data in main memory to save the memory bandwidth usage or to filter the unused words in a cache line to prevent wasting the precious memory bandwidth [17, 49]. However, there are no works for tiled accelerations. Unlike the accelerators attached to the computing system through PCIe, accelerators introduced tightly coupled with the core in the same memory hierarchy directly aggravates the challenges of the memory system designs. This calls for new
studies for the memory hierarchy designs for the tiled heterogeneous architecture tackling the memory wall problems.

1.2 Tiled heterogeneous Architecture

As discussed, we focus on the acceleration approach that integrates each core and accelerator tightly into a tile sharing a single memory hierarchy. In this section, we will first describe the class of tiled heterogeneous architectures we are targeting. Subsequently, we raise the key research questions for memory hierarchy and system design.

A tiled heterogeneous architecture computing chip is composed of a set of replicated designs called tiles, typically in a regular layout, and other chip-level resource shared by all the tiles. Each tile is composed of two different computing elements paired with each other, one core and one accelerator as shown in Figure 1.1. Please notice the differences of tiled heterogeneous architecture from the homogeneous computing architectures or the architectures attached with GPUs or other PCIe based accelerators (e.g. TPU), which include the tight integration of accelerators and shared memory hierarchy between cores and accelerators. The two computing elements in each tile employ private caches for fast memory accesses and bandwidth filtering. And there are also larger caches in each tile filtering the off-tile traffic going to the shared chip-level resources, and shared last-level caches (LLC) at chip-level to facilitate data sharing across the tiles and filter the memory traffic going off the chip.

The introduction of accelerators with higher performance creates greater memory hierarchy performance demands, such as large increases in bandwidth requirements on each level of the memory systems. The general question this research trying to solve is how to design the memory hierarchy for the tiled heterogeneous architecture to maximize the acceleration benefits. Specifically, the first question is how should the first-level private caches directly serving the accelerator be designed? Accelerators are of higher performance and thus pro-
duce higher memory reference issue rates, posing challenges for the design of its private caches. Secondly, despite data sharing opportunities, a tightly coupled accelerators also produce higher L2 miss rates, hitting the limits of the outstanding misses and LLC bandwidth. We seek cache hierarchy designs that reduce the traffics coming out of the backside of the L2. Furthermore, considering that faster accelerators would result in higher off-chip memory bandwidth requirement, the third research question is how the cache hierarchy should be designed to filter the traffic and require only feasible main memory bandwidth for the whole chip.

1.3 Approach

To maximize the generality of our memory hierarchy study results, we minimize assumptions about internal accelerator structure and behavior. The accelerator is assumed to deliver increased performance at an equal increase in memory traffic, and is tolerant of memory latency. This enables us to gain architecture knowledge and memory design insights for a set of tiled heterogeneous architecture employing different accelerators.

We employed trace-driven simulations with the above accelerator assumptions and model for performance and cost analyses to compare different architectures. Scalable scientific
workloads, namely, proxy applications for the Exascale Computing Project (ECP) were employed as the simulation workloads. Although trace-driven simulations don’t capture the execution timing interaction of different computing elements, they model the increased load the workloads impose on the memory hierarchy which is the core of our study. Trace-driven simulations also enable controlled experiments on memory hierarchies of different configurations.

To answer the aforementioned questions on the memory hierarchy designs for the tiled heterogeneous system, we took the following steps with the methods above:

Firstly, we evaluate several designs focused around key memory hierarchy and system research questions through trace-driven simulations. We start by focusing at tile-level on how private caches should be designed to serve the high-performance accelerators of high reference issue rates. And then, we look into L2 cache organizations on effective pooling off-tile traffic as well as cache capacity in a tile to mitigate the LLC bandwidth bottlenecks.

Secondly, we performed design optimizations across a design space of both accelerator sizing and cache configurations to consider the memory hierarchy designs under different accelerator scaling. The capacity allocation between L2 and LLC is explored to mitigate both LLC and main memory bandwidth bottlenecks. Furthermore, we seek a balance between scaling up and scaling out of the accelerated tiles to avoid bandwidth bottlenecks. Finally, pareto fronts trading performance with energy and area efficiency are explored to identify the maximum acceleration benefit can be achieved with the tiled acceleration and how our memory hierarchy design insights contribute to it.

1.4 Contributions

The specific contributions of this work are:

- The first trace-driven study of a tiled heterogeneous architecture to understand the implications of increased tile performance on chip memory hierarchy and memory system
• As accelerators speed computation, they have a fundamental requirement for a higher memory reference rate. For accelerators 4-16x faster than a reference core, L1 data caches with 8-32 banks are most attractive, supplying enough access parallelism and minimizing bank conflicts.

• An accelerated tile couples a core (fast single-thread) and an accelerator (parallel) tightly. However, the accelerator produces higher rates of L2 misses, hitting limits of outstanding misses and bandwidth to the backing LLC. An L2 shared by core and accelerator, allowing pooling of miss and LLC bandwidth resources as well as cache capacity, improves performance of the tiled heterogeneous architecture.

• Ultimately, the benefits of acceleration are limited by global resource constraints such as LLC bandwidth or memory bandwidth. Subject to these, we find that a balance of accelerator scaling and number of tiles produces the best area and power efficiency.

• The tiled heterogeneous architecture with high-performance accelerators aggravates the bandwidth bottlenecks at last-level caches and main memory, especially for memory-intensive workloads. This shifts the desirable area allocation across L2 and LLC towards L2 (even as large as 2MB or 4MB, especially if LLC is non-inclusive), because a larger L2 cache mitigates both the LLC bandwidth bottleneck and the memory bandwidth bottleneck, and saves overall power consumption through reducing off-tile traffics.

• Among a set of optimized designs in the Pareto front trading performance with energy efficiency from our design space explorations, an optimal design point shows the effectiveness of the above insights. A tiled heterogeneous chip design with 16 tiles, 12x faster accelerator in each tile and a carefully optimized memory hierarchy brings 3.2x performance than a homogeneous chip of 16 baseline tiles. Acceleration alone with
only baseline memory hierarchy design brings 2.2x performance. The use of the proposed high-bandwidth L1 caches improves the performance to 2.9x, and other memory hierarchy designs mitigating bandwidth bottlenecks at LLC and main memory brings the system performance to 3.2x.

1.5 Thesis organization

The rest of the thesis is organized as the following. We will introduce the background on OpenMP programming models and the existing memory system designs in Chapter 2. In Chapter 3, we first discuss the experiment methodology including the performance, application execution, area and power model for the tiled heterogeneous architecture and then the experiments focus around key memory hierarchy and system research questions to distill design insights of the memory systems in the tiled heterogeneous computing system. In Chapter 4, design explorations are performed to consider the memory hierarchy designs along with the different accelerator scaling. We also identify a set of designs on the pareto fronts trading performance and system efficiency to show the acceleration benefits and the contributions of memory hierarchy designs. We then discuss the related work in Chapter 5. Finally, we summarize the work and discuss the opportunities for future research in Chapter 6.
CHAPTER 2
BACKGROUND

2.1 OpenMP

OpenMP is a application programming interface that supports shared memory multiprocessing in C, C++ and Fortran on different platforms consisting of a set of compiler directives, library routines, and environment variables that influence run-time behavior [10]. We introduce the OpenMP programming model in this section considering that OpenMP is the programming model we used for tiled heterogeneous architecture which is further discussed in Section 3.1.4, and the trace collection pintool we developed is closely connected to GCC’s OpenMP implementations to capture the application parallelism.

2.1.1 Directive based parallel programming model

In general, OpenMP enables users to specify the work to be executed in parallel in an openmp construct. By marking a code block with an OpenMP directive (e.g. #pragma omp parallel), the compiler would generate the code to execute the content of the code block in parallel by the corresponding hardware with the control implemented in the OpenMP runtime. For example, the code snippet in Listing 2.1 specifies a set of N jobs to be executed in parallel.

2.1.2 Fork-join execution model

Under the hood, the compiler would transform the program in the OpenMP parallel constructs to function calls to the OpenMP runtime library that will start multiple threads, execute the code in the construct in parallel with the threads and tear down the threads respectively as shown in Figure 2.2 which illustrates GCC’s implementation of OpenMP [5].
```c
#include "omp.h"

void do_job(int i) {
    ...
}

...

#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < N; i++) {
        do_job(i);
    }
}

...
```

Figure 2.1: An example of specifying a parallel loop with a directive

This is the fork-join model implementing the executions of the OpenMP parallel construct. As a result, the execution of a program with multiple OpenMP parallel construct would look like interleaving parallel and serial regions, as depicted in Figure 2.3. Only the master thread executes the serial region, performing the setup and the summarization work. All the threads would execute the parallel regions concurrently.

As we notice in Figure 2.2, there are two specific functions responsible for creating and tearing down the slave threads for the parallel regions in the runtime library of the GCC implementation of OpenMP, namely `GOMP_parallel_start` and `GOMP_parallel_end`, they and their variants are the key for our trace-collection pintool to identify the transitions between serial regions and the parallel regions, which is crucial for the application execution model we designed for the tiled heterogeneous architecture as described in 3.1.4.
void subfunction (void *data) {
    use data;
    body;
}
setup data;
#pragma omp parallel
{
    subfunction, &data, num_threads);
    body;
    subfunction (&data);
} GOMP_parallel_end ();
(a) Original parallel construct
(b) Transformed code by the compiler

Figure 2.2: Parallel construct implementation

Figure 2.3: The fork-join execution model of OpenMP

2.1.3 Synchronizations

OpenMP also includes the synchronization mechanisms to allow user to control the parallel executions. We will talk about the mostly used one here, the barrier. A barrier is inserted into a parallel construct (the codeblock marked with ‘omp parallel’ pragma) through another pragma, ‘omp barrier’, as shown in Figure 2.4. The barrier inserted would stall the threads running into the barrier, waiting for all the threads to reach the barrier. And then, all the threads would be released to continue executing the parallel construct concurrently.

Similarly, the corresponding function in the GCC’s OpenMP runtime library implementing the barrier functionality, namely, GOMP_barrier, is also under the monitor of the pintool
collecting the traces. The trace collection tool would record each barrier in the trace. And each barrier would be implemented in trace-driven simulations to synchronize the threads running in parallel.

2.2 Memory system designs for existing systems

In this section, we will first introduce the background of the memory system designs for existing computing systems. And then, some basics about the circuit design for caches will be introduced to prepare for the discussions on the cache organizations for high parallelism in Section 3.3.1.

2.2.1 Memory system designs for CPU based systems

The memory system developed for conventional-processor-based homogeneous computing system is normally composed of multilevel hardware caches, memory controller and the main memory as shown in Figure 2.5. With the development of the Moore’s law, the num-
ber of transistors available on a chip was growing exponentially. So in modern CPU-based systems, caches and memory controllers (MMU) are implemented on chip, reducing the communication overhead in terms of both performance and power. And new memory technology also emerged as HBM and HMC taking advantage of three-dimensional integrated circuits to increase the memory bandwidth available for the computing elements.

