GIA: A Reusable General Interposer Architecture for Agile Chiplet Integration

Fuping Li[1,3], Ying Wang[1,3,6], Yuanqing Cheng[4], Yujie Wang[2,3,6], Yinhe Han[2,3,6], Huawei Li[1,3,5], Xiaowei Li[1,3]

[1]SKLP, Institute of Computing Technology, Chinese Academy of Sciences
[2]CICS, Institute of Computing Technology, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences, [4]Beihang University
[5]Peng Cheng Laboratory, [6]Zhejiang Laboratory

ABSTRACT

2.5D chiplet technology is gaining popularity for the efficiency of integrating multiple heterogeneous dies or chiplets on interposers, and it is also considered an ideal option for agile silicon system design by mitigating the huge design, verification, and manufacturing overhead of monolithic SoCs. Although it significantly reduces development costs by chiplet reuse, the design and fabrication of interposers also introduce additional high non-recurring engineering (NRE) costs and development cycles which might be prohibitive for application-specific designs having low volume.

To address this challenge, in this paper, we propose a reusable general interposer architecture (GIA) to amortize NRE costs and accelerate integration flows of interposers across different chiplet-based systems effectively. The proposed assembly-time configurable interposer architecture covers both active interposers and passive interposers considering diverse applications of 2.5D systems. The agile interposer integration is also facilitated by a novel end-to-end design automation framework to generate optimal system assembly configurations including the selection of chiplets, inter-chiplet network configuration, placement of chiplets, and mapping on GIA, which are specialized for the given target workload. The experimental results show that our proposed active GIA and passive GIA achieve 3.15x and 60.92x performance boost with 2.57x and 2.99x power saving over baselines respectively.

KEYWORDS

2.5D integration, chiplet, interposer, network-on-chip, reusability

1 INTRODUCTION

In post-Moore era, the design and manufacturing costs of conventional monolithic 2D ICs are rocketing due to the growing scale and complexity of modern chip architecture. As an alternative to 2D ICs, chiplet-based 2.5D integration is proposed in recent years to promote the design and fabrication productivity in ever-increasing SoC design. With this technology, reusable component dies can be chosen and stacked on a silicon interposer which serves as an integration carrier and interconnect fabric. In fact, the benefits of chiplet technology have been justified by several commercial 2.5D products such as AMD EPYC [21], Intel Agilex [8], and Huawei Kunpeng 920 [35].

First of all, the decomposition of large dies into smaller ones substantially reduces costs due to higher yield and better wafer area utilization[28]. Chiplet technology also enables heterogeneous process integration to save fabrication costs further [21]. For instance, the performance-critical components such as computing units can be built with expensive advanced processes while others can use mature and cost-effective processes. Since chiplets can be reused in various systems without repaying one-time costs like verification and masks, the NRE can be amortized efficiently. Also, chiplet reuse also facilitates and promotes agile hardware design. Thanks to chiplet integration, designers only need to focus on dies with their own customized logic. This LEGO-style design flow helps designers out of time-consuming low-level IP integration and greatly accelerates chip development.

However, if not considered properly in chiplet-based system design, expenses of interposers might swallow these benefits brought by chiplet reuse. On the one hand, the design and manufacture of interposers introduce NRE costs including IP licensing, verification, masks, etc. For active interposers, it can take millions of dollars even built by mature process [9], which is unaffordable for low-volume products. On the other hand, the cumbersome design flow of interposers especially active interposers [24] takes a lot of time that might slow down the chip development. To mitigate costs and shorten the development cycle, it is expected to reuse interposers like chiplets and realize the agile chip development that incurs only slight assembly overhead. Unfortunately, three major obstacles are limiting the reusability of interposers. Firstly, the interposer should accommodate flexible placements of chiplets with different sizes and quantities. Secondly, diverse application-specific communication patterns require flexible configurations of inter-chiplet interconnect topologies and routing. Thirdly, designers need a design automation framework to facilitate the design space exploration to shorten chip product time-to-market.

