PlaceIT: Placement-based Inter-Chiplet Interconnect Topologies

Patrick Iff[†] and Benigna Bruggmann[†] and Maciej Besta[†] and Luca Benini[†‡] and Torsten Hoefler[†]

ETH Zurich, Zurich, Switzerland[†], University of Bologna, Italy[‡]

patrick.iff@inf.ethz.ch, maciej.besta@inf.ethz.ch, torsten.hoefler@inf.ethz.ch, lbenini@iis.ee.ethz.ch

Abstract—2.5D integration technology is gaining traction as it copes with the exponentially growing design cost of modern integrated circuits. A crucial part of a 2.5D stacked chip is a low-latency and high-throughput inter-chiplet interconnect (ICI). Two major factors affecting the latency and throughput are the topology of links between chiplets and the chiplet placement. In this work, we present PlaceIT, a novel methodology to jointly optimize the ICI topology and the chiplet placement. While state-of-the-art methods optimize the chiplet placement for a predetermined ICI topology, or they select one topology out of a set of candidates, we generate a completely new topology for each placement. Our process of inferring placement-based ICI topologies connects chiplets that are in close proximity to each other, making it particularly attractive for chips with silicon bridges or passive silicon interposers with severely limited link lengths. We provide an open-source implementation of our method that optimizes the placement of homogeneously or heterogeneously shaped chiplets and the ICI topology connecting them for a user-defined mix of four different traffic types. We evaluate our methodology using synthetic traffic and traces, and we compare our results to a 2D mesh baseline. PlaceIT reduces the latency of synthetic L1-to-L2 and L2-to-memory traffic, the two most important types for cache coherency traffic, by up to 28% and 62%, respectively. It also achieve an average packet latency reduction of up to 18% on traffic traces. PlaceIT enables the construction of 2.5D stacked chips with low-latency ICIs.

Website & code: https://github.com/spcl/placeit

I. INTRODUCTION

The growing demand for computing performance has always been met by increasing the number of transistors per chip, which is only possible due to CMOS technology scaling. However, as we keep pushing the boundaries of technology scaling, we encounter multiple challenges. Firstly, whenever we transition to a more advanced technology node, the non-recurring cost due to physical design, verification, software, mask sets, and prototyping almost doubles [26]. As a result, designing a chip in an advanced technology node is only economically viable if the chip is manufactured in vast quantities. Secondly, many chip components such as I/O drivers, analog circuits, or static random access memories (SRAMs) have reached their scaling limit. This means that we cannot shrink these components further, even if we use a more advanced technology with a smaller feature size. Thirdly, advanced technology nodes suffer from high defect rates, diminishing the yield and inflating the recurring cost. To tackle these challenges, new chip-design paradigms have been developed.

One of these new paradigms is 2.5D integration, where multiple silicon dies called chiplets are integrated into the same package. Once designed, a single chiplet can be reused in multiple 2.5D stacked chips, which increases the ratio of production volume to non-recurring cost. Another advantage is that multiple chiplets - fabricated in different technologies - can be integrated into the same package. This means that only components that can take full advantage of technology scaling are built in bleeding-edge technologies. Components that have reached their scaling limit are fabricated in more mature and hence less costly technology nodes. Furthermore, chiplets are smaller than monolithic chips. Therefore, manufacturing chiplets results in less silicon area loss due to fabrication defects and hence a higher yield. Due to these economic advantages, chip vendors such as AMD [31] and NVIDIA [24] have adopted the 2.5D integration paradigm.

An important challenge when designing 2.5D stacked chips is the construction of a low-latency and high-throughput inter-chiplet interconnect (ICI). To build an ICI, we connect different chiplets using die-to-die (D2D) links. These links are fabricated in an organic package substrate, silicon bridge, or silicon interposer, and they are connected to the chiplets using controlled collapse chip connection (C4) bumps or microbumps. The number of bumps per chiplet is limited, and so is the bandwidth of D2D links. In addition to having lower bandwidth than links in monolithic chips, D2D links also have higher latency. This latency is caused by wire delay and by physical layers (PHYs) that are necessary in both the sending and the receiving chiplet. PHYs are needed to convert between protocols, voltage levels, and frequencies, which are usually different between on-chiplet links and D2D links. Due to these limitations, the ICI can quickly become a bottleneck.

Existing approaches to maximize the performance of the ICI either optimize the placement of chiplets (with potentially heterogeneous shapes) for a predetermined ICI topology [16], [27], [33], [11], [32], [28], [7], select one topology out of a set of candidates [9], [8], or they optimize the ICI topology for a 2D grid of homogeneously shaped chiplets on an active interposer [21], [34], [4]. To the best of our knowledge, there is no prior work on ICI topologies for chips with heterogeneously shaped chiplets or with passive silicon interposers or silicon bridges. To fill this gap, we propose PlaceIT, a novel optimization methodology to jointly optimize the chiplet placement and ICI topology of such architectures.

The key idea is as follows: We optimize the chiplet placement without a predetermined topology. For each placement generated by an optimization algorithm, we infer a placement-
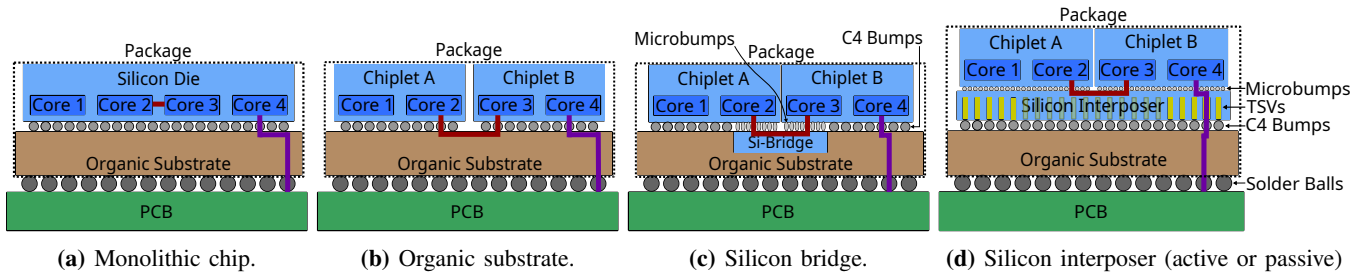
(a) Monolithic chip.   **(b)** Organic substrate.   **(c)** Silicon bridge.   **(d)** Silicon interposer (active or passive)

Fig. 1: (§II-A) 2.5D integration technologies (side view). We show a core-to-core link (red) and an off-chip link (purple).

based ICI topology by connecting chiplets that are in close proximity in that specific placement. We then compute the latency and throughput of this combination of placement and topology for different traffic types. These latencies and throughputs together with the total chip area are used to compute a user-defined quality-score of the placement, which is returned to the optimization algorithm. Based on this quality score, the algorithm can further optimize the placement. By following this iterative process, we jointly optimize the chiplet placement and the ICI topology.

We provide our open-source framework implementing the proposed placement and topology co-optimization methodology, which we evaluate using both synthetic traffic and traffic traces. A 2D grid of chiplets with a mesh topology is used as a baseline since many proposals for 2.5D stacked chips [12], [23], [35], [17], [37] use such an architecture. We reduce the latency of synthetic L1-to-L2 and L2-to-memory traffic, the two most important traffic types for cache coherency traffic, by up to 28% and 62% respectively. For real traffic traces, we reduce the average packet latency for almost all traces and architectures considered (reduced by an 8% or 18% on average depending on the configuration of PHYs within a chiplet).

II. BACKGROUND

A. 2.5D Integration

The most prominent difference between monolithic chips (Fig. 1a) and 2.5D stacked chips (Figs. 1b to 1d) is that the former only contain a single silicon die while the later contain multiple dies which are called chiplets. We categorize 2.5D stacked chips into those that only use an organic substrate (Fig. 1b), those that use silicon bridges (Fig. 1c), and those that use a silicon interposer (Fig. 1d). There are passive silicon interposers [25], [22] only containing metal layers fabricated in the back end of line (BEOL) and active silicon interposers [39], [10] also containing transistors fabricated in the front end of line (FEOL). Table I summarizes the packaging options.

1) 2.5D Stacked Chips using an Organic Substrate: In 2.5D stacked chips without a silicon bridge or interposer (Fig. 1b), the chiplets are directly connected to the organic substrate using C4 bumps. The rather large bump-pitch of 150-200 μm severely limits the number of bumps that can be used to connect a chiplet to the substrate. Therefore, only a limited number of wires is available for the construction of D2D links. Consequently, wide, high-bandwidth links that are commonly

Technology	Cost	Link Bandwidth	Maximum Link Length*	Package-Level Routers
Monolithic Chip	High	High	unlimited	n/a
Organic Substrate	Low	Low	10-25mm** [1]	No
Silicon Bridge	Mid	Mid	2-4mm [29], [6]	No
Passive Interposer	Mid	Mid	2-4mm [1]	No
Active Interposer	High	Mid	unlimited	Yes

TABLE I: (§II-A) Overview of packaging technologies.
*Maximum link length depends on data rate, bump pitch, etc.
**Source termination (up to 50 mm for double termination).

used in monolithic chips cannot be used to connect different chiplets. Instead, the data is serialized and transmitted over narrow D2D links, which can quickly become a bottleneck.