The multi-level caches are built in the system to bridge the performance gaps between the faster processors and the slower main memory. Caches are usually implemented in the form of static random access memory (SRAM) circuits, which we will introduce in details in 2.2.3. Different levels of the caches are built with different purposes:

- Private L1 data cache and L1 instruction cache are built for both data locality exploitation and fast data/instruction fetch, the sizes of them are limited considering that L1 caches have to keep up with the processor frequency [30]. There is a specific L1 size limit worth mentioning: \( L1\text{-cache}\_size \leq cache\_associativity \times OS\_page\_size \) [30]. This limit ensures that the portion of a request address corresponding to the set ID of the cache is always an in-page address. Thus, no address translation between physical and virtual addresses are needed when determining the cache set id for the request, enabling the concurrent tag array read and TLB lookup. This is the reason why L1D caches in modern processors are mostly 32KB 8-way associative with the knowledge that the pages in most environments are still mostly 4KB despite the emerging huge-page support.
• L2 caches are mostly private to each core respectively to benefit data locality exploitation for applications with working set size larger than L1 cache size. It also reduces bandwidth requirements as well as the power consumption on the shared resources of the network-on-chip and the LLC by filtering the traffics going down the memory hierarchy.

• L3 caches are usually the last-level caches filtering the off-chip traffics to reduce latency, energy consumption and memory bandwidth requirement. With the development of the multi-core scaling, L3 caches are now distributed over the chip in the non-uniform access fashions connected with the mesh interconnect[8]. All the L3 cache slices are shared by all cores, enabling efficient inter-thread data sharing through coherence protocols rather than costly off-chip accesses to the main memory. There is a recent trend seen in Intel’s recent Skylake architectures [12] where L3 caches are made non-inclusive [56] to improve the cache efficiency by saving huge cache capacity of L3 caches from duplicating the contents in L2 caches.

• The main memory for the conventional-processor-based system is usually implemented on a separate chip. Multi-channel Double Data Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM) is still the dominant type of the main memories. But there is also a trend to employ high bandwidth memory (HBM) based with three-dimensional integrated circuits to improve the bandwidth capacity of the main memory and ease the memory wall problem.

2.2.2 Memory system designs for GPU systems

Usage of GPU for general purpose computing rather than graphics has become more and more pervasive, especially with the rise of the recent trend around deep neural networks. The memory system architecture of a GPU system is different from that of CPU because of the
differences in the architecture. GPU is composed of multiple bandwidth-oriented lockstep SIMD engines.

We will be discussing the memory architecture of the Pascal architecture [38] featured in last-generation NVIDIA GPUs like GTX-10/Tesla P series as shown in Figure 2.6. It is composed of a execution context register file in each streaming multiprocessor (SM), a shared memory storage private to each SM, L1 caches private to each SM, L2 caches shared across all the SMs, and the global memory based on GDDR5 or HBM2 technology:

- The execution context register file is the storage for the execution context of each SIMD thread.
- Shared memory is in a separate address space in each streaming processor. It is a software-managed scratch pad enabling efficient inter-thread communication in the same streaming multiprocessor.
- Each L1 cache is private to each streaming multiprocessor. It is designed to explore the spatial locality across the threads running in the same stream processor, with the by-product of exploring the time locality of data accessed in each streaming multiprocessor.
- L2 caches are shared across all the streaming multiprocessors, exploring the spatial and temporal locality across the SIMD threads and reducing the bandwidth requirement and power consumption of the global memory.

It’s worth mentioning that not all applications running on GPU benefit from the L1 and L2 caches. Applications with divergent memory footprints are usually configured to be run
with the cache turned off to avoid performance loss resulted from wasted memory bandwidth by useless over-fetching cache lines with the built-in bypass feature [33]. There are also efforts researching variable cache line size to explore locality and reduce over-fetching at the same time [44].

2.2.3 Cache circuit design

With more and more transistors available on a chip in the 30 years’ development of Moore’s law, all levels of caches are now implemented on the chip to reduce latency, off-chip bandwidth requirement and power consumption. Caches on the chip are mostly implemented with static random access memories (SRAM) with the exceptions of the slow and sparse adoptions [45] of embedded DRAMs as the last level caches to increase storage density. We will focus on SRAM based on-chip caches here.

As shown in Figure 2.7, a cache is composed of two SRAM arrays, one for tags and the other for data. The cache address input would drive the wordline after parsing by the decoder, enabling the gates of the corresponding SRAM cells. In the reading situations, corresponding data and tags stored in the cache would be driving the bit lines so that the tag and cacheline stored can be read out from the bit lines, while in the writing situations, new cacheline and tag will be written into the cells through the bit lines. For simplicity, we will be discussing the data array from now on, the tag array would be similar.

Let us define the cache block size as \( B \) bytes, cache associativity as \( A \), and the number of cache set as \( S \). Please notice that these three variables define a basic cache, with a total size of \( B \cdot A \cdot S \) bytes.

In a naive way, the SRAM implementing the data array of this cache would be of \( S \) rows and \( B \cdot A \) columns, each row storing \( A \) cache lines corresponding to that cache set. However, this naive organization could result in long word and bit lines, which further implies long access and cycle time. To tackle this problem, the whole SRAM array could be evenly divided
into subarrays such that word and bit lines could be shorter and transmitting data faster in each subarray. It is also possible to put multiple cache sets in a row of the SRAM array if the number of sets is much higher than the number of bits in one cache block to make the data array balanced.
CHAPTER 3
EXPERIMENTS

We pursue a set of experiments to gain insights into the impact of tiled acceleration on memory hierarchy design. These experiments depend on a set of tools and experimental methodology, described in Section 3.1. The experiments are organized as follows. First, we explore the requirements an accelerator places on the memory hierarchy in order to achieve the performance improvement which could justify the addition of the accelerator. Then, we examine questions of multi-level cache architecture on the tiled, accelerated chip, exploring organization and coherence. And we also examine the bandwidth requirements on last-level caches and main memory after the introduction of the accelerators.

3.1 Methodology

The methodology section introduces models for the accelerated heterogeneous system with the tiled architecture and the methods for our experiments. Firstly, we will describe the scope of the model for the tiled heterogeneous architecture, specifying the components we are modeling in the tiled heterogeneous architecture. Secondly, we will discuss the performance model of the heterogeneous system under study which illustrates how we are modeling each component in the tiled heterogeneous architecture for performance analyses. Thirdly, we discuss how we model the area and power for each of components in the system. Fourthly, we describe the execution model, which illustrates how each of the components of the modeled heterogeneous system functions when executing the applications. Fifthly, we discuss the workloads, the scalable applications we used in our trace-driven simulation experiments. Sixthly, we discuss the trace simulation framework implementing the performance model and deriving the statistics of the memory hierarchy in the modeled tiled heterogeneous architecture for performance and area/power consumption analyses. Seventhly, we discuss
the design space of the modeled heterogeneous system we explored in the experiments. Eighthly, we introduce the metrics we used to evaluate the modeled system with different configurations explored.

### 3.1.1 Model scope

In this subsection, we specify what we are modeling in the tiled heterogeneous architecture as shown in Figure 3.1a and the tiled homogeneous architecture as shown in Figure 3.1b.

First of all, the model for a tile in the tiled heterogeneous system under study is composed of an accelerator, a conventional core, and the caches either private to or shared between the two computing elements. The model for a chip employing the tiled heterogeneous architecture is composed of multiple tiles and the same number of shared last level caches, following the commodity multi-core processors. We don’t model the functionality of the network-on-chip for its added latency because of our latency-tolerance assumption about the accelerator. However, we do model bandwidth limits and traffic conflicts of the tiles connected to it. The chip implementing the tiled heterogeneous architecture is connected to the main memory chip which is either a multi-channel DDR4 DRAM memory or a multi-stack HBM2 memory. We don’t intend to model the detailed functioning of the memory chips because we are mainly interested in the design of the compute chip with the tiled heterogeneous architecture. We do impose a bandwidth limit for the main memory, considering that it’s the defining characteristics of the main memory chip for the memory hierarchy design inside the computing chip.

The model of the system implementing the baseline, the tiled homogeneous architecture is shown in Figure 3.1b. There are two conventional cores and their private L1 and L2 caches in the baseline homogeneous tile. The cache hierarchy follows the the Skylake-SP microarchitecture [12], the configurations of which are described in Table 3.1. The baseline homogeneous chip is composed of multiple baseline tile and the same number of shared last
level caches. The bandwidth limit of the memory chip is also imposed onto the model of the baseline chip.

### 3.1.2 The performance model

Now we describe how we model each component in the computing system with the tiled heterogeneous architecture for performance statistics. We start with the tile-level performance model, and then the chip-level performance model, for the heterogeneous system depicted in Figure 3.1a. And then we describe the performance model for the baseline system, starting with a homogeneous system tile, and then a homogeneous chip composed of multiple baseline tiles and other shared resources.
There are two computing elements in each core-accelerator tile, a conventional core and an accelerator, the internal architecture of which are the subject of much research and design. However, their defining characteristics for the memory hierarchy is the issue rate and access patterns. Therefore, we model both of them as a memory reference engine at the full rate of execution. We estimate the execution rates of the core through the typical ratio of the memory reference uops to all the uops (1:5) and the function unit components in Skylake-SP microarchitecture [12]. Since there are four integer ALU, 2 load units and 1 store units, our best estimation is that the a skylake core on average can execute 4 computation uops and 1 data reference uop per cycle, resulting in a memory reference issue rate 1 reference per cycle. The accelerator is assumed to have higher speed than that of a core, issuing N references per cycle, the exact range we considered are 2 to 32, the upper bound of which is limited by the area and power cost and the diminishing returns when scaling up the accelerator. This performance model of the computing elements would provide statistics of cycle counts used in performance comparisons and bandwidth calculations. One thing to notice is that we didn’t model the cache system feedback stalling the core or accelerator issuing the request, but we did model bank conflicts where the accelerator are unable to issue references at the maximum rate possible. In general, it will be an aggressive estimate on the load of the memory hierarchy, so we may slightly overestimate the challenges for the memory hierarchy performance. But by tackling the overestimated challenges, we in turn show the effective of our methods for mitigating the memory wall problem.

In our model, the core and the accelerator in a core-accelerator tile each has its private cache hierarchy respectively. And there are last-level caches shared by all the tiles at the chip-level. Considering the memory hierarchy is the main subject of the study, the cache performance model we extended from the cache models in ZSim [46] cover almost all details about the cache architecture, including the modeling of the size, associativity, replacement policy, hashing algorithms for multi-slice caches, MSHRs, inclusiveness/noninclusiveness,
and coherence policy. This cache performance model would provide statistics like accesses, misses, and coherence operations, enabling performance analyses on the locality exploitations and the characteristics of the traffic going through different level of the memory system. Please notice that the cache organization shown in the tile of the Figure 3.1 is just a sample organization. Both the organization and the configuration of the cache hierarchy of the core-accelerator tile is the core subject of the study.