In this paper, to address these challenges, we propose a reusable interposer architecture, i.e. **GIA**. Based on GIA, we build an end-to-end design automation framework generating optimal assembly configurations for user applications, to speed up the chiplet integration process. The main contributions of this paper are as follows:

- We propose an assembly-time configurable interposer architecture applicable to both active and passive interposers. The proposed interposer inherently provides reusability targeting diverse applications and different scales of systems.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3549464

- We present a design automation framework to select chiplets, generate inter-chiplet network, place chiplets, and perform GIA mapping effectively.
- The experimental results show that our proposed active GIA and passive GIA achieve 3.15x and 60.92x performance boost with 2.57x and 2.99x power saving over baselines respectively.

The rest of this paper is organized as follows. Section 2 discusses the background and related work. We illustrate the proposed GIA and design automation framework in Section 3 and 4 respectively. Section 5 presents extensive experimental results. Finally, Section 6 concludes this paper.

2 BACKGROUND AND RELATED WORK

2.1 Silicon Interposer for 2.5D Chiplet Design

Breaking a large monolithic chip into small functional dies named *chiplets* alleviates the issues of escalating chip development costs. As shown in Fig. 1(a), a chiplet-based system consists of chiplets bonded to a silicon interposer side-by-side. In this system, microbumps provide inter-chiplet connectivity, mechanical support, and off-package signals or power from C4 bumps [11].

There are two types of silicon interposers[28]: one is the passive interposer containing metal wires only, and the other is the active interposer consisting of not only wires but also CMOS transistors. Active interposers are more expensive due to the complex fabrication process and can integrate repeaters and flip-flops to increase link frequencies. Interposer can adopt network-on-chip (NoC) for inter-die communication and flexible integration of third-party chiplets, and the inter-chiplet network consists of network interfaces (NIs) and routers. NIs located in chiplets are responsible for converting chiplet messages to network packets. Routers are used to multiplex multiple communication flows over the interconnect. For active interposers, routers are fabricated in the interposers, while routers are moved into chiplets for passive interposers due to the lack of active devices.

2.2 Reusable Interposer

Some attempts have been made to improve the reusability of interposers in prior works. The FSMC chiplet reuse scheme in [7] builds standardized sockets on the interposer and plugs chiplets into it. This method is strictly limited to chiplets owning the same footprint and only allows fixed placement. Silicon Interposer Service Layer (SISL) proposed in [27, 34] contains on-chip networks and can be employed to connect dies. It improves the interposer reusability over FSMC by removing restrictions on chiplet sizes and placements. However, the established inter-chiplet network in SISL has fixed topology and encounters performance issues when dealing with varied workloads. SiPterposer in [5] is a fully-passive structure and can be configured into any custom interconnect topology at the chiplet assembly time, but this design also faces performance issues caused by underutilized wire resources. Thus, the above methods cannot satisfy performance demands of diverse applications that require heterogeneous chiplet integration. In this work, our proposed GIA supports flexible placements of varied-footprint chiplets and provides a configurable high-performance interconnect fabric for various applications.

From the perspective of interconnection on interposers, [15] examined various inter-chip network architecture and developed the ButterDonut topology to improve system performance and also enhance the reusability of interposer. Also, the inter-chiplet connection issue is sometimes highly correlated with placement, [6] proposed a thermally-aware chiplet placement algorithm. In addition to single-metric optimization, the cross-layer design methodology in [3] jointly optimized the inter-chiplet network topology, chiplet placement as well as inter-chiplet links. However, these works focus mostly on homogeneous multi-processors with relatively deterministic architecture and have limited support for heterogeneous systems customized for diverse application scenarios.

3 GENERAL INTERPOSER ARCHITECTURE FOR AGILE CHIPLET INTEGRATION

In this section, we first illustrate the basic concept of the proposed GIA. Then we discuss the configurable inter-chiplet network in GIA. Lastly, we depict the detailed process of GIA configuration at the chiplet assembly time.