2) 2.5D Stacked Chips using a Silicon Bridge: To counteract the bandwidth bottlenecks formed by D2D links in organic substrates, silicon bridges (Fig. 1c) such as Intel's embedded multi-die interconnect bridge (EMIB) [29] or IBM's direct bonded heterogeneous integration (DBHi) [36] were introduced. For off-chip communication, chiplets are connected to the package substrate using C4 bumps. For high-bandwidth die-to-die communication, chiplets are connected to the silicon bridge using microbumps. Since these microbumps have a pitch of 30-60 μm, the number of wires and hence the bandwidth of D2D links is about $10\times$ higher than in chips with a package substrate only. However, these D2D links still deliver lower bandwidth than on-die links in monolithic chips. Furthermore, silicon bridges limit the link length [29], [6].

3) 2.5D Stacked Chips using a Passive Silicon Interposer: Another way of improving the bandwidth of D2D links are passive silicon interposers, such as TSMC's Chip-on-Wafer-on-Substrate (CoWoS) [3]. Here, chiplets are connected to the interposer using microbumps and the interposer is connected to the package substrate using C4 bumps. Through-silicon vias (TSVs) are used to provide connectivity between chiplets and the substrate. One major limitation of passive interposers is the limited length of D2D links. Since passive interposers do not contain any transistors, it is not possible to build buffered links. Therefore, unbuffered links are used, and their length cannot exceed some millimeters [38], [2], [1].

4) 2.5D Stacked Chips using an Active Silicon Interposer: Active interposers [39], [10] allow the construction of buffered D2D links, therefore, links in active interposers can be arbitrarily long. Another advantage of active interposers is that they allow the integration of package-level routers. However, active interposers come with their own challenges. Since active

interposers require the FEOL process, they are about ten times more expensive than passive interposers [8]. Furthermore, they suffer from a lower yield than passive interposers, which is problematic as interposers are usually large. Another challenge is the fact that active interposers generate heat, which is hard to remove since the interposer sits below the chiplets. Due to these drawbacks of active silicon interposers, our work focuses on the remaining 2.5D integration technologies.

B. Optimization Algorithms Suitable for Chiplet Placements

1) Best Random: This algorithm generates random solutions to an optimization problem until the time budget is exhausted, then, it returns the best solution that was found. We use this naïve algorithm as a baseline to determine whether the more advanced algorithms perform better than random.

2) The Genetic Algorithm: The genetic algorithm (GA) mimics biological evolution, where a solution to an optimization problem is viewed as an individual. Each individual has a fitness score, corresponding to the quality of the solution. The algorithm maintains a population of P individuals, which changes through a series of generations. In each generation, individuals with an insufficient fitness score are eliminated from the population and existing individuals with good fitness scores are merged to produce new individuals. The key idea is to merge two solutions with good properties to achieve a new solution with even better properties. For the genetic algorithm to work properly, we need to formulate a merge function that enables combining the strengths of two solutions while eliminating their weaknesses.

3) Simulated Annealing: The simulated annealing (SA) algorithm is based on the annealing process, in which a material is heated up and slowly cooled down in order to alter its properties. We start with a randomly selected solution and alter this solution through a series of iterations. In each iteration, we explore a *neighboring* solution. If this solution is better than the current one, we accept it, i.e., we set it as our current solution. Otherwise, we only accept it with a certain probability. This probability decreases over time since we assume that the longer our algorithm is running, the closer to the optimum we are and hence the less favorable it is to accept an inferior solution. When formulating a problem for SA, we need a method to generate *neighboring* solutions with a similar quality. If there is no sufficient correlation between the quality of a solution and the quality of its neighbors, then simulated annealing can not work properly.

III. PLACEMENT AND TOPOLOGY CO-OPTIMIZATION

2.5D stacked chips require low-latency and high-throughput ICIs. The latency mainly depends on the number of chiplet-to-chiplet hops and on the link latency. The throughput primarily depends on the frequency, at which the links can be operated and on the congestion, i.e. how many different flows compete for the same link. While hop count and congestion both depend on the ICI topology, link latency and link frequency both depend on the link length, which depends on the combination of ICI topology and chiplet placement. To maximize the performance of the ICI, chiplet placement and ICI topology must be perfectly aligned. Selecting a topology first and then optimizing the placement for that topology might not yield satisfying results, since the choice of a suboptimal topology can prevent us from finding a good placement. Selecting a placement first and the optimizing the topology results in a similar problem. The solution to this problem is to co-optimize the chiplet placement and ICI topology.
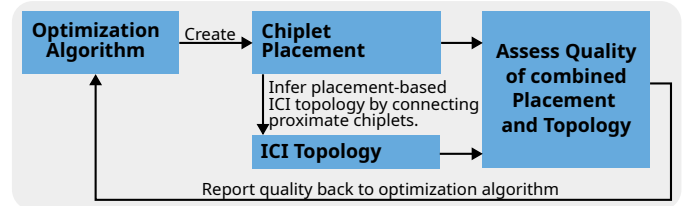

Fig. 2: (§III) Placement and topology co-optimization.

Fig. 2 visualizes our proposed chiplet placement and ICI topology co-optimization methodology. An optimization algorithm is used to optimize the chiplet placement. For each placement that the optimization algorithm produces, a placement-based ICI topology is inferred. In this inference process, we minimize the length of D2D links. We then assess the quality of the combined chiplet placement and ICI topology, which we return to the optimization algorithm. Using this methodology, the optimization algorithm does not only optimize metrics of the placement itself, like, e.g., the total area, but it also optimizes the placement in a way that enables us to construct good ICI topologies on top of it. This is comparable to placers in the very large scale integration (VLSI) place & route step that optimize the placement of macros not only for area but also for routability of wires. The difference to our methodology is that placers optimize the macro placement for the routability of predetermined nets, while we optimize the chiplet placement for the inference of an ICI topology with a yet unknown connectivity pattern.

IV. PLACEIT ARCHITECTURE

In this section, we present our open-source implementation of the proposed chiplet placement and ICI topology co-optimization methodology. Our framework is based on the following list of assumptions:

- Every chiplet is categorized as compute, memory, or IO.
- We know the number and locations of PHYs in each chiplet.
- PHYs use a common protocol, e.g., universal chiplet interconnect express (UCIe) [38] or bunch of wires (BoW) [2].
- The data-width of all PHYs is the same, i.e., we can connect any two PHYs with a D2D link.
- For each chiplet, we know whether it can relay traffic. Relaying traffic means that a message enters the chiplet through one PHY and leaves it through a different PHY.
- Off-chip traffic leaves the chip through an IO-chiplet as in the AMD EPYC and Ryzen processor families [31].

Fig. 3 provides an overview of the PlaceIT architecture. The experiment configuration contains the general configuration of PlaceIT and parameters describing the architecture
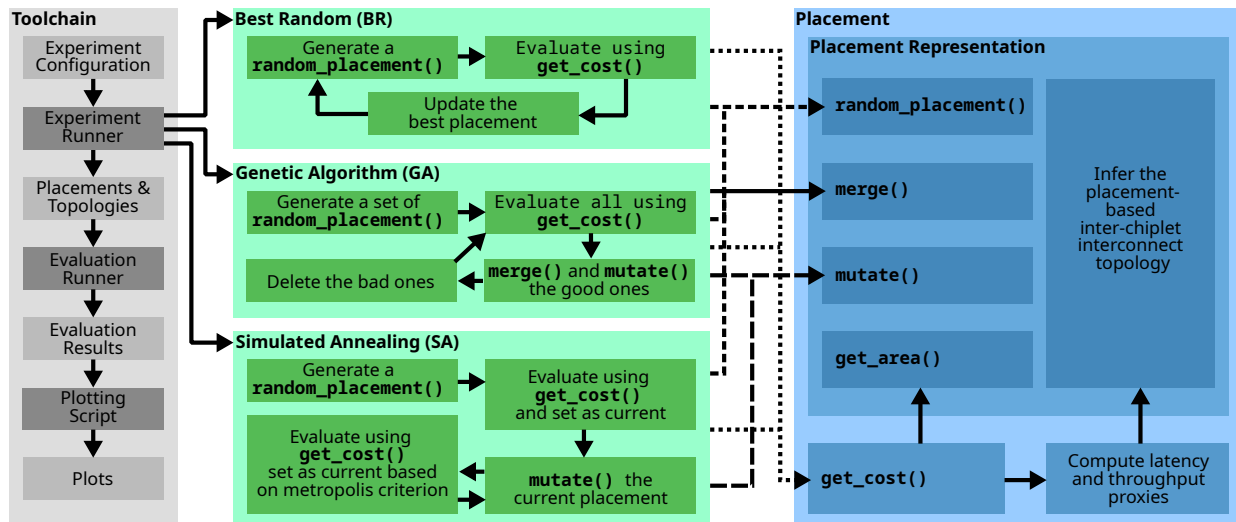
Fig. 3: (§IV) Overview of the PlaceIT architecture.

to be optimized (see Table II) as well as parameters for the three optimization algorithms. The experiment runner launches multiple runs of each optimization algorithm. We support the optimization algorithms best random (BR), genetic algorithm (GA), and simulated annealing (SA), but custom ones can be added. Optimization algorithms interact with the placement through the following functions:

- `random_placement()`: Return a random placement.
- `mutate(X)`: Return an altered version of placement X.
- `merge(X, Y)`: Return a hybrid of placements X and Y.
- `get_cost()`: Return the placement's cost.