The computing chip implementing the tiled heterogeneous architecture is composed of multiple core-accelerator tiles and the same number of shared last level cache slices. For the last-level caches, they are modeled in the same way as the caches in each tile with the extended ZSim cache model. The number of the tiles and the last level cache configurations are varied in the experiments because they are critical to the memory bandwidth mismatch challenges. We don’t model the functionality of the network-on-chip because it mainly affects the latency to serve a request, which is ignored considering the assumption of our accelerator on latency tolerance. We only model the constraint from the shared last level cache slices and network-on-chip by applying a per-tile performance-throttling model based on the tile-level LLC bandwidth requirement to each tile (both the core-accelerator tile and the baseline tile). There were two situations considered, as shown in Figure 3.2. The first situation (Figure 3.2a and 3.2b) is that the tile simply stalls when shared LLC can’t keep up with the tile’s reference bandwidth issuing to the LLC. And the tile speedup will thus stop to increase, when the required LLC bandwidth is beyond the threshold. We name this situation ”SF=0” and will use it in the related figures. 'SF' here actually stands for 'scaling factor', meaning how much the LLC can be scaled to adapt to the excessive bandwidth. The second situation (Figure 3.2c and 3.2d) is that the chip designer can fully scale up the LLC bandwidth capacity at the cost of increase in both area and power estimated using cacti [37], which was named ”SF=1”. The LLC bandwidth capacity ($LLCBW_0$ in Figure 3.2) is fixed at 100 GB/s. It’s an estimate considering the 32 Bytes/cycle interface between cache/home agent and the mesh network.
Please notice that we are overestimating the LLC bandwidth capacity, and downplay the bottlenecks resulted from the limited LLC bandwidth capacity. But we will notice in the experiments results that tile performance and system performance are still heavily limited by this overestimated LLC bandwidth capacity. We don’t model the memory controller, neglecting the effects of coalescing and reordering there and assuming all the traffics coming out of the shared last level cache would result in the same off-chip memory traffics. The main reason is that the memory controller is highly related to the technology the memory chip build upon, while we would like to maintain the general applicability of our work.

Furthermore, as described in Section 3.1.1, although we don’t model the detailed functioning of the memory chip, because we mainly focus on the computing chip design implementing the tiled heterogeneous architecture, we impose the bandwidth constraint of the memory chip onto the performance model of the computing chip. The effect of this potential bottleneck on the performance is modeled by applying a simple performance throttling mechanism that
dials down the performance (i.e. periodically stalling the memory reference issuing of the computing elements) if the memory bandwidth requirement of a multi-tile chip is higher than the upper bound we set. We will consider two different upper bounds, 100 GB/s modeling a multi-channel DDR4 memory system [2], and 1000 GB/s modeling a 4-stack HBM2 system since these two settings are the typical memory technologies available.

Now that we finish the introduction of performance model for the tiled heterogeneous system, we will discuss the performance model for the corresponding homogeneous system. We will start with the homogeneous tile and then the computing chip implementing the tiled homogeneous architecture.

The homogeneous tile which is also the baseline tile we compare the core-accelerator tile with includes two conventional cores and their corresponding cache system. Each conventional core is modeled the same way as the conventional core in the core-accelerator tile: a reference issuing engine with reference issue rate 1 reference per cycle which would determine the cycle consumed for a specific job. The performance model of the caches are also the same extended cache model from ZSim [46] as that of the core-accelerator tile. For the baseline tile, the cache organization is fixed to be private L1 and L2 caches for each core with configurations showing in Table 3.1 which is modeled after the Skylake-SP microarchitecture [12].

The homogeneous chip is composed of multiple homogeneous tiles and the shared resources like last level cache and network-on-chip. Just like the chip implementing the tiled heterogeneous architecture, the last level cache is modeled with the same cache performance model extended from ZSim[46]. And the network-on-chip, together with the LLC, imposes a LLC bandwidth upper bound for each tile at 100 GB/s. Similarly, the memory chip connected with the homogeneous computing chip imposes a 100 GB/s (DDR4) or 1000 GB/s (HBM2) memory bandwidth upper bound for a computing chip.
3.1.3 The area and power model

In this subsection, we discuss how we model the area and the power for different parts of the modeled heterogeneous system: cores, accelerators and caches.

For the conventional core, aiming to derive the area and the power for the logic part, we surveyed the die shot for a Skylake-SP core. By calculating the ratio of the logic part of one core in the die area, we can project logic area of a conventional core from the die area of 694 mm² [13]. Following a similar procedure and taking the power density differences between SRAMs and logics into considerations, we can get the power for the logic part of the core as well, as shown in Table 3.2. Please notice that since area and power for caches are modeled with cacti separately in our design exploration, the numbers we use for the core area and power are the numbers in column ‘core logic’ in Table 3.2 which, please also notice, doesn’t account for the AVX logic.

In terms of the accelerator in a core accelerator tile, we estimated its relative area efficiency and power efficiency compared to a conventional core by surveying a range of commodity accelerators in terms of FP32TFLOPS, area, and power. Supported by the data in Table 3.3, we assume the hypothetical accelerator is 8X area efficient and 16X energy efficient.
than a conventional core. As we scale the accelerator speed, we calculated the corresponding power and area of the accelerator as following:

\[
\text{Accelerator Area} = \text{Core Area} \cdot \frac{\text{Relative Accelerator Speed}}{\text{Relative Area Efficiency}}
\]

\[
\text{Accelerator Power} = \text{Core Power} \cdot \frac{\text{Relative Accelerator Speed}}{\text{Relative Power Efficiency}}
\]

For caches, we employ cacti[37] to model their area and power. We also adapt cacti to model work-interleaving multi-banking caches where the cache blocks are split into words and each bank stores one word of a cache block, which is further discussed in 3.3.1. Please notice, despite the fact that configurations of the last-level-cache are not included in the tile-level design, we will take LLC access energy into accounts using the access energy of the last-level cache configuration in baseline tile in Table 3.1, to reflect the effects of tile-level cache configurations on off-tile access power during the tile-level design exploration. In chip-level design exploration, both static power and the dynamic power of the corresponding last-level cache configuration are normally counted into the total power number.

### 3.1.4 The application execution model

In this section, we describe how we map the applications onto the model of the tiled heterogeneous architecture. We will start with the mapping for the model of a baseline tile, and then the mapping for the model of a core-accelerator tile and finally the model of a chip implementing the tiled heterogeneous architecture.

For the baseline tile, the application execution model follows the traditional OpenMP fork-join behavior as shown in Figure 3.3b. The core running the master thread will be executing a serial region at first where it does some setups and starts all other threads. Then, the parallel region will be executed by all the cores, which is essentially a SPMD execution model. When the parallel region is finished, all threads running on each core join
at a barrier. The core running the master thread will then summarize the results and may start a new fork-join cycle in the same serial region.

In terms of the core-accelerator tile, the accelerator will be assigned to execute the parallel regions on the assumption that the accelerator delivers more performance or energy efficiency, while the core will execute the serial region for the setups and the summarizations, considering that the work may be branch-intensive and thus suitable for the conventional cores. The way we map the serial region and parallel region onto the accelerator-core tile in Figure 3.3a reflects this the execution model we discussed here.

At the chip level, the chip implementing the tiled heterogeneous architecture is assumed to explore the task-parallelism across multiple tiles, so each tile is running its own application process. We don’t consider shared memory applications with multiple threads running across multiple tiles.
### 3.1.5 Workloads

The workloads used for this study are Exascale Computing Project (ECP) proxy apps [4] as listed in Table 3.4. These are scientific computing applications featuring workloads like finite element methods, binary searches, sparse matrix multiplications. They are scalable, highly parallel and often memory system bounded. The OpenMP versions of these applications compose the workload of this study to facilitate the application execution model described in Section 3.1.4.

We chose this application set as our workload out of two reasons. First, the scientific algorithms are highly parallel, and thus feasible to be accelerated by the accelerator. Second, the memory intensive applications in them would create huge design challenges for the memory hierarchy. By resolving these hard challenges, we intend to argue the effectiveness of the methods we proposed.

### 3.1.6 Trace-driven simulation

In this subsection, we discuss the trace-driven simulation framework we built to implement the performance model described in Section 3.1.2 and the application execution model described in Section 3.1.4 to collect different statistics of the modeled system.

The memory reference address traces of the workloads described in Section 3.1.5 are

<table>
<thead>
<tr>
<th>App Names</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG</td>
<td>Parallel algebraic multigrid solver</td>
</tr>
<tr>
<td>XSBench</td>
<td>Monte Carlo neutronics</td>
</tr>
<tr>
<td>HPCCG</td>
<td>Conjugate gradient</td>
</tr>
<tr>
<td>SWFFT</td>
<td>Discrete fast Fourier transform</td>
</tr>
<tr>
<td>CoMD</td>
<td>Molecular dynamics</td>
</tr>
<tr>
<td>miniFE</td>
<td>Unstructured implicit finite element codes</td>
</tr>
<tr>
<td>miniTri</td>
<td>Triangle based data analytics</td>
</tr>
<tr>
<td>ExaMiniMD</td>
<td>Particular molecular Dynamics</td>
</tr>
<tr>
<td>Sw4lite</td>
<td>3D seismic modeling</td>
</tr>
</tbody>
</table>

Table 3.4: ECP proxy apps and their descriptions
collected through a Intel Pin [11] tool during the normal run on a homogeneous Xeon Server. The identification of the OpenMP serial and parallel regions is carried out at the same time through the pintool by tracking the calls of functions in OpenMP library that mark the beginnings and the ends of the parallel regions. Figure 3.4 shows how the trace looks like. It is composed of interleaving serial regions and parallel regions where traces entries in parallel regions are coming from multiple running threads. In the trace-driven simulations of the core-accelerator tile, considering our application execution model described in Section 3.1.4, the traces from the serial regions are issued to the core-side memory hierarchy by the reference issuing engine modeling the conventional core, while the traces from the parallel regions are issued to the accelerator-side memory hierarchy by the reference issuing engine modeling the accelerator. We don’t replay the instruction traces under the assumption that the branch prediction and instruction prefetch is perfect because we observe very low instruction misses on early experiments. Taking out the consideration of the instruction trace and L1 instruction caches not only shrinks the design space, but also enables us faster design space explorations with the minimal cost on accuracy.

The trace-simulation tool described above is integrated with the ZSim [46] cache performance model. Furthermore, a ZMQ [15] based multi-process online simulation infrastructure is developed. We use ZMQ as the transport to stream the traces from the trace collection pintool to the simulation processes. As shown in Figure 3.5, multiple processes each simu-
Figure 3.5: Online trace-driven simulation based on ZMQ and ZSim

lating one variant of the memory hierarchy of system implementing the tiled heterogeneous architecture can thus share the same trace to avoid the effects of variances on traces from multiple runs.

3.1.7 Metrics

This section describes the metrics we used to evaluate the different tile-level and chip-level designs while performing the design space exploration. We will start by describing the metrics used for tile-level designs and then extend them for the chip-level designs.