3.1 Tile-Based Interposer Structure

To support flexible placements of chiplets with different sizes, we propose a novel strategy to organize resources in a modular tile-based form. As shown in Fig. 1(b), chiplets and interposers are divided into multiple tiles with the same size. Each tile contains a group of predefined connections via microbumps between the chiplet and the interposer responsible for the power supply, clock delivery, external IO connectivity, and inter-chiplet communication. With this modular design, a chiplet can plug into any part of free tiles of the interposer. Note that by orchestrating the signal layout of tiles, the chiplet has full access to a portion of power/clock/IO without being resized to cover tiles fully. We take the scheme as mentioned in [5, 16, 22, 34] for power, clock, and external IO delivery. In the following, we elaborate on the inter-chiplet signal transmission.

As depicted in Fig. 1(b), there is one NI in the tile of the chiplet responsible for inter-die communication, and it can provide bandwidth of link width $\times$ clock frequency. The number and positions of NIs are decided by chiplet vendors according to design constraints such as chiplet bandwidth demand. Routers are built in the interposer and are connected through horizontal and vertical wires which contain repeaters to decrease wire delay for active GIA. For passive GIA, routers are moved into chiplets and the interposer only contains disconnected interconnect metal wires.

3.2 Configurable Inter-Chiplet Network

Both network topology and routing strategy have great impacts on performance of the system [12]. For application-specific interconnection networks, topologies and routing are normally optimized for target workloads. To support topology and routing customization, we use the configurable inter-chiplet network that relies on configurable routers and NIs. Active GIA and passive GIA have the same inter-chiplet network architecture design except the locations of routers. An exemplary overview of the inter-chiplet network in active GIA is shown in Fig. 1(d) with the corresponding topology in Fig. 1(c).

The router structure in GIA interconnection is similar to the silicon proof SMART router in [2] and optimized for the scenario of interposer integration. There are two functions of routers in GIA. The first one is multiplexing traffic flows, for example, router-B in Fig. 1(d) forwards packets from NI-5 to NI-3 and packets from NI-5 to NI-4 to distinct output channels. The second function is passing packets along the preset paths, e.g. router-C in Fig. 1(d)

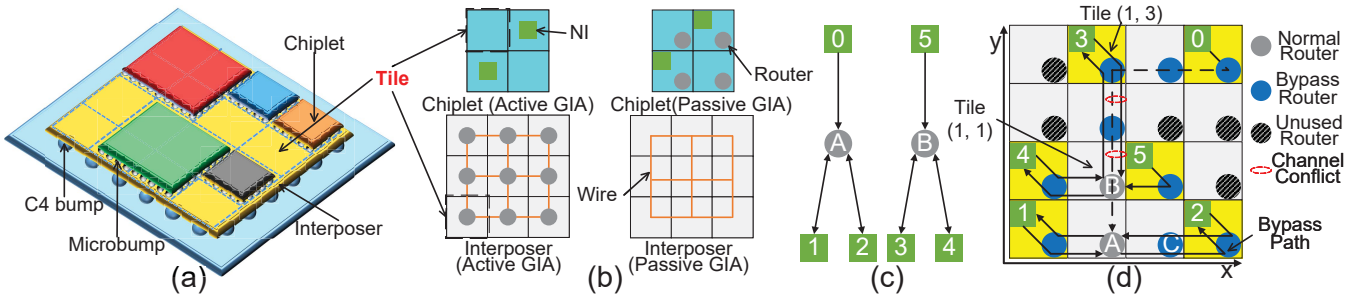

Figure 1: (a) a chiplet-based system built on GIA. (b) tile-based active GIA and passive GIA. (d) an example of configurable inter-chiplet network in active GIA of six chiplets with topology in (c).