General PlaceIT configuration	
Opt. Algorithm	*Best Random*, *Genetic Algorithm*, *Simulated Annealing*.
Placement Repr.	*Homogeneous* or *Heterogeneous*.
Time Budget	Run the algorithm for this many seconds.
Repetitions	Perform this many repetitions of the experiment.
Norm. Samples	Number of samples used to compute the cost-function normalizers (see Section IV-B).
Mutation Mode	How to mutate placements (see Sections V-A and VI-A).
Parameters specifying the architecture to be optimized	
Distance Type	*Manhattan* or *Euclidean* distance.
Max. Length	The maximum length of a D2D link [mm].
N_C,N_M,N_I	Number of compute-, memory-, and IO-chiplets.
Dimensions	The width and height of each chiplet type [mm].
PHYs	The number and position of PHYs in each chiplet type.
Relay Chiplets	List of chiplets that can relay traffic.
L_P,L_L,L_R	Latency of PHYs, links, and relaying a message through a chiplet [cycles].

TABLE II: (§IV) Experiment configuration parameters.

The core of the placement is the placement representation, which implements the first three out of the four functions listed above. We provide two placement representations, one for homogeneously shaped chiplets (see Section V) and one for heterogeneously shaped chiplets (see Section VI), but custom ones can be implemented. In addition to the `random_placement()`, `merge()`, and `mutate()` functions, the placement representation contains functions to get the area and the placement-based ICI topology. To assess the quality of a placement, we infer the placement-based ICI

topology and use it to estimate the ICI latency and throughput, which we combine with the total chip area to compute a user-defined cost function. Our performance estimates and cost function are explained in detail in the following sections.

A. Computation of Performance Estimates

We use the RapidChiplet [19] toolchain to estimate the latency and throughput of the ICI. RapidChiplet provides high-level latency and throughput proxies for compute-to-compute (C2C), compute-to-memory (C2M), compute-to-IO (C2I), and memory-to-IO (M2I) traffic as well as simulation-based results for both synthetic traffic patterns and application traces.

B. The Cost Function

Specifying the cost function is the most crucial part of applying the PlaceIT framework, as it sets the goal towards which the chiplet placement and ICI topology are co-optimized.

As a running example, throughput this paper, we use PlaceIT to build a general-purpose cache-coherent system targeted at various scientific simulation workloads from the high-performance computing domain. To that end, we use a weighted sum of the latency and throughput of C2C, C2M, C2I, and M2I traffic. In addition, we incorporate a term for the area of a minimal rectangle fully enclosing all chiplets (from now on referred to as the area) to ensure that PlaceIT produces compact placements. Fig. 4 visualizes the correlation of the final cost value with its nine components (four latency and four throughput terms plus the area term).

The user-defined cost function makes PlaceIT applicable to a wide range of optimization goals. One could for example use estimates of the ICI latency and throughput under a certain application trace or a set of application traces to design a domain-specify accelerator, e.g., for machine learning training and inference, image and video processing, or graph analytics.

V. PLACEIT FOR HOMOGENEOUS CHIPLET SHAPES

We discuss our placement representation and optimization results for chiplets with homogeneous shapes. In this setting,
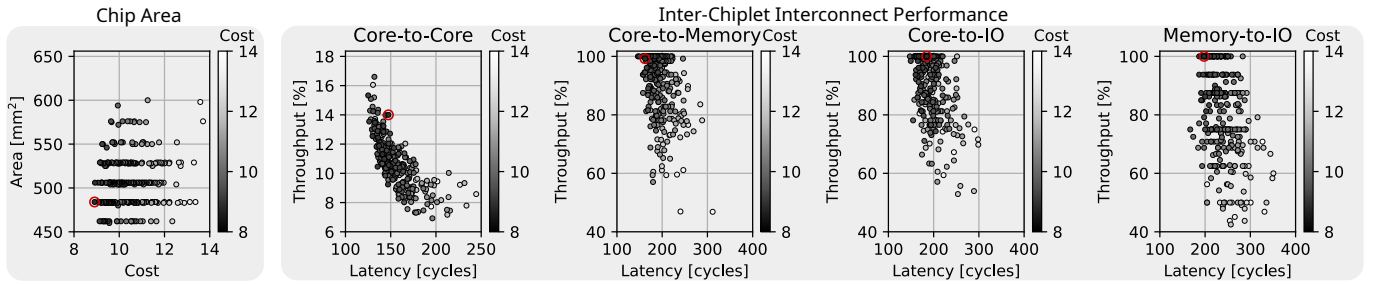
Fig. 4: (§IV-B) Correlation of cost value with its components. Points corresponds to random designs with colors indicating the placement's cost. Red circles highlight the lowest-cost placement. Throughput is given in percent of the theoretical peak.

we assume that there are two possible configurations of PHYs for a chiplet: A chiplet either has four PHYs, one on each side (we use this for compute-chiplets), or a single PHY on one side (we use this for memory- and IO-chiplets).

A. Placement Representation

We represent a placement of homogeneously shaped chiplets as an $R \times C$ grid such that $R \cdot C \geq N_C + N_M + N_I$. A grid cell can contain a compute-, memory-, or IO-chiplet, or it can be empty (see Fig. 5a). Chiplets with a single PHY can be rotated, but chiplets with four PHYs cannot, since this would result in an isomorphic placement with identical cost. Furthermore, chiplets with a single PHY are always placed in a way s.t. the PHY faces another chiplet and not the "outside". We provide the following functions for homogeneous placements:

- `random_placement()`: Randomly place all chiplets in the $R \times C$ grid (see Fig. 5a for an example).
- `mutate(X)` We provide four different mutation modes.
  - *any-both*: swap two chiplets **and** rotate one.
  - *any-one*: swap two chiplets **or** rotate one.
  - *neighbor-both*: swap two adjacent chiplets **and** rotate one.
  - *neighbor-one*: swap two adjacent chiplets **or** rotate one.

  Only chiplets of different types can be swapped, since swapping two chiplets of the same type would result in an isomorphic placement with identical cost. Fig. 5b visualizes an *any-one* and a *neighbor-one* mutation.
- `merge(X,Y)` If chiplet types and/or chiplet rotations match between placements X and Y, then, those types and/or rotations are carried over to the merged placement. The remaining, unplaced chiplets are placed and rotated randomly. Figs. 5c and 5d visualize the merge process.
- `get_network()` The placement-based ICI topology is created by connecting any two opposing PHYs of adjacent chiplets (see Fig. 5e). This might result in an unconnected placement (as in Fig. 5e). In those cases, the operation that created the unconnected placement is repeated.
- `get_area()` The area of a homogeneous placement is chiplet_size $\cdot R \cdot C$. Consequently, in the homogeneous case, all placements of the same architecture have the same area.
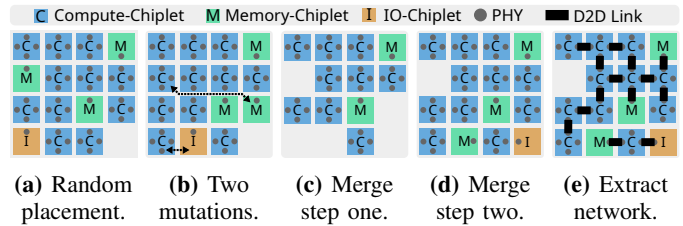

(a) Random placement.   **(b)** Two mutations.   **(c)** Merge step one.   **(d)** Merge step two.   **(e)** Extract network.

Fig. 5: (§V-A) Homogeneous placement representation. **(b)** is a mutation of (a), **(c) and (d)** show the process of merging (a) and (b), **(e)** extracts the network of (d).

B. Optimization Results

We use PlaceIT to optimize two architectures, one with 32 compute-, 4 memory-, and 4 IO-chiplets (called *32 cores homogeneous*) and one with 64 compute-, 8 memory-, and 8 IO-chiplets (called *64 cores homogeneous*). Both architectures use chiplets of size $3mm \times 3mm$. We perform our experiments on a single core of an Intel Xeon X7550 running Debian 11.

To set the weights of C2C, C2M, C2I, and M2I latency and throughput in the cost function accordingly, we analyze multiple cache-coherency traffic traces from the Netraces v1.0 [13] trace collection. We observe that 0%-5% of the messages are C2C traffic, 80%-95% are C2M traffic, and 3%-16% are M2I traffic. Therefore, we set the cost function weights for the area as well as C2M and M2I latency and throughput to 2 while we set the weights for C2C and C2I latency and throughput to 0.1. Table III shows the remaining parameters.