In the design space explorations of the tile design, we define three metrics to evaluate the tiles with different configurations:

- Tile speedup: Ratio of the execution time on the baseline to the execution time the accelerated tile for the same workload.

- Normalized area efficiency: Ratio of the tile speedup to the normalized area.

- Normalized power efficiency: Ratio of the tile speedup to the normalized power.

The normalized area is the ratio of the area consumption of the core-accelerator tile to that of two cores. The normalized power is the ratio of the power consumption of the core-accelerator tile to that of two cores. We normalize to two cores instead of the baseline tile.
because core area and power is presumably a better-known comparison baseline. The three metrics are chosen to shed a light on the tradeoff between the performance gain with the area and power costs when employing and scaling the accelerator.

We further extend these three metrics for the design space exploration of the chip design:

- Normalized chip performance: The ratio of chip performance to that of a baseline tile.
- Normalized area efficiency: The ratio of 'Normalized Performance' to normalized chip area.
- Normalized power efficiency: The ratio of 'Normalized Performance' to normalized chip power.

Please notice that the normalized chip area/power is still normalized to that of two cores.

3.2 Memory Hierarchy Challenges for Tiled Heterogeneous Architectures

In this section, we analyze the memory system performance characteristics, specifically L1 access rates, coherence traffics, LLC and memory bandwidth requirements of a core-accelerator tile collected from trace-driven simulations, to identify design challenges for memory hierarchies in tiled heterogeneous architectures.

3.2.1 L1 data cache Access Rate Requirements

Given our accelerator model that generates memory references at an increased rate (in proportion to the accelerator speed), the L1 data cache, which typically services one access per cycle, will limit the accelerator performance. In general, an accelerator $S$ times faster than the core has to be served with a cache fulfilling $S$ requests per cycle to deliver its full benefits.
Figure 3.6: Tile speedup with variant speed and #L1D banks

We explored the L1D cache organizations, specifically multi-banking, to deliver the required access rate.

This experiment varies bank parallelism of the accelerator-side L1 data cache (L1DB) and the accelerator speed (S) which leads to varied accelerator reference issue rate and observes the variances of the performance measured in tile speedup metric of the core-accelerator tile when running ECP workloads. The core-accelerator tile is configured with the cache configurations same as the baseline tile as shown in Table 3.1.

Figure 3.6 presents the experiment results from our trace-driven simulation. It shows the tile speedup along the y-axis while running each ECP application along the x-axis with different accelerator speed (S) and L1 data cache bank parallelism (L1DB) configurations as shown in the legend. First of all, we can see from the figure that the number of banks should be at least the relative accelerator speed to not become the bottleneck. But it should also be noticed that higher number of banks than the accelerator issue rates can improve
the accelerator performance through reducing bank conflicts. For example, a 32-bank L1D cache can provide around 25% performance improvement over a 8-bank L1D cache when the relative speed of the accelerator is 8. SWFFT, as an exception, doesn’t get fully parallelized into the OpenMP parallel constructs, and thus didn’t get as significant performance increase in all accelerator speed and L1 data cache configurations.

As shown in Figure 3.6, it is necessary to increase the L1 data cache bank-parallelism of the accelerator-side cache hierarchy to actually deliver the benefits of the scaled up accelerator. The performance increase when employing L1 data caches with more banks mainly results from concurrent serving of the requests via the bank-level parallelism of the caches and the reductions on bank conflicts of the concurrent requests issued by the engine modeling the accelerator. However, we observe diminishing returns when either increasing the number of banks in L1D cache or increasing the speed of the accelerators. Please recall that in our area/power model for the accelerator, the area and power consumptions grows linearly with the scaling of the accelerator speed. So there would be a trade-off between area/power efficiency and performance when scaling up the accelerators and its immediate caches during the design space explorations which we would discuss later.

3.2.2 Coherence traffic in a tile

Our tiled heterogeneous architecture features tight coupling between a core and an accelerator. High single-thread performance of the core and high parallel performance of the accelerator are combined to address the both part of the limitations in Amdahl’s law [31] in speeding applications. However, the performance asymmetry between the core and the accelerator and the interleaving coroutine-like execution model of the tiled heterogeneous architecture establish a reasonable doubt: Is there excessive traffics moving data between the core and the accelerator to maintain the coherency between this two computing elements?

We conduct experiments of a baseline tile and a core-accelerator tile both with baseline
Figure 3.7 shows the coherence operation per kilo references along the y-axis when two tile as described in legends are running each ECP application along the x-axis. The light color part in each bar corresponds to the cross invalidations and the dark color part in each bar corresponds to the cross downgrades.

As we can observe, the coherence traffic between the core and the accelerator in the core-accelerator tile is actually lower than that between the two cores in the homogeneous baseline tile. The reason is that, for ECP applications except SWFFT, OpenMP parallel loops essentially handle all the computations, and thus the core in the core-accelerator tile is not touching the data from the computation part. On the other hand, in the baseline tile, the execution of the parallel loops handling all the computations is split between two cores, creating interferences and coherence traffic between the two cores because of the lack of iteration-level affinity supports. Thus, we can conclude that the coherence traffic is unlikely an issue in the tiled heterogeneous architecture with our OpenMP based coroutine-like application execution model described in Section 3.1.4.
3.2.3 LLC and memory bandwidth requirement per tile

The integration of the accelerator introduces a computing element with higher execution and reference issuing rates. So it’s worthy to look into the bandwidth requirement posed by each tile on the performance-critical chip-level shared resources like last-level caches and main memory to see if the increase on data reference rate makes the shared resources bottleneck(s) in the system.

We conduct experiments to measure the LLC and memory bandwidth requirements of a core-accelerator tile when executing the ECP workloads through trace-driven simulations. The core-accelerator tile is with cache configurations as described in Table 3.1. We varied the accelerator speed (S) and keep the bank-level parallelism of the accelerator-side L1 data cache (L1DB) four times of the accelerator speed for minimal bank conflicts to investigate the bandwidth requirements of tiles with accelerators of different speeds.

The experiment results are shown in Figure 3.8 with two subfigures corresponding to LLC bandwidth requirements and memory bandwidth requirements respectively. The bandwidth requirement for the core-accelerator tile with different accelerator speed and L1 data cache bank-level parallelism running each ECP application along the x-axis is shown along the y-axis. The LLC bandwidth and memory bandwidth requirements vary across the applications. But for memory intensive applications like AMG, HPCCG, and SWFFT, the LLC bandwidth and memory bandwidth for just one core-accelerator tile can easily reach 100 GB/s and 80 GB/s respectively under only moderate acceleration, putting great pressure on shared chip resource like interconnects and memory bandwidth and making them easily a bottleneck in the system when scaling up and scaling out the tile.

3.2.4 LLC bandwidth imbalance from a tile

We also noticed the heavy imbalance in terms of the LLC bandwidth requirement between the core and the accelerator in a core-accelerator tile from the experiment results of the
bandwidth requirements on the last-level caches of the previous subsection. It results from
the tight coupling and performance asymmetry between the core and the accelerator in each
tile of our tiled heterogeneous architecture.

Figure 3.9 shows the bandwidth requirement along the y-axis from the core and the
accelerator separately in a accelerator-core tile with the configuration of an 8-issue-per-cycle
accelerator and a 32-bank L1 data cache running different ECP workloads along the x-axis.

As shown in Figure 3.9, for applications that can expose abundant parallelism into the
OpenMP parallel regions to be accelerated, the average LLC bandwidth requirement from
the accelerator-side is significantly higher than that of the core-side. SWFFT doesn’t explore
the full power of the parallelism, leaving a huge amount of work in the serial region executed
Figure 3.9: Average LLC bandwidth requirement from the core and the accelerator respectively

only by the conventional core, resulting in a higher core-side LLC bandwidth requirement. miniTri has a very good hit rate on the acc-side L2 but a bad hit rate at the core-side L2, resulting in higher core-side LLC bandwidth requirement.

These irregular imbalances bring unwanted complexities to the design of network-on-chip. It’s hard to make a static decision on how much LLC bandwidth is needed for the L2 cache at each side, as the relationship of the two LLC bandwidth requirements varies hugely across the applications. And the situation gets even extreme if we consider the temporal bandwidth requirement, the coroutine-like interleaving application execution model essentially makes the temporal LLC bandwidth requirement from the idle-side to be zero.

### 3.3 Evaluations on proposed cache system improvements

In this section, we explore two improvements on the cache system trying to tackle the challenges identified in the previous section. The first one, trying to resolve the bandwidth mismatch challenges between the accelerator and the L1 data caches as described Section
searches for the L1 data cache bank organization that actually sustains the high request bandwidth issued from the accelerator with minimal performance, area, power overhead and maintain the original level of locality exploitation. The second one, aiming to resolve the problems on LLC bandwidth requirements as described in Section 3.2.4, explores L2 organizations to eliminate LLC bandwidth imbalance and effectively pool the L2 cache capacity as well as LLC bandwidth capacity between the core and the accelerator.

Heavily coupled with the scaling up of the accelerator in each tile, the other two challenges of LLC and memory bandwidth bottleneck will be tackled later in the tile-level and chip-level design space explorations where the accelerator sizing will be explored at the same time with memory hierarchy design.

3.3.1 Associativity makes way for multi-banking L1 data caches

As we have seen in Section 3.2.1, multi-banking is required for L1 data caches to deliver the accelerator benefits. There are two forms of multi-banking as shown in Figure 3.10:

- Word-interleaving multibanking: The words in each cache line can be made to be distributed in different cache banks. Considering that the width of the data requests is usually 8 bytes, this technique can support up to 8-way parallelism given that the cache line size is as usual 64 bytes.

- Set-interleaving multibanking: The sets can be made to interleavingly distribute in different cache banks. The parallelism exploited from this kind of multi-banking is virtually not bounded until it divides the number of rows of the SRAM array to be too few to implement practically as discussed in Section 2.2.3.

We considered a hybrid multibanking scheme as shown in Figure 3.10 for L1 caches which employs both word-interleaving multibanking and set-interleaving multibanking to improve cache parallelism. If L1 cache requires less than 8-way parallelism, only the word-interleaving
multi-banking is considered. But when the parallelism required is more than 8-way, the set-inteleaving multibanking will be exploited to provide the extra bank-level parallelism needed. Word-interleaving multi-banking is preferred because it doesn’t further require the bandwidth capacity of L2 to be increased, while the set-interleaving multibanking does. And as we discussed in Section 3.1.3, Cacti [37] which only supports the set-interleaving multi-banking natively is tailored to model the caches with the hybrid multi-banking simply by getting area and power cost for one cache bank and scaling it linearly to be the cost of a multi-bank cache for the word-interleaving multi-banking.

On the other hand, we have discovered that traditional high associativity L1 caches would act against increasing serving bandwidth through mutli-banking. As we have discussed in Section 2.2.3, the data array or the tag array of the cache would be implemented in either one or multiple SRAM subarrays. For a cache with associativity $A$, cache block size $B$ and number of cache set $S$. If the cache size $C = A \cdot B \cdot S$ is fixed, high associativity $A$ would
result in a low number of sets $S$. The straightforward one-bank implementation of this high
associativity cache already result in a height $S$ too low to be efficiently implemented, there
is no room for the set-interleaving multibanking to further divide the caches into multiple
subarrays through interleaving the sets which would result in a extremely low height for each
SRAM subarray.