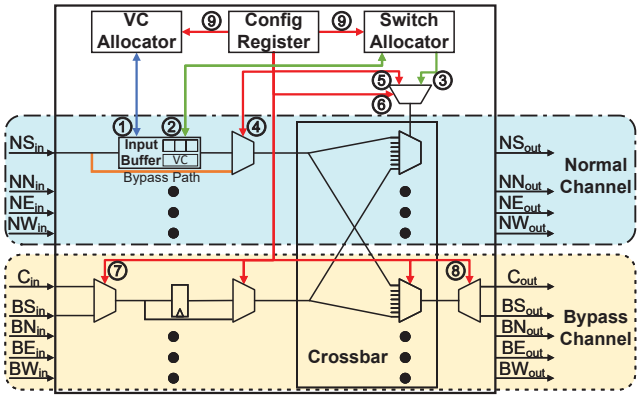

Figure 2: Router microarchitecture of GIA.

transmits packets from NI-2 to router-A directly and does not play as a router of network topology. To realize this feature, we use normal channels controlled by the configuration register in router, i.e. *NS/NN/NE/NW* in Fig. 2, where *NS* denotes the normal south channel for example. Firstly, a normal channel can be configured to a canonical four-stage pipeline [12]. In the stage of buffer write, the incoming flit is buffered in the input virtual channel (VC), and an output port is chosen according to the route preset by its header. In the stage of VC allocation, the header flit arbitrates for a VC of the next router (controlled by ①). In the stage of switch allocation, the buffered flit arbitrates for the switch input and output ports after winning a VC (controlled by ②). Upon successful arbitration, in the stage of crossbar/link traversal, the flit traverses the crossbar and output link to the downstream node (controlled by ③). Secondly, the normal channel can be set to bypass the buffering and allocation stages with *bypass path* (controlled by ④). The flit traverses the crossbar and goes to the preset output port directly (controlled by ⑤ and ⑥) [2].

One important constraint of GIA configuration is that one channel can be assigned to only one link of network topology. For example, conflicts of two channels occur in Fig. 1(d) if we use dotted paths to set up the link connecting NI-0 and router-A in Fig. 1(c). To enhance the support for large-scale networks containing plenty of links, we optimize the router by adding one additional channel on each direction named bypass channel, i.e. *C/BS/BN/BE/BW* in Fig. 2 where *C* connects to NI and *BS* denotes the bypass south channel for example. With bypass channels, packets on the link from NI-0 to router-A can still go through channels between tile (1, 3) to tile (1, 1) even if the normal channels have been occupied.

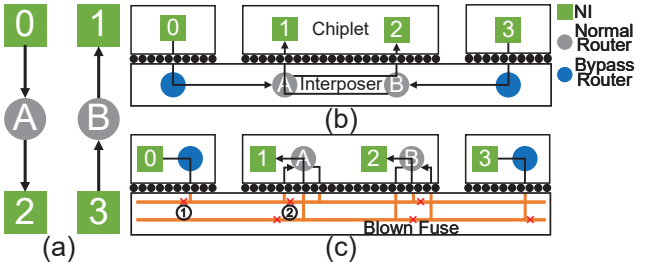

Figure 3: Example of GIA assembly-time configuration. (a) target inter-chiplet network topology. (b) and (c) configurations for active GIA and passive GIA respectively, and unused channels are not shown.

To alleviate impacts caused by bypass channels on power, performance, and area, we remove their pipeline relevant logics which typically dominate the critical path [12] as well as input buffers. The results in Subsection 5.2 and 5.3 show that this optimization substantially increases the scale of supported inter-chiplet network topology with little overhead.

Thus, there are three modes for a configurable router to operate in the tile as shown in Fig. 1(d). First, a configurable router is mapped as a *normal router* corresponding to a router in the inter-chiplet network topology, implying portions of normal channels are configured into the pipeline mode. Meanwhile, other channels can be used to pass packets like wires independently. Second, a router is used as a *bypass router*, i.e. a crossbar switch passing packets in a pre-deterministic way, and is not considered as part of the topology. For this case, unnecessary control units such as VC allocators are turned off to save power (controlled by ⑨). Third, the unused router can be powered down to reduce power further.