	Parameter	32 cores homogeneous	64 cores homogeneous
General	Time Budget	3600 s	3600 s
	Repetitions	10	10
	Norm. Samples	500	500
	Mutation Mode	neighbors-one	neighbors-one
	L_R, L_P, and L_L	10, 12, and 1 cycles	10, 12, and 1 cycles
GA	Population size (P)	200	50
	Elitism size (E)	30	8
	Tournament size (T)	30	8
	Mutation prob. (p_m)	0.5	0.5
SA	Initial Temp. (T_0)	40	35
	Iterations (L)	250	50
	Cooling param. (α)	1	1
	Adaptive param. (β)	5	5

TABLE III: (§V-B) Experiment configuration for homogeneously shaped chiplets.

Fig. 6 shows our results for the two architectures. For both architectures, all three optimization algorithms are able to outperform the baseline architecture which is based on a 2D mesh (see Fig. 13). This is not surprising, as such an architecture is not ideal for C2M and M2I traffic, which was the highest weighted traffic in our cost function. Furthermore, we observe that both the GA and SA significantly outperform best random (BR) which shows that the two more complex optimization algorithms work as intended.
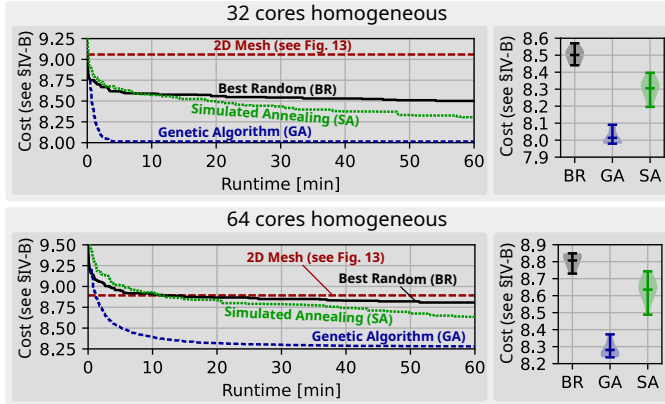

Fig. 6: (§V-B) Results for homogeneously shaped chiplets. We show the evolution of the result over time (left) and the distribution of the final result over 10 repetitions (right). See Fig. 13 for the placements found by the best algorithm.

We observe that the GA is able to quickly converge to a good solution. For the 32 core architecture with a solution space of approximately 10^{14} solutions, the GA converges after five minutes. For the 64 core architecture with a solution space of approximately 10^{30} solutions, the convergence takes about 30 minutes. SA is able to continuously increase the quality of the solution, however, the allocated compute budget was not sufficient for SA to reach convergence. This shows the advantage of maintaining multiple good solutions and combining them (GA) over keeping a single solution (SA).

VI. PlaceIt for Heterogeneous Chiplet Shapes

We explain our method to represent and optimize placements of heterogeneously shaped chiplets with arbitrary, rectangular shapes and arbitrary PHY counts and positions.

A. Placement Representation

An intuitive representation of a placement of heterogeneously shaped chiplets would be a list of chiplets where each chiplet has a location and a rotation. However, with such an approach, the `mutate()` and `merge()` operations could result in overlapping chiplets. Only allowing operations that do not result in overlaps would severely restrict the number of operations and prohibit an exploration of the full solution space. Allowing operations that result in overlaps and using a legalization step to make the chiplets non-overlapping again makes the placement quality deteriorate over time (we tried this as our first approach). Therefore, in the remainder of this section, we present our more elaborate placement representation for chiplets with heterogeneous shapes.

Instead of optimizing the location of chiplets directly, we optimize the *order* and *rotations* in which our custom placement algorithm places the chiplets. Like this, every possible combination of *order* and *rotations* results in a placement with non-overlapping chiplets. Our custom chiplet placement algorithm iterates through the chiplets in the specified order and places each of them by performing the following steps:

Step 1: Draw a line along the perimeter of the current placement (see the blue, dashed line in Fig. 7).

Step 2: Identify all L-shaped corners of the perimeter-line (see the red solid corners in Fig. 7).

Step 3: Place the chiplet in the corner, that minimizes the area of a minimum square enclosing the whole placement.

Step 4: The third step can result in overlaps (see Fig. 7c). If the newly placed chiplet has an overlap on the right, move it to the top and vice versa (see Fig. 7d).
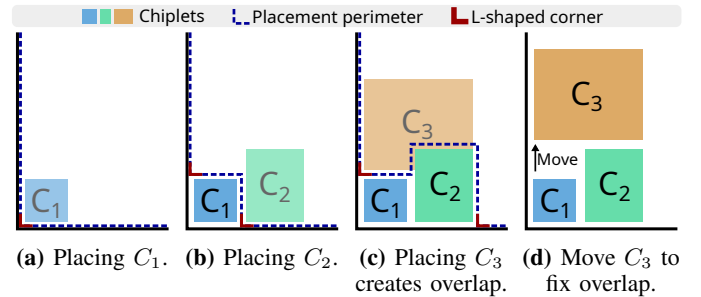

(a) Placing C_1. **(b)** Placing C_2. **(c)** Placing C_3 creates overlap. **(d)** Move C_3 to fix overlap.

Fig. 7: (§VI-A) Our custom placement algorithm.

Notice that the first-class citizen on which the optimization algorithms operate are the *order* and *rotations* in which the chiplets are placed, not the placement itself. Due to this indirection, we need to avoid isomorphic representations (different (order, rotations)-pairs that result in the same placement). If this is not the case, an optimization algorithm might use a set of supposedly different solutions that all result in the same placement. To avoid isomorphic representations, we represent the order by chiplet types and not by chiplet IDs since two different orders by ID can result in equivalent placements, but two different orders by type cannot (see Fig. 8 left). Furthermore, notice that a chiplet can be rotation-invariant, rotation-sensitive, or rotation-hybrid depending on whether the chiplet shape and PHY locations change upon rotation (see Fig. 8 right). To prevent isomorphic representations, we disallow rotations of rotation-invariant chiplets, and we only allow $90°$ rotations for rotation-hybrid chiplets.
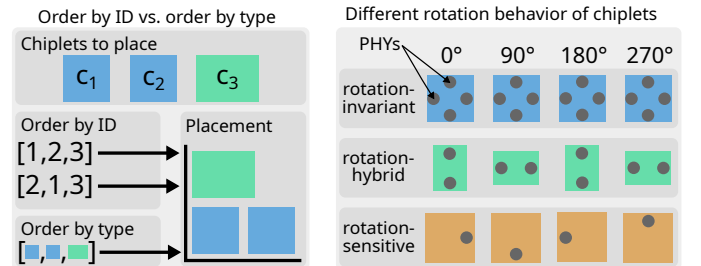

Fig. 8: (§VI-A) Prevent multiple (order, rotations)-pairs from producing the same placement by using order by type and dissallowing rotations based on the rotation behavior.

(a) Input: Placemet. **(b)** Create graph. **(c)** Compute MST. **(d)** Add more edges. **(e)** Output: Topology.
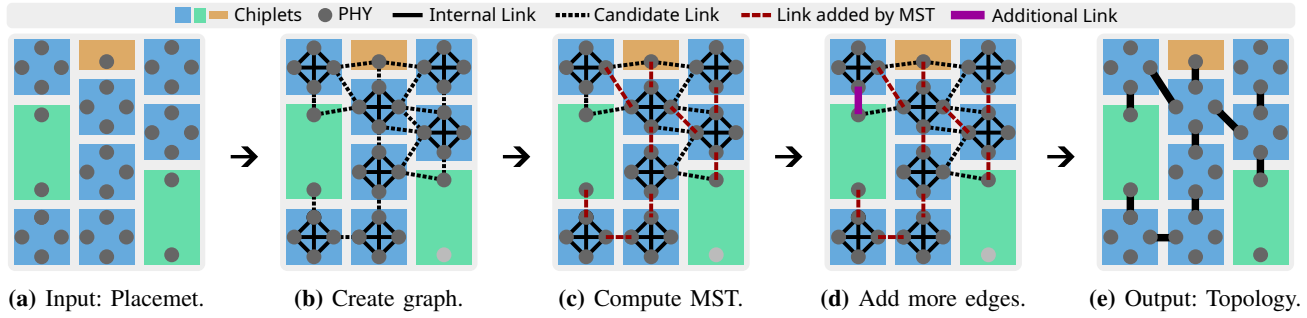
Fig. 9: (§VI-A) Inferring the placement-based ICI topology from our representation for heterogeneous chiplet placements.

Our placement representation for heterogeneous chiplets implements the same five functions as the representation of homogeneous chiplets. Instead of generating, mutating or merging placements directly, we perform these operations on the chiplet *order* and *rotations* (see Fig. 10). To implement the `get_area()` function, we compute the area of a minimal rectangle that encloses the whole placement.
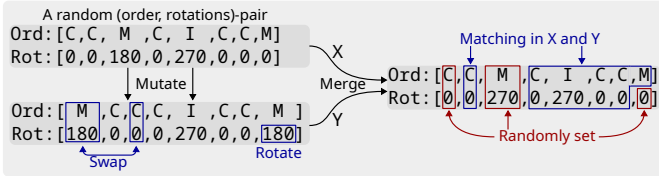

Fig. 10: (§VI-A) Mutate & merge for heterogeneous chiplets. C: Compute-chiplet, M: Memory-chiplet, I: IO-chiplet.