To enable high bank parallelism of L1 data cache, we propose to use high capacity low
associativity L1 caches, which, as shown in Table 3.5, don’t lose on capturing locality because
of higher capacity, run at lower latency and shorter cycle, and make way for employing set-
interleaving multi-banking for L1 caches requiring high bank-parallelism. The only catch is
increased static power and area consumption, which is still less than a hundredth of that of a conventional core and can thus be tolerated. Therefore, the L1 data caches we used
for design space exploration described later are all 256KB direct map caches with varied
bank-level parallelism. L1 cache size and associativity are not varied in the later design
space explorations because they are critical to the timing of the computing elements like the
conventional core or the accelerator.

### 3.3.2 Tile-level-shared L2 cache to eliminate LLC bandwidth imbalances

To eliminate the adverse effects arose from the imbalance on LLC bandwidth requirements
as described in Section 3.2.4, we propose to combine the core-side and the acc-side L2
cache, i.e. employ one L2 cache shared by the accelerator and the core in a core-accelerator

<table>
<thead>
<tr>
<th></th>
<th>Miss rate as L1D for the accelerator</th>
<th>Cycle time (ns)</th>
<th>Access time (ns)</th>
<th>Access energy (nJ)</th>
<th>Leakage power (mW)</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32KB Assoc=8</td>
<td>3.27%</td>
<td>3.46093</td>
<td>1.34201</td>
<td>0.0808</td>
<td>0.9490</td>
<td>0.01315</td>
</tr>
<tr>
<td>256KB Assoc=1</td>
<td>3.00%</td>
<td>0.6392</td>
<td>0.7106</td>
<td>0.0388</td>
<td>3.3364</td>
<td>0.07802</td>
</tr>
</tbody>
</table>

Table 3.5: Comparison table between a 32KB 8-way associative cache and a 256KB direct
mapped cache
Figure 3.11: Sharing a single L2 cache in each tile

![Diagram showing core and accelerator with L1 and L2 caches]

Figure 3.12: Global L2 miss rate before and after employing a single tile-level shared L2 cache

![Bar chart comparing separate and fused L2 caches across different workloads]

tile as demonstrated in Figure 3.11. This way the bandwidth requirement imbalances are eliminated because there will only be one network-on-chip node in each core-accelerator tile sending and receiving traffics from and to the last-level caches. And the L2 cache capacity as well as the LLC bandwidth capacity are effectively pooled between the core and the accelerator, enabling better locality exploitation. Moreover, if by any chance the core and the accelerator is sharing data, they would be able to achieve that via the shared L2 cache rather than served by last-level caches, reducing both latency and power.

We performed trace-driven simulations to evaluate this proposal. The cache system of the core-accelerator tile is configured same as the baseline tile as discussed in Table 3.1.
For the system with tile-level-shared L2, instead of having two L2 with size of 512KB each private to the core or the accelerator respectively, there is only one L2 of 1MB sharing by both the accelerator and the core.

Figure 3.12 presents the global L2 miss rate along the y-axis for each application along the axis. As we can see, combining the core-side and the accelerator-side L2 cache together and sharing it by the two computing elements reduces the global L2 miss rates and thus LLC bandwidth requirements because of the effective pooling of the capacity of L2 cache. In conclusion, we employed the tile-level-shared L2 organizations in the rest of the experiments.

### 3.4 Summary

In this chapter, we introduced our experiment methodology which include the performance, application execution, area and power model for the tiled heterogeneous architecture and the trace-driven simulation framework implementing these models. We employ this methodology to perform experiments identifying the design challenges at each level of the memory hierarchy, namely, the private cache design challenge to serve the high-performance accelerator, the L2 cache design challenge to accommodate the balance requirement discrepancy between the core and accelerator and the bandwidth challenge at LLC and main memory. We propose and evaluate the design suggestions for them. Specifically, the design insights distilled from the studies of this chapter are:

- Employ low-associativity high-capacity high-bank-parallelism L1 data cache for the accelerator to sustain the high request rates from the accelerator and deliver the accelerator benefits while maintaining the locality exploitation quality.

- Employ a single L2 cache shared between the core and the accelerator enables effective pooling of L2 cache capacity and the LLC bandwidth capacity between the core and the accelerator, improving the locality exploitations as well as mitigating the LLC
bandwidth bottleneck.
CHAPTER 4
DESIGN SPACE EXPLORATIONS

In this chapter, we continue to distill design insights about the memory hierarchy designs for the tiled heterogeneous architecture through design space explorations first at tile-level and then at chip-level. We took the accelerator sizing into considerations at the same time to explore the balance between scaling up and scaling out and look for ways to mitigate bandwidth bottlenecks under varied acceleration scenarios.

4.1 Design Space Explored

This section describes the design space we explored for both the core-accelerator tile and the chip implementing the tiled heterogeneous architecture. We are primarily interested in the memory system characteristics after the introduction of the accelerator(s), so the design space we are exploring are mainly about four entities in the model: the accelerator, L1 data caches, L2 caches, shared LLC caches. And we will discuss the configurations explored for each of them as shown in Figure 4.1 under the resource constraints. Please notice that we are not considering L1 instruction caches because the decision of not replaying the instruction traces as described in Section 3.1.6.

Firstly, we scaled the accelerator speed which will result in varied accelerator issue rates

![Figure 4.1: Design space explored](image-url)
at the cost of the area and power as described in Section 3.1.3. It not only enabled us to investigate the balance of resources allocations between the core and the accelerator but also enabled the investigations of the memory system characteristics under the impacts of the accelerator of different speeds. Due to the resource constraints and diminishing returns of scaling, the accelerator speeds we considered are 2, 4, 8, 12, 16, 24, 32 times faster than a core, which correspond to a issue rate of 2, 4, 8, 12, 16, 24, 32 references per cycle respectively.

Secondly, for L1 data caches, a low-associativity high-capacity high-bank-parallelism organization is proposed in Section 3.3.1 to be able to sustain the high request rate from the accelerator, and we use this organization for the accelerator-side L1 data cache throughout the study. The only parameter for the L1 data cache we varied is the bank-level parallelism to facilitate the need of the accelerator with varied performance. The bank level parallelism we considered are 2, 4, 8, 16, 32, 64, 128, the upper bound of which is limited by the exponentially growing cost of the switch network routing the requests to each bank.

Thirdly, for L2 caches, we explored its size and associativity configurations to guide the trade-off between the costs in area and power of L2 caches and its benefits in off-tile traffic reduction. The sizes we considered are 0(no L2), 512KB, 1MB, 2MB 4MB, 8MB, 16MB and the associativity we considered are 1, 2, 4, 8, 16 the upper bound of which are both limited by the diminishing returns and resource constraints. Furthermore, we explored the L2 caches organization on whether it should be shared between the core and the accelerator as described in Section 3.3.2.

Fourthly, for LLC caches which will be explored in chip-level design, the fixed assumptions are that it is shared across all the tiles in the chip and it is noninclusive as discussed in [56] to improve cache capacity efficiency. And there are the same number of LLC cache slices as the number of tiles. We explored its configuration in terms of the per-tile-sizes (1MB 2MB, 4MB, 8MB, 16MB). This explorations could guide the tradeoff between the costs of area and power of LLC slices and its benefits on off-chip traffic reduction which in turn helps on the
4.2 Designing the core-accelerator tile

In this section, we aim to tackle the LLC bandwidth bottleneck as identified in Section ?? and further distill understanding and insights on memory hierarchy design for the tiled heterogeneous architecture by exploring the tile-level design space as described in Section 4.1.

There are four metrics fighting against each other here, namely, tile speedup, tile LLC bandwidth requirement, tile area, and tile power. When one tries to increase the tile speedup by scaling up the accelerator speed, the LLC bandwidth requirement, area and power are all going to increase, signaling the increase on resource consumptions. When one tries to increase L2 size to mitigate both LLC bandwidth and memory bandwidth bottleneck and thus improve the performance, additional cache area and static power cost is immediate, though the effects on dynamic power is uncertain.

To shed a light on this entangled web, we will first try to understand how these metrics change based on the performance statistics from the trace-driven simulations when we only vary one element in the tile at a time. We will start by scaling up the accelerator speed (modeled as varied issue rates) only and then vary the configurations of the tile-level-shared L2 cache in terms of sizes and associativity only. The starting configuration of a tile includes a core with issue rate 1 reference per cycle, a 16x faster accelerator, a direct mapped, 64-bank, 256KB L1 data cache for the accelerator, a 32KB 8-way-associative L1 data cache for the core, a 512KB 16-way-associative tile-level-shared L2 cache.

4.2.1 Scaling up the accelerator speed

In this subsection, we conduct experiments scaling up accelerator speed which is modeled as varied accelerator issue rates from the starting configuration to be 2, 4, 8, 16, 32 references
per cycle when running trace-driven simulation on the aforementioned ECP workloads. The number of banks at the acc-side L1 data cache is always kept at four times of the accelerator issue rate to ensure minimal bank conflicts. We considered only the "SF=0" situation for the performance throttling resulted from the excessive per-tile LLC bandwidth requirement as described in Section 3.1.2 where the execution speed (memory reference issuing rate) of a computing element (either a core or an accelerator) is essentially dialed down if the LLC bandwidth requirement appears to reach the upper bound.

Figure 4.2 presents the experiment results. The three subfigures show the ‘Tile speedup’, ‘Normalized area efficiency’ and ‘Normalized power efficiency’ defined in Section 3.1.7 respectively along the y-axis while scaling the accelerator speed along the x-axis in the log scale. Please notice that the first figure has a log-scale y-axis, while the other two don’t. And the metrics are all calculated as the geometric mean of the corresponding data across all the ECP applications. Besides the tile speedup, area efficiency and power efficiency coming out from our model in each figure, we also project the corresponding data when there is no LLC-throttling (Non-throttled) and when there is no LLC-throttling and applications can be fully and abundantly parallelized to be explored by the accelerator (Ideal). The two projections help us understand the factors limiting the performance, area efficiency and power
efficiency.

In Figure 4.2a, first of all, we can see that tile speedup measured for the core-accelerator tile with varied accelerator speed are all larger than 1, suggesting the introduction of the heterogeneity has improved the tile performance comparing to a baseline tile. Secondly, the tile speedup increase as the accelerator speed increase, but the speedup increase are diminishing at high accelerator speeds. Comparing to the two ideal situations we projected, it is clear that the performance of a tile is suffering from the fact that ECP applications are not abundantly parallelized to be explored by the accelerator of high parallelism. This inadequacy of the parallelization gets worse when the accelerator speed scales up. Moreover, when the accelerator speed goes up to more than 8 references per cycle, the overwhelming LLC bandwidth requirements make last-level caches the performance bottleneck of the tile, considering that tile performance is clearly falling behind the performance projection in Non-throttled situation as shown in the figure.