Another optimization is sharing *BS* channel with *C* channel connected to the NI, which is controlled by ⑦ and ⑧, to reduce overhead. This is supported by the experimental result in Subsection 5.3, which shows that a 30 * 30-tile interposer can perfectly support the network topology with up to 98 NIs, implying that 89.11 % of *C* channels are deactivated in practice and do not deserve dedicated physical channels.

To support the arbitrary network topology and flexible routing, table-based routing that stores routing tables in configurable NIs is adopted.

3.3 GIA Assembly-Time Configuration

An example of configuring GIA is shown in Fig. 3. For active GIA in Fig. 3(b), routers are configured to operate with the designated modes and channel assignment mentioned in Subsection 4.5. For instance, the configurable router connected to NI-1 in Fig. 3(b) is

mapped as router-A in Fig. 3(a), and the router connected to NI-0 Fig. 3(b) is set as the bypass router. For long-path transmission, the buffers or flip-flops in input channels of intermediate routers can be configured as pipeline stages for multi-cycle data transmission. The assembly configuration for passive GIA is more complex. In addition to configuring routers, we use the fuse technology [5] to create adaptive topologies. As depicted in Fig. 3(c), endpoints of topological links, e.g. ① and ② of the link from NI-0 to router-A, are blown to prevent signal interference when multiple packets are propagating on the shared wire. Compared with the active interposer, wires in the passive one need to resurface back to the chiplet in two special cases. One is for long-path transmission when buffers or flip-flops are needed [11]. The other is when encountering turns or switching channels on the path, e.g. one turn exists on the path from NI-0 to router-A in Fig. 1(d). Since wires of channels in the interposer are disconnected in the passive GIA, routing packets can neither change directions nor switch to another channel in the same direction in the interposer. To handle these two cases, we combine the designs of gas-station in [3] and bridge chiplet in [5]. For tiles covered by chiplets, wires can be bridged into the chiplet, get registered or channel-switched, and then return to the passive interposer. For a tile not covered by chiplets, if resurfacing is needed, we use a modular auxiliary chiplet containing a configurable router to realize the functionality as in [5]. The auxiliary chiplet has a standard one-tile size and can be produced in large quantities with low cost [5]. For passive GIA, to reduce the area overhead of moving routers into chiplets, we replace normal channels in tiles without NIs by another set of bypass channels because tiles with NIs can provide enough routers to be mapped as normal routers for inter-chiplet networks (See Subsection 4.3).

4 DESIGN AUTOMATION FRAMEWORK FOR GIA-BASED SYSTEM

In this section, we present the proposed GIA-based 2.5D system synthesis framework, aiming at attaining the optimal end-to-end integration solution for a certain design specification.

4.1 Overview of the Design Automation Framework

We consider five common metrics to optimize in this process: power, performance, area, temperature, and cost. The complicated optimization process is divided into four stages, as shown in Fig. 4. In the first stage of chiplet selection, we determine which chiplets to use and focus on optimizing power, performance, area, and cost. After the chiplets are chosen, an inter-chiplet network with low latency and low power is generated in the second stage. Then, in the stage of placement generation, we decide the position and orientation of each chiplet. In GIA mapping, the configuration of routers and wires in GIA is determined to minimize power and latency according to the generated network topology and chiplet placement. In the last step of optimization iteration, we evaluate power, performance, temperature, which are used in the cost function of placement generation. To formulate the problem of GIA-based system synthesis flow, we use definitions similar to those in [18].

Definition 1. The application used as target workload of the system can be described by the *application task graph*, which is a directed acylic graph $ATG(T, D)$ with vertex $t_i \in T$ representing a task and edge $d_{i,j} \in D$ representing data dependencies between t_i and t_j. Each task t_i is annotated with information including the potential chiplet set C_i^f, execution time $et_{i,m}$ and resources consumption $r_{i,m}$ on the specific chiplet c_m. The edge weight $wt_{i,j}$ indicate the data transfer rate from source task t_i to target task t_j.