Extracting the placement-based ICI topology from a placement with heterogeneously shaped chiplets is more involved than for the homogeneous case: Given a placement (see Fig. 9a), we create a graph where each vertex represents a PHY. We add "internal" edges between all PHYs of a chiplet with relay-capabilities (the black, solid edges in Fig. 9b). Furthermore, we add a "candidate" edge between any two PHYs of different chiplets that are at most the maximum link length apart (the black, dotted edges in Fig. 9b). We set the weight of "candidate" edges based on the link length and compute a minimum spanning tree (MST) [30] on top of this graph (the red, dashed edges in Fig. 9c). Each edge in the MST corresponds to a D2D link. If some PHYs of otherwise connected chiplets remain unconnected, we ignore them. If a whole chiplet remains unconnected, we abort and the operation that created this placement (random generation, mutate, or merge) is repeated. Next, we iterate through the remaining candidate-edges, ordered by increasing weight. If an edge is connected to two otherwise unused PHYs, we add that edge to the ICI topology (the purple, fat edges in Fig. 9d). Fig. 9e shows the resulting ICI topology.

B. Optimization Results

We run PlaceIT on one core of an Intel Xeon X7550 running Debian 11 to optimize the same two architectures as in Section V-B, but with heterogeneous chiplets (see Fig. 11).
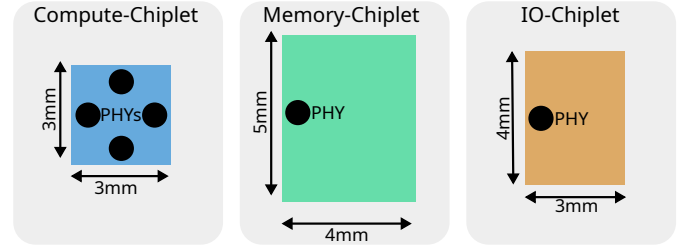

Fig. 11: (§VI-B) Chiplet dimensions and PHY locations.

Since we still want to optimize the ICI topology for cache coherency traffic, we use the same cost function weights as in Section V-B. The remaining parameters are shown in Table IV.

	Parameter	32 cores heterogeneous	64 cores heterogeneous
General	Time Budget	3600 s	3600 s
	Repetitions	10	10
	Norm. Samples	500	500
	Mutation Mode	any-one	any-one
	L_R, L_P, and L_L	10, 12, and 1 cycles	10, 12, and 1 cycles
	Distance Type	Eucledian	Eucledian
	Max. Length	3mm	3mm
GA	Population size (P)	30	20
	Elitism size (E)	6	5
	Tournament size (T)	6	5
	Mutation prob. (p_m)	0.5	0.5
SA	Initial Temp. (T_0)	33	28
	Iterations (L)	50	45
	Cooling param. (α)	1	1
	Adaptive param. (β)	5	5

TABLE IV: (§VI-B) Experiment configuration for heterogeneously shaped chiplets.

Even though the problem of placing heterogeneously shaped chiplets is significantly more complex than that of placing homogeneous ones, we were able the achieve a similar solution space size of approximately 10^{14} solutions for the 32 core architecture and 10^{30} for the 64 core architecture. We achieve this by optimizing the order and rotations, in which our custom placement algorithm places the chiplets, not the chiplet placement itself. Like this, we are able to remove many unfavorable or invalid placements from the solution space. However, it is possible that we also removed some good placements or even the best one. In our opinion, this is not problematic as finding the single best solutions in a solution space of that scale is highly unlikely, and our approach preserves enough good solutions to achieve good results.

Fig. 12 shows our results for the two heterogeneous architectures. All optimization algorithms outperform the 2D mesh baseline (see Fig. 13) and both the GA and SA outperform BR. As in the homogeneous setting, the GA performs better than SA, however, for the 32 core architecture with heterogeneous chiplet shapes, SA performs comparably to the GA.
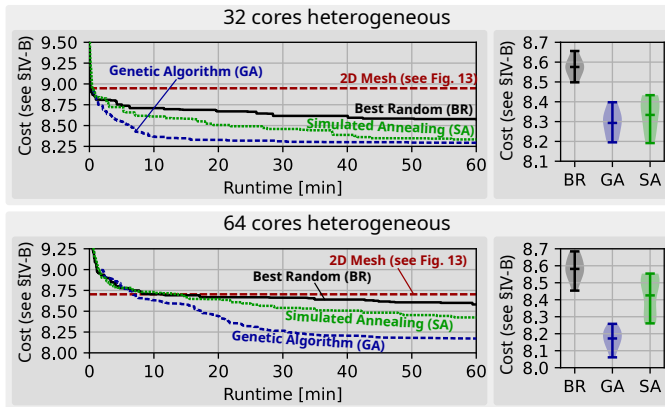

Fig. 12: (§VI-B) Results for heterogeneous chiplet shapes. We show the evolution of the result over time (left) and the distribution of the final result over 10 repetitions (right). See Fig. 13 for the placements found by the best algorithm.

Comparing our results for the heterogeneous setting to those of the homogeneous ones reveals that even though the solution spaces of both settings have the same size, reaching convergence in the heterogeneous setting takes significantly longer than in the homogeneous one. The reason for this is that the heterogeneous setting is more complex: For each new (order, rotations)-pair that is generated, we need to run our placement algorithm and the function to infer the placement-based ICI topology. Therefore, in the heterogeneous setting, the number of placements that an optimization algorithm can create within the time budget is almost an order of magnitude lower than in the homogeneous setting (see Table V).

Algorithm	32 cores Homog.	64 cores Homog.	32 cores Heterog.	64 cores Heterog.
Best Random (BR)	87.0k	17.3k	8.5k	1.2k
Genetic Alforithm (GA)	41.3k	11.5k	8.3k	1.7k
Simulated Annealing (SA)	92.6k	20.4k	14.1k	3.1k

TABLE V: (§VI-B) Number of placements generated by our optimization algorithms for different architectures.

VII. EVALUATION

We evaluate our proposed chiplet placement and ICI topology co-optimization methodology on the two homogeneous architectures from Section V-B and on the two heterogeneous architectures from Section VI-B. For each of these four architectures, we design a baseline architecture consisting of a 2D mesh of compute-chiplets in the center with memory- and IO-chiplets on the perimeter. This type of architecture is the de-facto standard that is used in numerous systems [12], [23], [35], [17], [37]. We perform our evaluation using two different chiplet configurations: In the *baseline* configuration, memory- and IO-chiplets only have a single PHY and they cannot relay messages, which is highly unfavorable for PlaceIT, as PlaceIT often places memory- and IO-chiplets in the center of the chip (off-chip links of IO-chiplets are routed to the border on the redistribution layer as in AMD's EPYC and Ryzen [31]). In the *PlaceIT* configuration, all chiplets have four PHYs and relay capability. To ensure a fair comparison, the total memory- and IO-bandwidth stays unchanged and the increased off-chiplet bandwidth due to additional PHYs is only used to relay messages. Fig. 13 shows baselines and optimized architectures for the *baseline* configuration. Unfortunately, a direct comparison to prior work (see Section VIII) is infeasible since frameworks to optimize the placement are not open-source, or they do not scale to our chiplet counts, and proposals for ICI topologies on active interposers are not applicable to passive interposers, silicon bridges, and organic substrates.

A. Evaluation Methodology

We use RapidChiplet's [19] feature to run simulations in BookSim2 [20] using synthetic traffic and application traces from Netrace [14]. BookSim2 is an established, cycle-accurate network-on-chip (NoC) simulator and Netrace is a tool for dependency-driven, trace-based NoC simulations. We use the Netrace trace collection [13], which is based on the PARSEC benchmark suite [5]. Each trace is split into five regions (see Table VI). Since these traces span across billions of cycles, simulating them in a cycle accurate simulator is extremely time-consuming. The *blackscholes_64c_simsmall* trace was the only one to terminated within 24 hours, therefore, for the remaining traces, we only simulate the first 1'000'000 cycles of each region. All traces contain cache coherency traffic between the L1 cache (mapped to compute-chiplets), the L2 cache (mapped to memory-chiplets), and the main memory (mapped to IO-chiplets).