Changing the subject to Figure 4.2b and 4.2c about normalized area efficiency and power efficiency. Firstly, the fact that the normalized area and power efficiency of the designs in this experiment are both higher than one suggests that the introduction of the heterogeneity into the system actually improves the system efficiency on area and power usage because of the use of the accelerator which is more efficient than the core. Secondly, we observe the increase at first and then decrease on both efficiency, creating a peak for normalized area efficiency and normalized power efficiency respectively when scaling up the accelerator speed. The increase at first is coming from devoting more area and power resources to the accelerators which are of higher efficiency. But as the accelerator speed continues to grow, the and the system hit the shared resource (last-level cache bandwidth) bottleneck. Thus, the sublinear performance increase can’t justify the linear cost increase when further scaling up the accelerator, which results in the decrease in both area efficiency and power efficiency. Thirdly, because the normalized area efficiency and normalized power efficiency are going
to decrease below 1 if continuing increasing the accelerator speed (modeled as issue rate) higher than 32, it probably doesn’t make sense to build extremely scaled-up accelerators into the core-accelerator tile because it wouldn’t work as efficient as a baseline tile, which, please recall, is composed of two conventional cores.

The takeaways from this set of experiments are two-fold:

- Scaling up accelerator speed is necessary to better allocate resources towards the more efficient accelerator.

- The bottlenecks limiting the tile speedup and result in the reductions of area and power efficiency when scaling-up the tile are the incompetence of full and adequate parallelization of the applications and the limited last-level cache bandwidth capacity.

### 4.2.2 Varying L2 cache size and associativity

In this subsection, we conduct experiments varying only the configurations of the tile-level-shared L2 cache in terms of the size (512KB, 1MB, 2MB, 4MB) and the associativity (1, 2, 4, 8, 16, 32) from the starting configuration. We considered only the ”SF=0” situation for the performance throttling resulted from the excessive per-tile LLC bandwidth requirement as described in Section 3.1.2 where the execution speed (memory reference issuing rate) of a computing element (either a core or an accelerator) is essentially dialed down if the LLC bandwidth requirement appears to reach the upper bound.

Figure 4.3 presents the experiment results when we only vary the configurations of the tile-level-shared L2 cache in the core-accelerator tile in terms of the size and the associativity from the starting configuration. The first three subfigures show the tile speedup, tile LLC bandwidth requirement, tile power consumption decomposition along the y-axis with different L2 configurations. Specifically, in Figure 4.3c, each stacked bar shows the network-on-chip access power, L2 cache static power, L2 cache dynamic power from the bottom to the top for each L2 cache configuration.
In Figure 4.3a, we can see that tile speedup would increase when employing larger size tile-level-shared L2. This is because that larger L2 cache mitigates the LLC bandwidth bottleneck, reducing the throttling effects for the memory intensive applications, which is confirmed by the LLC bandwidth requirement reductions shown in Figure 4.3b. Enlarging the L2 cache from 512KB to 4MB would result in a 18% reduction on LLC bandwidth requirement, which further leads to a 15% improvement on tile performance. Figure 4.3c suggests that a larger L2 cache actually saves the total power consumption of a tile consid-
ering the reduction of off-tile access power on network-on-chip, despite the higher static and dynamic power from the L2 cache. And it also suggests that high associativity of L2 caches above 16 may not pay for its cost on LLC traffic reduction.

Finally, Figure 4.3d draws the pareto fronts of relative performance improvement relative LLC bandwidth reduction when devoting additional area to a larger L2 cache or higher associativity varied from the starting configuration with 512KB, Assoc=1 L2 caches. We can see that a larger L2 cache can deliver roughly 4% speedup when only 1% additional area is consumed by the larger L2 cache. This is even better than scaling the accelerator speed in the ideal situation. And it’s worth mentioning that a larger L2 cache would deliver more benefits when further scaling up the accelerator speed which leads to more serious LLC bandwidth bottleneck, or there are multiple tiles contending for the limited main memory bandwidth in the chip-design phase.

The main takeaway from this experiment is that larger L2 caches should be explored to reduce the off-tile traffic and resolve the bandwidth mismatch challenges at last-level caches and main memory. It not only improves the performance by mitigating the bottlenecks on LLC and memory bandwidth, but also reduces the system power consumption by reducing off-tile access power. Moreover, these benefits come at a relative low cost on the area resource which results in a better efficiency on improving the tile performance than even ideal scaling of the accelerator speed.

### 4.2.3 Design space exploration for core-accelerator tiles

Finally, we perform extensive design space explorations varying different configurations of the modeled tile as described in Section 4.1, and evaluate all these configurations using metrics ‘Tile speedup’, ‘Normalized area efficiency’, and ‘Normalized power efficiency’ as described in Section 3.1.7 based on the statistics collected through the trace-driven simulations running ECP workloads.
Figure 4.4 shows the tile speedup, normalized area efficiency, normalized power efficiency and L2 cache size choices respectively along the y-axis of the tiles optimized for tile speedup, normalized area efficiency and normalized power efficiency respectively at each accelerator speed along the x-axis under two different LLC throttling situations as described in 3.1.2. The SF=0 data corresponds to the situation where the execution speed which is modeled as memory reference issue rate is dialed down when the last-level cache bandwidth requirement is reaching the upper bound. The SF=1 data could be also seen as a proxy for no-throttling because any excessive LLC bandwidth requirement would be sufficed at additional costs on area and power in “SF=1” situation.

As we can see, there are diminishing returns of tile performance when scaling up accelerator speed just as shown in the previous experiments when only accelerator speed is varied. The decrease on tile performance increase results from the incompetence of parallelizing ECP applications and the throttling effects on the excessive LLC bandwidth requirement, despite the fact that the tiles are employing larger l2 caches to reduce LLC bandwidth requirements as shown in the fourth subfigure. Furthermore, we could also see in the fourth subfigure that because of the power reduction effects of larger L2 caches as discussed in 4.2.2, the tiles optimized for power efficiency tends to employ larger L2 caches than the tiles optimized for area efficiency.

Moreover, when scaling up the accelerator speed, we observe the increase at first and then decline in both area efficiency and power efficiency. The reason for this is two-fold, which is similar to that when we only scale up the accelerator speed:

- The initial increase of the normalized area and power efficiency when scaling up the accelerator speed results from two factors. Firstly, the tile design is devoting more relative resources in terms of both area and power to the accelerator which is more efficient than the core. Secondly, at the initial scaling up of the accelerator, the tile doesn’t suffer from the throttling of the shared resource because the LLC bandwidth
Figure 4.4: Tile speedup, normalized area efficiency, normalized power efficiency and L2 cache size choices of optimized designs for different metrics at each accelerator speed.

- The decrease of the normalized area and power efficiency when further scaling up the accelerator speed comes from the two factors as well. Firstly, at the high accelerator speed, the accelerator already costs the most part of the tile area and power, further scaling up the accelerator no longer bring as much benefits of better allocating the resources. Secondly, the tile suffers from severe performance bottleneck of incompetence on applications parallelization and performance throttling resulted from the excessive requirement of the tile is below the upper bound.
Table 4.1: Optimal tiles for three different metrics under two LLC bandwidth throttling scenarios at high accelerator speed.

<table>
<thead>
<tr>
<th>SF=0</th>
<th>Tile Speedup Optimal</th>
<th>32 256 KB 1 128</th>
<th>16 MB 16 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area Efficiency Optimal</td>
<td>8 256 KB 1 128</td>
<td>1 MB 16 1</td>
</tr>
<tr>
<td></td>
<td>Power Efficiency Optimal</td>
<td>12 256 KB 1 128</td>
<td>2 MB 16 1</td>
</tr>
<tr>
<td>SF=1</td>
<td>Tile Speedup Optimal</td>
<td>32 256 KB 1 128</td>
<td>16 MB 16 1</td>
</tr>
<tr>
<td></td>
<td>Area Efficiency Optimal</td>
<td>8 256 KB 1 128</td>
<td>0 - -</td>
</tr>
<tr>
<td></td>
<td>Power Efficiency Optimal</td>
<td>12 256 KB 1 128</td>
<td>1MB 16 1</td>
</tr>
</tbody>
</table>
the applications and the available bandwidth capacity of shared LLC, which further limits the efficiency when scaling up the accelerator. A balance should be seek between scaling up and scaling out the tile.

- Employing larger L2 caches can reduce the LLC bandwidth capacity bottleneck and off-tile access energy, which improves the tile performance and power efficiency at the same time.

4.3 Designing the tiled heterogeneous chip

Now that we have some understandings about the tile level designs, we move on to distill understandings and insights of the chip-level designs and tackle the LLC and memory bandwidth bottleneck for the tiled heterogeneous system. We will first scale out the tile-level designs optimized for tile speedup, area efficiency and power efficiency respectively to understand the performance characteristics of a multi-tile chip design. And then we will perform design space exploration to further distill understanding and insights and evaluate the cache designs tackling the bandwidth challenges.

4.3.1 Scaling out optimal tiles

In this subsection, we study the memory system characteristics when naively scaling up the three tiles optimized for tile speedup, area efficiency and power efficiency respectively as described in Table 4.1 to understand the chip-level performance bottlenecks.

We performed trace-driven simulation with the ECP workloads as described in 3.1.5 on each tile design and project the statistics gathered to chip-level performance statistics based on the assumption that multiple tiles explore the process level parallelism as described in Section 1.2. And as we discussed in Section 3.1.2, we considered two upper bounds for our throttling model in terms of the excessive memory bandwidth requirement, 100 GB/s for
a DDR4 system and 1000 GB/s for a 4-stack HBM2 system. Accelerator speed would be dialed down if there is excessive memory bandwidth requirement.

Figure 4.5 shows the experiment results. Besides the six series showing the tile speedup, area efficiency and power efficiency respectively along the y-axis in each subfigure when scaling out the three aforementioned tile designs under the SF=0 LLC throttling scenario with two main memory configurations, there are two series 'Optimal (100 GB/s)' and 'Optimal (1000 GB/s)' added that does not correspond to the scaled out optimal tile designs. The two series show the best performance, the highest area efficiency, the highest power efficiency across the full design space explored with each number of the tiles. And please notice that the points with the same number of tile in one of these two series may correspond to different tile designs in the three subfigures, while the points in one of the other six series all correspond to the same tile design, which is one out of three described in Table 4.1. In short, the added two lines in each subfigure plot the best corresponding metric one can get when optimizing the tile design for the corresponding metric at each number of tiles.