Table 1: Notations used in design automation framework.

Notation	Meaning
C, C_i^f	Set of candidate chiplets; set of feasible chiplets for task t_i in candidate chiplets.
$s_{i,m}$	Task t_i is mapped to chiplet c_m.
$b_{i,j,m,n}$	Task t_i is mapped to chiplet c_m and task t_j is mapped to chiplet c_n.
BI_m, BO_m	Input/output bandwidth of chiplet c_m.
$wt_{i,j}$	Communication rate from task t_i to task t_j.
R_m	Total available resources of chiplet c_m.
$r_{i,m}$, $et_{i,m}$	Resources consumption/execution time of task t_i when running on chiplet c_m.
ft_i	Finishing time of task t_i.
l	Delay of inter-chiplet communication.
u_m	Chiplet c_m is contained in the ultimate system.
P, A, Co, FT	Total power, area, and costs of chiplets; latest finishing time of tasks.
P', L, Te, Te_{th}	Power of interposer; inter-chiplet network latency; peak system temperature; temperature threshold.
Te_{norm}, P'_{norm}, L_{norm}	Normalized coefficients.
k_P, k_{FT}, k_A, etc.	Coefficients of corresponding metrics in objective function/ energy function.

Definition 2. Communication between chiplets can also be represented as a *chiplet communication graph*, which is a directed graph $CCG(N, F)$. Vertex $n_i \in N$ represents an NI of the chiplet and edge $f_{i,j} \in F$ characterizes the directed data transfer from n_i to n_j. Every $f_{i,j}$ has a weight denoted as $wn_{i,j}$ representing the communication rate from n_i to n_j.

4.2 The Optimal Selection of Chiplets from Chiplet Library

The goal of chiplet selection is to find the appropriate chiplets from the available chiplet library regarding the design specification and the given application. When selecting chiplet components, multiple design goals must be considered.

We first include all possible chiplets used by tasks as the chiplet candidate set. For example, we assume task t_0 can run on CPU and GPU, and task t_1 can run on CPU and DSP. Then the constructed chiplet candidate set for t_0 and t_1 is {CPU 0, CPU 1, GPU, DSP}.

Similar to [36], the task mapping process is formulated as a binary integer linear programming (ILP) problem as follows(see notations in Table 1). Eqn. (1) is the weighted objective function where $k_P + k_{FT} + k_A + k_{Co}$ is 1. Eqn. (2) ensures each task can only be mapped to one of the feasible chiplets. Eqn. (3) makes sure $b_{i,j,m,n}$ is 1 if $s_{i,m}$ and $s_{j,n}$ are both 1. Eqn. (4) and (5) guarantee the inter-chiplet communication meets the bandwidth limitation. Eqn. (6) is the resource constraint of the chiplet, where resources such as memory consumed by all tasks running on it must not exceed the capacity. Eqn. (7) restricts task t_j to start only after the completion

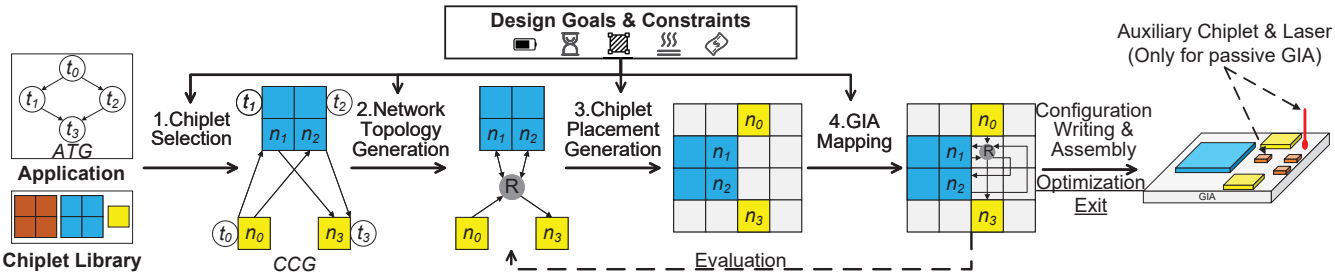

Figure 4: 2.5D integration flow with proposed design automation framework.