Trace	Region 1			Region 2			Region 3			Region 4			Region 5		
	P	C	I	P	C	I	P	C	I	P	C	I	P	C	I
blackscholes_64c_simsmall	189k	5.6M	0.0337	1.2M	219M	0.0056	4.9M	75M	0.0655	195k	10M	0.0019	129k	5.7M	0.0228
bodytrack_64c_simlarge	189k	5.6M	0.0337	30M	654M	0.0453	355M	3.9B	0.0914	429k	24M	0.0176	161k	5.7M	0.0283
canneal_64c_simmedium	189k	5.6M	0.0337	240M	20B	0.0121	74M	300M	0.2473	58M	2.9B	0.0198	133k	5.7M	0.0235
dedup_64c_simmedium	189k	5.6M	0.0337	37M	838M	0.0201	379M	2.6B	0.1477	16M	1.0B	0.0153	160k	5.7M	0.0282
ferret_64c_simmedium	189k	5.6M	0.0337	8.6M	648M	0.0133	273M	7.5B	0.0365	5.8M	145M	0.0402	220k	5.7M	0.0387
fluidanimate_64c_simsmall	189k	5.6M	0.0337	6.8M	777M	0.0087	21M	499M	0.0420	6.1M	599M	0.0103	139k	5.7M	0.0245
swaptions_64c_simlarge	189k	5.6M	0.0337	247k	9.7M	0.0254	310M	1.7B	0.1800	194k	14M	0.0141	113k	5.7M	0.0199
x264_64c_simsmall	189k	5.6M	0.0337	1.8M	82M	0.0220	31M	1.5B	0.0212	102M	12B	0.0084	129k	5.7M	0.0227

TABLE VI: (§VII-A) Trace-regions used in the evaluation. P: Number of packets, **C**: number of cycles, **I**: injection rate.
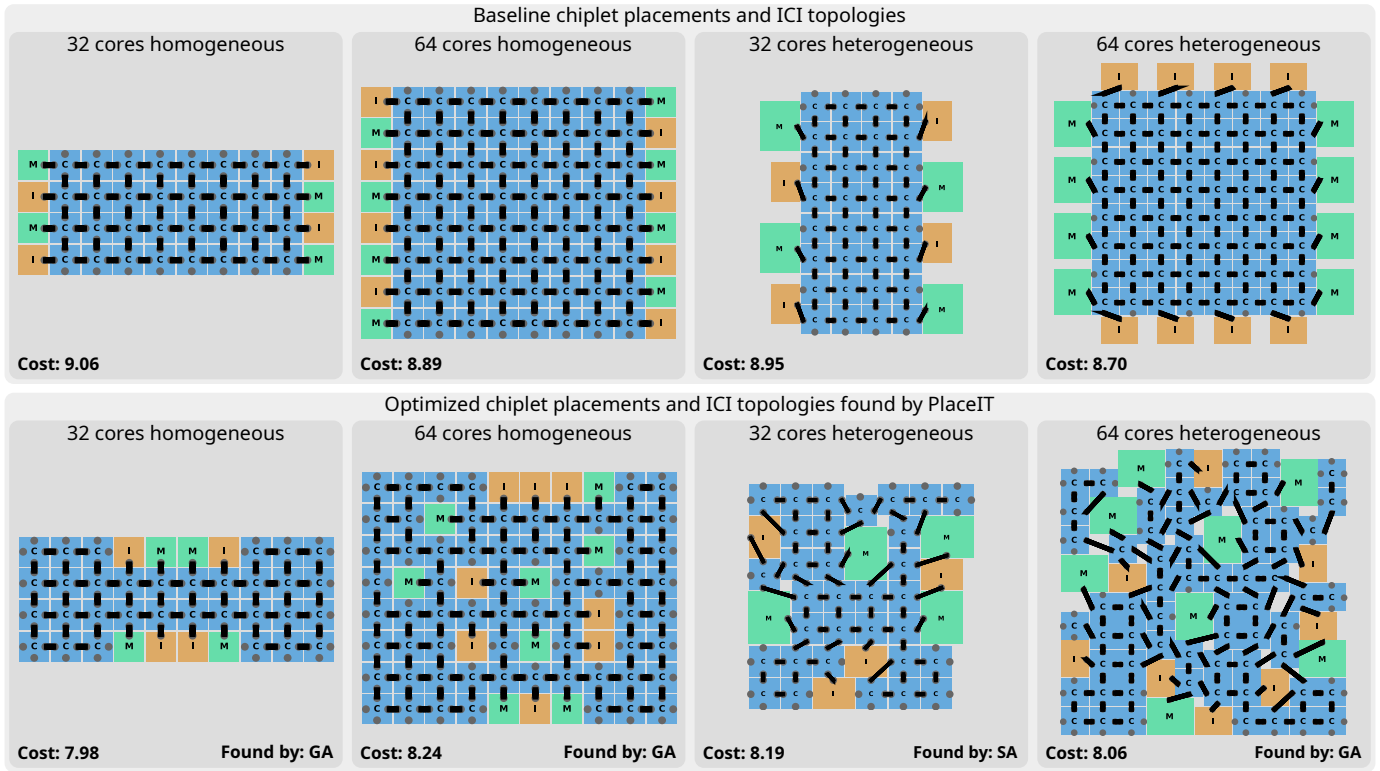
Fig. 13: (§VII) **Baseline architecture and optimized architecture found by PlaceIT (for the *baseline* configuration).**

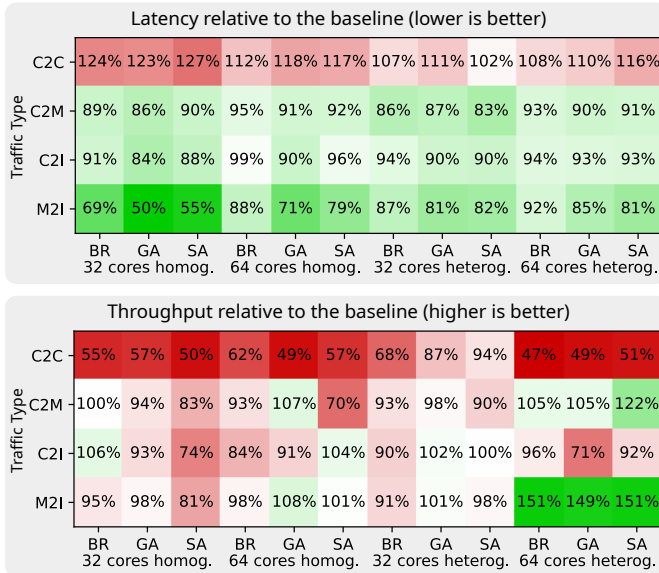**Fig. 14:** (§VII-B) **Results on synthetic traffic using the *baseline* configuration.**

Fig. 15: (§VII-B) **Results on synthetic traffic using the *PlaceIT* configuration.**

We set the parameters of RapidChiplet and BookSim2 to match the latencies described in Tables III and IV. BookSim2 models input-queued virtual channel (VC) routers with a four-stage pipeline (routing, VC allocation, switch allocation, crossbar traversal) and wormhole flow control. We use 1-flit packets for control messages and 9-flit packets for data transfers [15]. Furthermore, we use shortest path routing, up to 8 virtual channels, and 8-flit buffers.
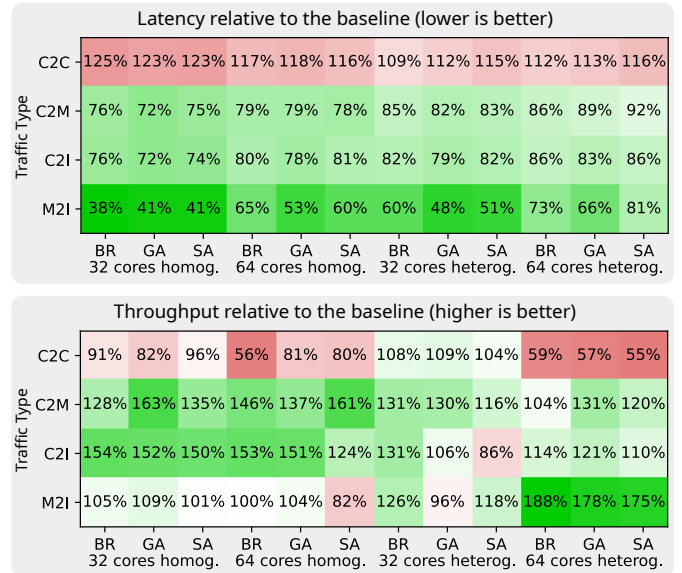
B. Performance Comparison using Synthetic Traffic

We compare our optimized ICI topologies against the baselines in terms of latency and throughput using synthetic C2C, C2M, C2I, and M2I traffic. The advantage of synthetic traffic over real traces is its generality, as synthetic traffic does not depend on the application. Figs. 14 and 15 show the latency and throughput results under synthetic traffic for the *baseline* and *PlaceIT* chiplet configurations, respectively.
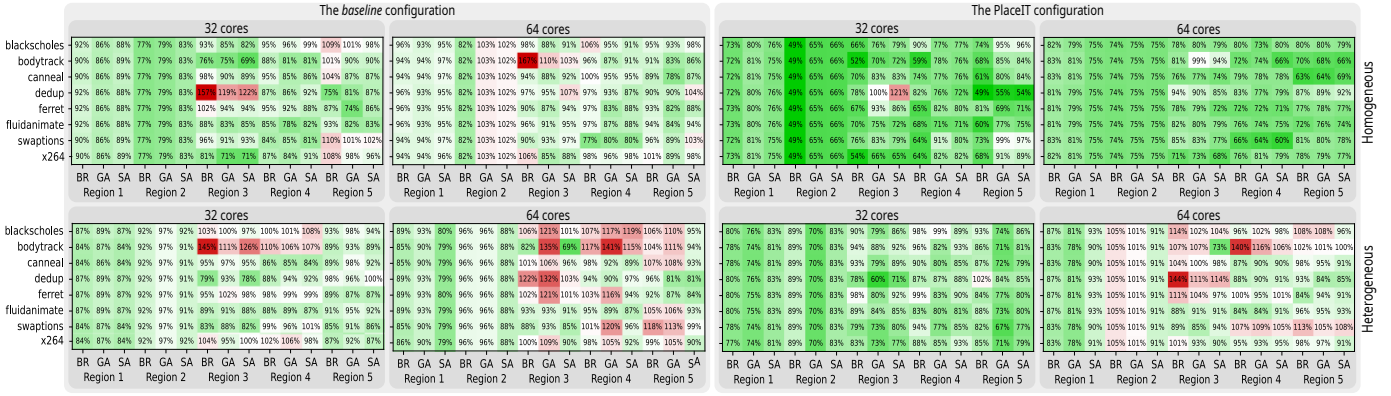
Fig. 16: (§VII-D) Results for the partial trace regions: speedup in average packet latency compared to the baseline.