As Figure 4.5 shows, firstly, although speedup optimal tile delivers the highest chip
performance when scaling the number of tiles, it comes with huge area and power costs considering its rather low area and power efficiency showing in the figure. The low efficiency comes from the throttling for the excessive LLC and memory bandwidth requirement and the incompetence of parallelizing applications to be explored by the accelerator of high parallelism. Secondly, we can see a diminishing increase in normalized performance when scaling out all the tile designs. This is because the memory bandwidth bottleneck are limiting the performance of more and more applications when scaling out. Thirdly, the memory bandwidth bottlenecks induced drops on area efficiency and power efficiency across all designs are observed starting at 2 tiles for DDR4 DRAMs, while HBM2 push this turning point to 8 tiles. The memory bandwidth throttling effect is also confirmed by the fact that same design in a 1000 GB/s HBM2 system significantly outperforms its counterpart in 100 GB/s DDR4 DRAM system in all aspects. For designs with number of tiles in the range of interest (2-32), 10-fold memory bandwidth of HBM2 can increase area efficiency and power efficiency by 448% 482% respectively.

The main takeaway is that the main memory bandwidth constraint is the major performance bottleneck at chip level design for the tiled heterogeneous system. Improving the cache bandwidth filtering capability should be of top design priority for the tiled heterogeneous system with scarce memory bandwidth capacity.

4.3.2 Design space exploration for the tiled heterogeneous chip

In this subsection, we discuss the results of our design space explorations for the tiled heterogeneous chip, trying to distill understanding and insights about the design of the memory hierarchy tackling the memory bandwidth challenge. To be clear, we are always scaling out the same core-accelerator tile, there are no considerations about the combination of the core-accelerator tiles with different configurations. The LLC bandwidth throttling scaling factor we use in this experiment is "SF=0", where the execution speed which is modeled
as the reference issue rate of the computing elements is dialed down if the LLC bandwidth requirement is reaching the upper bound.

We performed trace-driven simulations on each tile design in the design space discussed in 4.1. And the tile design performance statistics are projected to chip design performance statistics with the consideration of the memory bandwidth constraint induced performance throttling. Just like LLC bandwidth induced performance throttling, the execution speed of the computing elements modeled as memory reference issue rate is dialed down if the memory bandwidth requirement is reaching the upper bound.

The results are summarized into two Pareto fronts as shown in Figure 4.6. One Pareto front shows the trade-off between normalized chip performance and normalized area efficiency, the other shows the trade-off between normalized chip performance and normalized power efficiency. Each figure is composed of two sets of dotted lines, each corresponding one memory bandwidth upper bound. Taking Figure 4.6a as an example, the blue dotted line and yellow dotted line are of one set corresponding to the 100 GB/s memory bandwidth upper bound modeling the multi-channel DDR4 memories. As suggested in the legend, the yellow dotted line draws the pareto front about the tradeoff between the chip performance and power efficiency. While the blue dotted describes chip area consumption for each design in the yellow pareto front with certain normalized chip performance. Similarly the green dotted line and the red dotted line describes the pareto front about the tradeoff between chip performance and power efficiency and the area consumption of designs in the pareto front in the 1000 GB/s memory bandwidth upper bound situation.

As we can see from Figure 4.6, first of all, all designs on the Pareto fronts with HBM2 memory and most designs on the Pareto fronts with DDR4 memory have normalized power efficiency and normalized area efficiency larger than one, suggesting that the introduction of the heterogeneity improves the system efficiency to be better than the homogeneous system built with conventional cores. However, normalized chip performance increases at the cost of
the normalized area and power efficiency. The performance increase comes from either scaling up the accelerator issue rate or scaling out the number of tiles, but none of them can deliver the same level of the performance increase at the rate as the cost of area/power increases. Because the LLC or memory system bandwidth rather than execution/issue rate is the performance bottleneck for memory intensive ECP applications, none of the two scaling can improve the performance for these memory intensive applications. In other words, the design hits an memory wall for memory intensive applications like AMG, SWFFT and HPCCG.
Thus, the geometric mean of the performance across the applications don’t increase as faster as the cost increases, leading to the decrease on area/power efficiency. In the 1000 GB/s multi-stack HBM2 system, because the upper bound of memory bandwidth is much higher and fewer applications saturate it, area and power efficiency are thus degrading much slower as the increase of the normalized chip performance than the case in the 100 GB/s system with multi-channel DDR4 memory. And the maximum performance systems with HBM2 memory can scale out to 4 times higher than that of systems with DDR4 memories.

Figure 4.7a shows the number of tiles and accelerator speed of each tile of the designs in the two pareto fronts corresponding to the systems with HBM2 memory(1000 GB/s memory bandwidth upper bound). As we can see, it’s the increase on the number of tiles which enables the process-level parallelism that leads to the major performance increase of the tiled heterogeneous chip, considering that the number of tile increase as the growth of the normalized chip performance, while the accelerator speed fluctuates between 8 and 20. Because the exposed parallelism from the applications and LLC bandwidth bottleneck limits the benefits brought by fast accelerators as we discovered in Section 4.2.3, the chip performance scale poorly with the accelerator speed in each tile. On the other hand, memory bandwidth bottleneck only limits the performance of the applications that are extremely
memory intensive when scaling out the core-accelerator tile in the systems with HBM2 memories. Other applications can get linearly increased performance with more tiles in the chip through process-level parallelism.

Figure 4.7b shows the L2 caches and LLC slices size choices of the design in the two Pareto fronts with the HBM2 memory (1000 GB/s memory bandwidth upper bound). As we can gather, chip designs from both the area efficiency VS chip performance and the power efficiency VS chip performance Pareto fronts employ L2 caches that are even larger than each LLC slice. Please notice that noninclusive LLC caches are employed following the Skylake-SP microarchitecture as described in 4.1 such that L2 caches can be larger. The reasons here are two fold. Firstly, larger l2 caches mitigate not only the LLC bandwidth bottleneck but also the memory bandwidth bottleneck through reducing traffic going to the LLC. Secondly, the data locality is much less after filtering by L1 and L2 caches, making large LLC slices not cost-effective to build. We can also observe that the design optimized for the power efficiency employs the larger L2 caches than the designs optimized for the area efficiency. It could be explained by our understanding in Section 4.2.2: Larger L2 caches not only reduces the LLC traffics and bandwidth requirements, but also reduce network-on-chip and LLC access power by reducing the traffics coming out of L2.

To further confirm our idea that locality left after L1 and L2 caches doesn’t justify large LLC slices, we did a follow-up experiment to measure the memory bandwidth requirement when employing the LLC slices of different sizes. The cache configuration follows the baseline tile as described in 3.1. The accelerator speed is 4, and the number of L1D banks is 32. Figure 4.8 presents the memory bandwidth requirement of each tile when employing different LLC slice sizes as shown in the legend. As we can figure out, large LLC slices only reduce the memory bandwidth for the non-memory-intensive applications, for which there isn’t high memory bandwidth requirement in the first place. Thus, the mitigation on memory bandwidth bottleneck won’t justify a large LLC slice configuration.
To summarize, by analyzing the configurations usage of these pareto fronts, we could distill the following design suggestions:

- Employ low-associativity high-capacity high-bank-parallelism L1 data cache for the accelerator to sustain the high request rates from the accelerator and deliver the accelerator benefits while maintaining the locality exploration quality.

- Employ a single L2 cache shared between the core and the accelerator enables effective pooling of L2 cache capacity and the LLC bandwidth capacity between the core and the accelerator, improving the locality exploitations as well as mitigating the LLC bandwidth bottleneck.

- Shifting the cache capacity allocation across L2 and L3 towards the L2 caches. Large tile-level-shared L2 cache in each tile not only mitigate the LLC bandwidth bottleneck, but also reduces the power consumption by reducing the off-tile traffic. In the meantime, the locality left to be explored after two levels of caches is not worthy a big shared LLC slice.

- Employ high bandwidth memory. The scaling out and scaling up of the core-accelerator tile are both severely limited by the memory bandwidth bottleneck if the system is only equipped with the multi-channel DDR4 system as we observe from the results of naively scaling out the three tile designs as well as the results in the chip design space exploration. Using a HBM2 based memory system can achieve more than 3 times higher chip performance as well as area or power efficiency under the same area or power usage constraint than that of a system with the DDR4 memories.

- Ultimately, the benefits of acceleration are limited by global resource constraints such as LLC bandwidth or memory bandwidth. A balance between scaling up and the scaling out the core-accelerator tile is necessary to achieve good normalized chip performance as well as area/power efficiency.
Among the set of designs in the Pareto front trading performance with energy efficiency from our design space explorations, an optimal design point shows the effectiveness of the above insights. A tiled heterogeneous chip design with 16 tiles, 12x faster accelerator in each tile and a carefully optimized memory hierarchy brings 3.2x performance than a homogeneous chip of 16 baseline tiles. Acceleration alone with only baseline memory hierarchy design brings 2.2x performance. The use of the proposed high-bandwidth L1 caches improves the performance to 2.9x, and other memory hierarchy designs mitigating bandwidth bottlenecks at LLC and main memory brings the system performance to 3.2x.
CHAPTER 5
DISCUSSIONS AND RELATED WORK

As pointed out in the introduction, memory wall problem has been under extensive research for years, from the Dennard scaling era to the multi-core era. In this chapter, we discuss recent literatures tackling the memory wall. We separate them into two categories: methods reducing the memory bandwidth requirements and methods increasing available memory bandwidth. The primary differences of them from our work is that we focus on the memory hierarchy requirements and designs for tiled heterogeneous architectures, the adoptions of which have been seen in both academic studies [23, 50, 27] and industry adoptions [6].

5.1 Reducing memory bandwidth requirements

5.1.1 Effective data sharing

At the end of Dennard scaling, the switch to multi-core encourages extensive efforts on memory hierarchy redesigns. The defining characteristics made the multi-core processors different from multiple single core processors are the same-address-space data sharing opportunities across cores which, at the same time, bring challenges for locality exploitations. Private last-level caches can’t exploit locality for the shared data, and would thus waste the already scarce memory bandwidth, while shared caches introduce higher latency. Many works [48, 19, 24, 25] tried to optimize for both access latency and memory bandwidth requirement with hybrid last-level cache organizations. Chang et al. demonstrated that private caches with cooperative management for cache-to-cache transfers, replacement policy that discriminates against replicas to increase the number of unique cache block on chip, and global replacement of inactive data into peer private caches can deliver the benefits of both private and shared caches in terms of low latency and low capacity waste from replication and imbalanced cache usage [48]. Chitiahi et al. showed that employing a private tag array
for each core and shared data array at the last-level caches enables controlled replication, in-situ read-write sharing and capacity stealing which deliver 10% performance improvement over both private and shared last-level caches [24]. Furthermore, shared LLC caches with non-inclusive data array but inclusive tag array is proposed to reduce the capacity overhead of the inclusive replica for L2 data inside LLC cache and thus improve cache capacity efficiency while maintaining the snoop filtering functionality [56]. This has been adopted by industries in the Skylake-SP microarchitecture [12].

Switching the subject to GPUs. Traditionally only equipped with texture caches for texture mapping, GPU’s massive execution parallelism made the memory bandwidth a crucial resource. Out of within-warp data locality, within-block data locality and cross-instruction data reuse, caches on GPU mostly help exploring the within-block data locality because within-warp data locality is already explored by coalescing and cross-instruction reuse requires unreasonable cache size to be fully explored considering the streaming properties of the applications running on GPU [33]. Thus, industries adopted L1 and L2 caches, enabling efficient intra-SM and inter-SM data-sharings for the warps of the same thread block in modern GPU architectures like NVidia Pascal [38].