of t_i it depends on. When tasks are assigned onto different chiplets, inter-chiplet communication delay which is assumed as a constant l here for the lack of interconnecting details is also taken into account. Eqn. (8) ensures that completion time of the application is the latest finishing time of all tasks. Eqn. (9) describes that a chiplet is included into the system if any task is assigned to it. Eqn. (10) sums up the total power of used chiplets. Area and cost have the same form and are omitted here. Hard constraints such as maximum allowable power can also be added to the optimization process easily.

Minimize:

$$k_P \cdot P + k_{FT} \cdot FT + k_A \cdot A + k_{Co} \cdot Co \tag{1}$$

Subject to:

$$\sum_{c_m \in C_i^f} s_{i,m} = 1, \quad \sum_{c_m \in C, c_m \notin C_i^f} s_{i,m} = 0, \forall t_i \in T \tag{2}$$

$$2 \cdot b_{i,j,m,n} - 1 \leq s_{i,m} + s_{j,n} - 1 \leq b_{i,j,m,n}, \forall t_i, t_j \in T, c_m, c_n \in C \tag{3}$$

$$\sum_{d_{i,j} \in D} \sum_{c_n \in C, n \neq m} (b_{i,j,n,m} \cdot wt_{i,j}) \leq BI_m, \forall c_m \in C \tag{4}$$

$$\sum_{d_{i,j} \in D} \sum_{c_n \in C, n \neq m} (b_{i,j,m,n} \cdot wt_{i,j}) \leq BO_m, \forall c_m \in C \tag{5}$$

$$\sum_{t_i \in T} (s_{i,m} \cdot r_{i,m}) \leq R_m, \forall c_m \in C \tag{6}$$

$$ft_j \geq ft_i + l \cdot \sum_{c_m \in C} \sum_{c_n \in C, n \neq m} b_{i,j,m,n} + \sum_{c_m \in C} (s_{j,m} \cdot et_{j,m}), \forall d_{i,j} \in D \tag{7}$$

$$FT \geq ft_i, \forall t_i \in T \tag{8}$$

$$s_{i,m} \leq u_m \leq \sum_{t_j \in T} s_{j,m}, \forall t_i \in T, c_m \in C \tag{9}$$

$$P = \sum_{c_m \in C} u_m \cdot power_m \tag{10}$$

$$s_{i,m}, b_{i,j,m,n}, u_m \in \{0, 1\}, \forall t_i, t_j \in T, c_m, c_n \in C \tag{11}$$

With $s_{i,m}$ and u_m, we have the selected chiplets and task mapping, then we can obtain the chiplet communication graph with the user-defined NI access strategy for the next stage.

4.3 Network Topology Generation

We employ the graph partition-based method proposed in [20] to generate the inter-chiplet network topology. In each iteration, the chiplet communication graph is split by means of i min-cut partitioning. NIs with more traffic are allocated to the same partition and connected to the same router to reduce communication overhead. The value of i will be increased iteratively until all constraints such as bandwidth constraint are satisfied [20]. Since there are fewer routers than NIs in the generated topology, we can replace normal channels with bypass channels for routers in NI-free tiles in this step, thereby reducing chiplet area overhead for the passive GIA.

4.4 Chiplet Placement Generation

This stage determines the position and orientation of each chiplet on the interposer. Since the power, performance, area, and cost of chiplet side have been decided in the chiplet selection stage, we focus on optimizing power of interposer (P'), performance of inter-chiplet network measured by the average packet latency (L), and the peak temperature of the system (Te). We adopt the simulated annealing-based technique [3, 17] for placement optimization which includes three main parts: generation of initial placement, placement perturbation, and acceptance of neighbor placement. Unlike former works [3, 17], which evaluate all metrics in the entire optimization process, we split the iterative optimization process into two stages: thermal optimization and holistic optimization. The placer optimizes the heat distribution at the beginning and transits to holistic optimization when the temperature drops to the preset thermal threshold.