Recall that our primary optimization goal was to minimize C2M and M2I latency and to improve C2M and M2I throughput. We observe that for all combinations of architecture and optimization algorithm, PlaceIT improves C2M, C2I, and M2I latency. The fact that the baseline provides the best C2C latency is not surprising, given that in the baseline, compute-chiplets form a regular grid with a 2D mesh topology.

PlaceIT is only able to significantly outperform the baseline architecture in terms of C2M and M2I throughput if we use the *PlaceIT* chiplet configuration, where memory- and IO-chiplets have four PHYs and relay-capabilities. The *baseline* configuration with only a single PHY per memory- and IO-chiplet turns out to be too restrictive to provide significant throughput improvements.

Fig. 17: (§VII-C) speedup over baseline in average packet latency (blackscholes trace, *baseline* configuration).

Fig. 18: (§VII-C) speedup over baseline in average packet latency (blackscholes trace, *PlaceIT* configuration).

C. Performance Comparison on Full Traffic Trace

We evaluate the performance of our optimized ICI topologies using the full blackscholes-trace (see Table VI). We simulate this trace in two different simulation modes: In the *authentic* mode, a packet is only injected if all dependencies are satisfied and the cycle, in which the packet appears in the trace, is reached. This represents a scenario where after receiving a packet, the compute cores need some time to perform computations before injecting the next packet. The second mode is called *idealized* and it injects a packet as soon as all dependencies are satisfied, assuming ideal cores that perform computations instantly. This mode is intended as a stress-test for the ICI as the packet injection rate is significantly higher than in the *authentic* mode. Our results in Figs. 17 and 18 show that PlaceIT is able to achieve speedups in average packet latency of up to $1.17\times$ (for the *baseline* configuration) and $1.34\times$ (for the *PlaceIT* configuration).

D. Performance Comparison on Partial Traffic Traces

Fig. 16 shows our results for the simulation of partial trace regions. PlaceIT is able to reduce the average packet latency to 92% (*baseline* configuration) and 82% (*PlaceIT* configuration) on average. In Sections V-B and VI-B we observed that the GA performed significantly better than BR with respect to the minimization of the cost function. However, in our partial trace simulation, we see that this is not always the case and in some instances, BR is even better than the GA. This shows that either our performance estimate or our cost function does not fully reflect the performance on real traces. Nevertheless, co-optimizing the chiplet placement and ICI topology works, as we outperform the baseline architecture in almost all cases.

E. Area Comparison

The area of all homogeneous placements for a given architecture is identical, therefore, we only discuss the area of heterogeneous placements. For the 32-core architecture, BR and SA increase the area by 5.4% and 0.8% respectively, but the GA reduced the area by 8.1% compared to the baseline. For the 64-core architecture, BR and SA both increase the area by 3.3% but the GA reduced the area by 6.3% compared to the baseline. We conclude that PlaceIT is able to improve the ICI-performance without introducing significant area overheads.
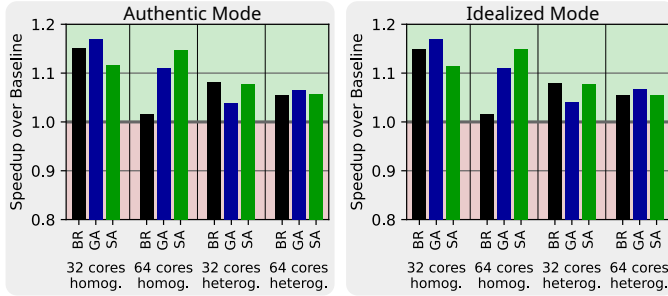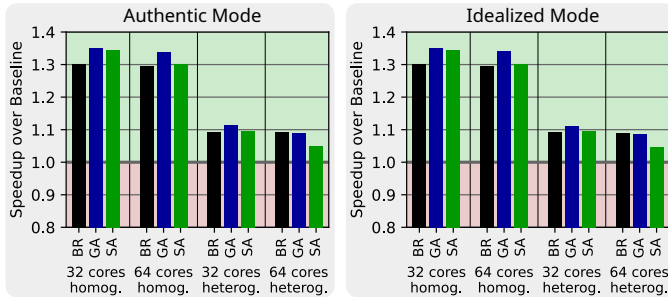
Name	Target Technology	Heterogeneous Chiplet Shapes	Optimized Placement	Optimized Topology	Target Metric	Optimization Method	Approximate Runtime
Ho et al. [16]	PSI	✔	✔	✗	TWL, A	HB*-tree, SA	8 min (4 chiplets)
Liu et al. [27]	PSI	✔	✔	✗	TWL	Enumeration, Flow	3 h (8 chiplets)
Seemuth et al. [33]	PSI	✔	✔	✗	TWL, A	SA	⑦
Eris et al. [11]	PSI	✗	✔	✗	P, C	Greedy	450 h (16 chiplets)
Osmolovskyi et al. [32]	PSI	✔	✔	✗	TWL	B&B, CSP	1 h (11 chiplets)
Coskun et al. (1) [9]	PSI	✗	✔	(✔)*	P, C, TWL	MILP, SA	⑦
Coskun et al. (2) [8]	PSI	✗	✔	(✔)*	P, C, T	MILP, SA	⑦
Tap-2.5D [28]	PSI	✔	✔	✗	TWL, T	MILP, SA	25 h (8 chiplets)
Chiou et al. [7]	PSI	✔	✔	✗	TWL, T	B&B, Pruning	6 min (11 chiplets)
ButterDonut [21]	ASI	✗	✗	✔	P	Construction	n/a
ClusCross [34]	ASI	✗	✗	✔	P	Construction	n/a
Kite [4]	ASI	✗	✗	✔	P	Construction	n/a
HexaMesh [18]	OPS, PSI	✗	✔	✔	P	Construction	n/a
PlaceIT (This Work)	OPS, SB, PSI	✔	✔	✔	P, A, (TWL)	SA, GA	1 h (73 chiplets)

TABLE VII: (§VIII) Overview of related work. Target technology: **OPS**: Organic package substrate, **SB**: Silicon bridge, **PSI**: Passive silicon interposer, **ASI**: Active silicon interposer. Target metric: **TWL**: Total wire length, **T**: Temperature, **P**: Performance, **C**: Cost, **A**: Area. Optimization method: **SA**: Simulated annealing, **B&B**: Branch & bound, **CSP**: Constraint-satisfaction problem, **MILP**: Mixed integer-linear problem, **GA**: Genetic algorithm. *Select best topology out of 8 candidates.

VIII. RELATED WORK

Multiple recent studies have focused on placing chiplets on a passive silicon interposer. These works usually assume that the ICI topology is given as an input. Most of them optimize the total wire length (TWL) [16], [27], [33], [32] or a combination of TWL and thermal properties [7], [28]. Some also consider the ICI performance or the cost [11]. Many of these works can be combined with PlaceIT. We could, e.g., use PlaceIT to find a placement and ICI topology, and then apply TAP-2.5D [28] to fine-tune the placement for thermal properties.

An interesting line of work is that of Coskun et al. [9], [8]. They apply a cross-layer co-optimization approach to jointly optimize a 2.5D stacked chip across the logical-, physical- and circuit layer. They consider a predetermined set of well-known ICI topologies out of which they select the most suitable one. This is in contrast to PlaceIT, where completely new ICI topologies are created. We see potential in combining the two approaches by first finding an ICI topology and placement using PlaceIT and then applying the cross-layer co-optimization approach to optimize the remaining layers or to select the placement found by either BR, the GA or SA.

Research on ICI topologies focuses on active interposers, since they offer longer links and package-level routers. Such works assume a regular 2D grid of compute-chiplets with memory- or IO-chiplets on the side. ICI topologies such as ButterDonut [21], ClusCross [34], or Kite [4] are optimized for low ICI latency and high ICI throughput.

One of the few works focussing on ICI topologies for passive silicon interposers is HexaMesh [18]. They propose a hexagonal arrangement of chiplets where each non-border chiplet has six D2D links to other chiplets. However, this approach is only applicable to homogeneous architectures.

PlaceIT is the first work known to us that jointly optimizes ICI topology and chiplet placement. Furthermore, it is the first work on ICI topologies for heterogeneously shaped chiplets. Table VII compares PlaceIT to its related work.