In terms of the cache-coherent heterogenous system that tightly integrates accelerators(GPUs) with conventional cores, efforts have been made to reduce the memory bandwidth requirements by lowering the coherence traffic overheads for data-sharing across computing elements. Gelado et al. [28] explore the opportunities that only the general purpose core can access accelerator memory space but not vice-versa, reducing the coherence traffics travelling up and down the memory hierarchy while providing an coherency illusion in the core’s perspective. Power et al. suggested that region caches enable redirecting the coherence traffics to direct access buses, reducing the consumptions on main memory bandwidth [43].

In the tiled heterogeneous architectures which is the subject of this study, although the ideas above can be applied for efficient data sharing between the core and the accelerator in
each tile, they are marginally beneficial. As we have shown in Section 3.2.2, our OpenMP-based interleaving programming model and performance discrepancy between the core and accelerator implies and encourages low interactions between the two computing elements.

5.1.2 Compressing data on caches or main memory

Compressing data on caches and main memory are indirect and direct ways to reduce the memory bandwidth requirement respectively. Compressing the cacheblocks in caches improves effective capacity of a cache, enabling better locality exploitations. Studies [40, 47] show that the hardware based compression of the SRAM caches with clever tag sharing supports improves effective cache capacity and reduces memory bandwidth indirectly. On the other hand, compressing and decompressing data on chip before sending writebacks to main memory and after receiving cachelines from memory respectively not only increases the effective main memory capacity, but also increases the effective main memory bandwidth directly. Wang et al. has demonstrated that hardware-based approaches to online clustering the data and compress the clustered data with a xor-based codec before sending to main memory effectively reduce the data movement bandwidth requirement [51]. Bernini et al. explored opportunities of on-the-fly transparent compressions and decompressions between the LLC and main memory [20]. It reduces 34% of the memory traffic, improving the system performance by mitigating the memory bottlenecks. These hardware-based transparent compression/decompression methods discussed above are orthogonal to our work, and is possible to be applied in the tiled-heterogeneous architecture to further mitigate the memory bandwidth bottlenecks. But there’re great challenges on the design hardware compressor/decompressor considering the tightly integrated of high-performance accelerator would generate high reference bandwidth to be compressed/decompressed.
5.1.3 Adaptive cache access granularity

Ideas to reduce the unnecessary data fetching for caches have been explored in both CPU-based [55, 54] and GPU-based [44] systems. For applications with irregular memory accesses, cachelines fetched from the main memory are not fully used for computations. All of the mentioned works develop mechanisms to adaptively adjust the cache block size to be fetched from the main memory to avoid fetching unneeded data and reduce main memory bandwidth requirements. For GPU systems, there are also works exploring complete bypassing the cache systems for data with low reuse opportunities under the software controls [32]. The efforts of adaptive cache access granularity could be applied to the tiled heterogeneous architecture, though the benefits and the overhead is unknown and likely varied across different applications. We leave this as part of the future work.

5.1.4 Design space explorations

There are multiple efforts [35, 21, 18, 39] employing analytical models to reason how to allocate resources of energy and chip area to computing elements and memory hierarchy, whether to enable different architecture features or configurations of each of their sub-components through design space explorations in order to effectively balance the computing capability and bandwidth requirements. Shifting resource allocation towards the memory hierarchy in memory-bound situations enables better locality exploitations and reduce the main memory bandwidth requirements without. Yingmin et al. took ITRS data to form pin bandwidth, thermal and power delivery constrains in design space explorations aside from normal resource constraint of area and energy. They further discovered that thermal constraint dominates and the configuration desires between computing intensive and memory intensive applications are irreconcilable [35]. On the other hand, Azizi et al. employed marginal analysis to evaluate the choices on both architecture and circuits and identified voltage scaling as the effective method for trading off performance and energy because the optimal architec-
ture and circuits doesn’t change during voltage scaling [18]. We focus on memory intensive applications in our memory hierarchy studies for tiled heterogeneous architecture, so the efforts discussed here are only marginally relevant.

5.2 Increasing available bandwidth

5.2.1 Employing new memory technology with higher bandwidth capacity

Traditionally, memory bandwidth has been improved through increasing the memory interface frequency or the number of channels at the interface. With the development of 2.5D IC process, we have seen increasing adoptions of through silicon vias and interposers for further increasing the channels between memory chips and the computation chip, such as HBM or HMC. AMD and NVidia has already been exploring HBM2 on their recent GPU products to suffice the memory bandwidth need [3, 9]. Although being able to provide higher bandwidths through interfaces with more channels, high bandwidth memory doesn’t catch up with traditional DRAM in terms of capacity. There are efforts [41, 42] trying to address this issues through efficient management of a hybrid memory hierarchy that employs both stacked DRAM(HBM) and traditional DRAMs. Peng et al. argue that a horizontal organization of HBMs and traditional DRAMs wins over a vertical organization considering the aggregated benefits of both bandwidth and capacity in a horizontal organization [41]. Moreover, Peng et al. found out HBM helps on applications with regular access patterns but not on the applications with irregular patterns where LLC miss rate are high and the system performance limited by longer access latency of the HBM [42]. In our work of tiled heterogeneous architecture, we also suggest to employ HBM to mitigate the memory bandwidth bottlenecks as discussed in Section 4.3.
5.2.2 In-memory processing

Also benefited from the technology process development of 2.5D IC, there is a trend of exploring in-memory processing [36, 29, 26, 16] to move computations to data instead of the other way around. 2.5D IC enables computation logic to be implemented on the same interposer with memory chips inside the same package and access data via through silicon vias. In-memory computing can thus enjoy immense memory bandwidth, and dramatically reduce data movement which is the main source of energy consumption. There are studies [36, 29] exploring the opportunities of conducting data transformation in memory for the applications running on the general purpose core for higher memory bandwidth usage available. But the benefits are limited considering they don’t well on hot data where hot data has to be transferred back to memory from caches in the first place to enjoy the data transformation. Other studies [26, 16] address this issues by further consider pushing data analysis operators on to the in-memory logic and perform major computations in memory to improve performance and reduce energy consumption at the same time. They are still marginally beneficial in the tiled heterogeneous architectures because we are considering high-performance accelerators of which the energy budget is hard to met in in-memory processing scenarios.

5.3 Summary

This chapter presents the related works tackling the memory wall through both reducing the memory bandwidth requirement and increasing the memory bandwidth available. We covered works on efficient data sharing at LLC, compressing caches or memories, adaptive cache access granularity, system design space explorations, employing high bandwidth memories and in-memory processing to mitigate the memory bandwidth bottlenecks and improve system performance.
CHAPTER 6
SUMMARY AND FUTURE WORK

6.1 Summary

In this work, we studied the memory hierarchy design for the systems implementing the tiled heterogeneous architecture where bandwidth-oriented latency-tolerant high-performance accelerators are introduced to improve the performance as well as the efficiency of the computing system at the end of the moore’s law.

To understand the memory system characteristics in the systems implementing the tiled heterogeneous architecture, we proposed a general accelerator model, as well as the performance model, area model, power model, and application execution model for the tiled heterogeneous architecture. A trace-driven simulation framework is built implementing the proposed models to collect statistics about performance and efficiency of the memory hierarchy while running ECP workloads which are scalable scientific workloads for evaluations on different memory hierarchy configurations.

Several bandwidth mismatch challenges are identified regarding the memory hierarchy design for tiled heterogeneous architecture in our preliminary experiments of trace-driven simulations with ECP workloads. Firstly, the bandwidth capacity of conventional L1 caches becomes the performance bottleneck when serving faster accelerators. Secondly, faster accelerator creates huge traffics and off-tile bandwidth requirement imbalance between the core and the accelerator of the core-accelerator tile. Thirdly, the LLC bandwidth requirement and memory bandwidth requirement are exceeding the system limit because of the acceleration.

Five design suggestions are proposed to tackle these problems. Firstly, a low-associativity high-bank-parallelism cache organization is proposed to suffice the accelerators’ request rate at the L1 cache and actually deliver the accelerator benefits. Secondly, a tile-level-shared L2 cache is proposed to eliminate the off-tile bandwidth requirement imbalance and effectively
pool the L2 cache capacity and LLC bandwidth capacity between the core and the accelerator in each tile. Thirdly, large L2 caches are recommended to reduce both LLC and main memory bandwidth requirement and save off-tile access energy at the same time. Fourthly, it is encouraged to employ high bandwidth memory for both higher performance and better system efficiency. Last but not the least, it's recommended to scale up the accelerator in a core-accelerator tile only to moderate performance, while scaling out the core-accelerator tiles in a system with the high bandwidth memory equipped exploring the process-level parallelism to achieve the balance between scaling up and scaling out and deliver good system performance as well as area and power efficiency at the same time.

Furthermore, an optimal design point among the pareto fronts trading performance with energy efficiency from our design space exploration demonstrates the effectiveness of our design insights and suggestions. A tiled heterogeneous chip with 16 tiles and 12x faster accelerator in each tile brings 3.2x performance than a homogeneous chip of 16 baseline tiles. Acceleration alone with only baseline memory hierarchy design brings 2.2x performance. The use of the proposed high-bandwidth L1 caches improves the performance to 2.9x, and other memory hierarchy designs mitigating bandwidth bottlenecks at LLC and main memory brings the system performance to 3.2x.

### 6.2 Future Work

There are multiple directions this study can be extended.

Firstly, we learned that the cache capacity allocation across L2 and L3 caches are shifting towards L2 to better mitigate the bandwidth bottlenecks at both LLC and main memory under the programming model exploring processor-level parallelism across multiple tiles. Considering a shared memory thread-level parallelism across multiple tiles and extend the performance model accordingly would complement our results and have a better understanding and design guideline for shared non-inclusive LLC caches because cross-tile data sharing
is one of their major functionalities.

Secondly, we didn’t consider the virtualization of the computing elements and assume the exclusive execution of the applications. Applying a performance model with virtualization considerations as well as a corresponding execution model would enable the design considerations of the translation lookaside buffer which could also be a performance bottleneck of the heterogeneous systems considering the higher memory reference rates from the accelerator.

Thirdly, LLC bandwidth bottlenecks are identified as huge performance impacts under our performance model with a simple LLC bandwidth throttling mechanism. It’s worth to further look into a detailed LLC functionality and performance model which could potentially enable the co-design of the memory hierarchy with the network-on-chip to better mitigate the LLC bandwidth bottlenecks and improve the system performance.

Furthermore, as shown in the study, the bottleneck from the limited bandwidth capacity of the shared resources like LLC and main memory significantly reduces the performance of the tiled heterogeneous system. It signals the opportunity of employing memory system storing compressed/encoded data and accelerators operates directly on compressed/encoded data for certain applications, e.g. the analytical processing of the column store databases, to improve performance by reducing bandwidth requirements through the compressed formats.
REFERENCES


73