IX. CONCLUSION

In this work, we present PlaceIT, a novel methodology to jointly optimize the chiplet placement and ICI topology for chips with heterogeneous chiplet shapes and silicon bridges or passive silicon interposers. The main novelty of our approach is that we perform optimization on the chiplet placement itself, where we infer a custom, placement-based ICI topology for each placement produced by an optimization algorithm. We use the placement and its inferred ICI topology to compute proxies for ICI latency and throughput of different traffic types, which we combine into a user-defined quality metric that is returned to the optimization algorithm.

The open-source PlaceIT framework is modular and allows adding custom optimization algorithms or placement representations. PlaceIT offers a wide range of configurable parameters, making it applicable for a variety of designs with different chiplet dimensions, PHY-counts, and D2D links.

Our evaluation on synthetic traffic shows that PlaceIT produces ICIs with vastly lower C2M, C2I, and M2I latency (reduced by up to 62%) compared to a 2D mesh baseline. On real traffic traces, PlaceIT reduces the average packet latency in almost all traces and architectures considered. The average packet latency is reduced by up to 18% on average.

By using our open-source PlaceIT framework, architects can co-optimize their chiplet-placement and ICI topology to build 2.5D stacked chips with low-latency interconnects.

REFERENCES

[1] E. Alon, M. Hempel, K. Poulton, S. Ardalan, and B. Vinnakota, "Bunch of Wires (BoW) PHY Specification," https://opencomputeproject.github.io/ODSA-BoW/bow_specification.html.

[2] S. Ardalan, H. Cirit, R. Farjad, M. Kuemerle, K. Poulton, S. Subramanian, and B. Vinnakota, "Bunch of wires: An open die-to-die interface," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2020, pp. 9–16.

[3] B. Banijamali, C.-C. Chiu, C.-C. Hsieh, T.-S. Lin, C. Hu, S.-Y. Hou, S. Ramalingam, S.-P. Jeng, L. Madden, and D. C. Yu, "Reliability evaluation of a cowos-enabled 3d ic package," in *2013 IEEE 63rd Electronic Components and Technology Conference*. IEEE, 2013, pp. 35–40.

[4] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna, "Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[6] H. Braunisch, A. Aleksov, S. Lotz, and J. Swan, "High-speed performance of silicon bridge die-to-die interconnects," in *2011 IEEE 20th Conference on Electrical Performance of Electronic Packaging and Systems*. IEEE, 2011, pp. 95–98.

[7] H.-W. Chiou, J.-H. Jiang, Y.-T. Chang, Y.-M. Lee, and C.-W. Pan, "Chiplet placement for 2.5 d ic with sequence pair based tree and thermal consideration," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 7–12.

[8] A. Coskun, F. Eris, A. Joshi, A. B. Kahng, Y. Ma, A. Narayan, and V. Srinivas, "Cross-layer co-optimization of network design and chiplet placement in 2.5-d systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5183–5196, 2020.

[9] A. Coskun, F. Eris, A. Joshi, A. B. Kahng, Y. Ma, and V. Srinivas, "A cross-layer methodology for design and optimization of networks in 2.5 d systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[10] P. Coudrain, J. Charbonnier, A. Garnier, P. Vivet, R. Velard, A. Vinci, F. Ponthenier, A. Farcy, R. Segaud, P. Chausse, L. Arnaud, D. Lattard, E. Guthmuller, G. Romano, A. Gueugnot, F. Berger, J. Beltritti, T. Mourier, M. Gottardi, S. Minoret, C. Ribiere, G. Romero, P.-E. Philip, Y. Exbrayat, D. Scevola, D. Campos, M. Argoud, N. Allouti, R. Eleouet, C. F. Tortolero, C. Aumont, D. Dutoit, C. Legall, J.Michailos, S.Cheramy, and G. Simon, "Active interposer technology for chiplet-based advanced 3d system architectures," in *2019 IEEE 69th Electronic Components and Technology Conference (ECTC)*. IEEE, 2019, pp. 569–578.

[11] F. Eris, A. Joshi, A. B. Kahng, Y. Ma, S. Mojumder, and T. Zhang, "Leveraging thermally-aware chiplet organization in 2.5 d systems to reclaim dark silicon," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1441–1446.

[12] R. Guirado, H. Kwon, S. Abadal, E. Alarcón, and T. Krishna, "Dataflow-architecture co-design for 2.5 d dnn accelerators using wireless network-on-package," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 806–812.

[13] J. Hestness, B. Grot, and S. W. Keckler, "Netraces v1.0 (A collection of network traces with dependency information)." https://www.cs.utexas.edu/~netrace/.

[14] ——, "Netrace: dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, 2010, pp. 31–36.

[15] J. Hestness and S. W. Keckler, "Netrace: Dependency-tracking traces for efficient network-on-chip experimentation," *The University of Texas at Austin, Dept. of Computer Science, Tech. Rep*, 2011.

[16] Y.-K. Ho and Y.-W. Chang, "Multiple chip planning for chip-interposer codesign," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

[17] Z. Huang, S. Fan, C. Tang, X. Lin, S. Deng, and Y. Liu, "Hecaton: Training and finetuning large language models with scalable chiplet systems," *arXiv preprint arXiv:2407.05784*, 2024.

[18] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler, "Hexamesh: Scaling to hundreds of chiplets with an optimized chiplet arrangement," *arXiv preprint arXiv:2211.13989*, 2022.

[19] P. Iff, B. Bruggmann, M. Besta, L. Benini, and T. Hoefler, "Rapidchiplet: A toolchain for rapid design space exploration of chiplet architectures," 2023.

[20] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2013, pp. 86–96.

[21] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Proceedings of the 48th international symposium on Microarchitecture*, 2015, pp. 546–558.

[22] J. Kim, V. C. K. Chekuri, N. M. Rahman, M. A. Dolatsara, H. Torun, M. Swaminathan, S. Mukhopadhyay, and S. K. Lim, "Silicon vs. organic interposer: Ppa and reliability tradeoffs in heterogeneous 2.5 d chiplet integration," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 80–87.

[23] S. Kim, J. Kim, J. Choi, and J. Ho Ahn, "Cifher: A chiplet-based fhe accelerator with a resizable structure," *arXiv preprint arXiv:2308.04890*, 2023.

[24] J. H. Lau and J. H. Lau, "Chiplet heterogeneous integration," *Semiconductor Advanced Packaging*, pp. 413–439, 2021.

[25] H.-J. Lee, R. Mahajan, F. Sheikh, R. Nagisetty, and M. Deo, "Multi-die integration using advanced packaging technologies," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2020, pp. 1–7.

[26] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun, "Chiplet heterogeneous integration technology-status and challenges," *Electronics*, vol. 9, no. 4, p. 670, 2020.

[27] W.-H. Liu, M.-S. Chang, and T.-C. Wang, "Floorplanning and signal assignment for silicon interposer-based 3d ics," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[28] Y. Ma, L. Delshadtehrani, C. Demirkiran, J. L. Abellan, and A. Joshi, "Tap-2.5 d: A thermally-aware chiplet placement methodology for 2.5 d systems," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1246–1251.

[29] R. Mahajan, R. Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik, "Embedded multi-die interconnect bridge (emib)–a high density, high bandwidth packaging interconnect," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*. IEEE, 2016, pp. 557–565.

[30] M. Mareš, "The saga of minimum spanning trees," *Computer Science Review*, vol. 2, no. 3, pp. 165–221, 2008.

[31] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 57–70.

[32] S. Osmolovskyi, J. Knechtel, I. L. Markov, and J. Lienig, "Optimal die placement for interposer-based 3d ics," in *2018 23rd Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2018, pp. 513–520.

[33] D. P. Seemuth, A. Davoodi, and K. Morrow, "Automatic die placement and flexible i/o assignment in 2.5 d ic design," in *Sixteenth International Symposium on Quality Electronic Design*. IEEE, 2015, pp. 524–527.

[34] H. Shabani and X. Guo, "Cluscross: a new topology for silicon interposer-based network-on-chip," in *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, 2019, pp. 1–8.

[35] Y. S. Shao, J. Cemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. Gray, B. Khailany, and S. W. Keckler, "Simba: scaling deep-learning inference with chiplet-based architecture," *Communications of the ACM*, vol. 64, no. 6, pp. 107–116, 2021.

[36] K. Sikka, R. Bonam, Y. Liu, P. Andry, D. Parekh, A. Jain, M. Bergendahl, R. Divakaruni, M. Cournoyer, P. Gagnon, C. Dufort, I. de Sousa, H. Zhang, E. Cropp, T. Wassick, H. Mori, and S. Kohara, "Direct bonded heterogeneous integration (dbhi) si bridge," in *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*. IEEE, 2021, pp. 136–147.

[37] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla's exa-scale computer," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–28.

[38] The UCIe Consortium, "Universal Chiplet Interconnect Express (UCIe) Specification," https://www.uciexpress.org/specification.

[39] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, Senior, C. Fuguet, I. Miro-Panades, G. Moritz, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harr, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, and Q. L. M. and, "IntAct: A 96-core processor with six chiplets 3D-stacked on an active interposer with distributed interconnects and integrated power management," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 79–97, 2020.