First, we use the compact placement generated by [14] as the initial placement. In each iteration, to get the neighbor placement, the current placement will be perturbed by four kinds of operations: moving a chiplet, making a chiplet jump to an empty location, swapping two chiplets, and rotating a chiplet [3, 17]. The maximum distance of chiplet movement in the thermal optimization stage is greater than that in the holistic optimization stage (normally one tile), in order to improve heat dissipation effectively. In addition, the rotation operation might only be executed in holistic optimization to achieve fine-tuned improvement.

To evaluate the neighbor placement, we use energy functions defined in Eqn. (12a) and (12b) for stages of thermal optimization and holistic optimization respectively. As shown in Eqn. (12a), only the thermal analysis is performed in thermal optimization. In holistic optimization, the interposer power and inter-chiplet network latency are obtained after GIA mapping. In Eqn. (12b), sum of $k_{P'}$ and k_L is 1 and the last item will be effective to impose a penalty when the system temperature exceeds the threshold. To have normalized coefficients such as Te_{norm}, we perturb the initial placement multiple times and have the average evaluation results before optimization starts.

$$E = \begin{cases} \dfrac{Te}{Te_{norm}} & \text{(12a)} \\ k_{P'} \cdot \dfrac{P'}{P'_{norm}} + k_L \cdot \dfrac{L}{L_{norm}} + k_{Te} \cdot max(Te - Te_{th}, 0) & \text{(12b)} \end{cases}$$

Algorithm 1: Interposer resource graph construction.

Input: Height/Width of interposer H/W, chiplet communication graph CCG
Output: Interposer resource graph IRG

1 $IDG, IRG \leftarrow$ `EmptyGraph()`, `EmptyGraph()`;
2 **for** $x \leftarrow 0$ **to** $W-2$ **do**
3 **for** $y \leftarrow 0$ **to** $H-1$ **do**
4 `AddEdge`(IDG, (x, y), $(x+1, y)$);
5 `AddEdge`(IDG, $(x+1, y)$, (x, y));
6 **end**
7 **end**
  `/* The addition of vertical edges are same as above and omitted. */`
8 **for** ni **in** $CCG.nodes$ **do**
9 $x_{ni}, y_{ni} \leftarrow$ `GetCoord`(ni);
10 $e_{S_i} \leftarrow ((x_{ni}, y_{ni}-1), (x_{ni}, y_{ni}))$;
11 $e_{S_o} \leftarrow ((x_{ni}, y_{ni}), (x_{ni}, y_{ni}-1))$;
12 **if** $ni.out\_degree > 0$ **then** `DelBC`(IDG, e_{S_i}) ;
13 **if** $ni.in\_degree > 0$ **then** `DelBC`(IDG, e_{S_o}) ;
14 **end**
15 **for** e_f **in** $IDG.edges$ **do**
16 **for** e_t **in** `GetSucEdge`(IDG, e_f) **do**
  `/* "NC" is normal channel and "BC" is bypass channel. */`
17 `AddEdge`(IRG, $(e_f$,"NC"$)$, $(e_t$,"NC"$)$);
18 **if** `HasBC`(IDG, e_f) & `HasBC`(IDG, e_t) **then** `AddEdge`(IRG, $(e_f$,"BC"$)$, $(e_t$,"BC"$)$) ;
19 **if** `HasBC`(IDG, e_f) **then** `AddEdge`(IRG, $(e_f$,"BC"$)$, $(e_t$,"NC"$)$) ;
20 **if** `HasBC`(IDG, e_t) **then** `AddEdge`(IRG, $(e_f$,"NC"$)$, $(e_t$,"BC"$)$) ;
21 **end**
22 **end**
23 `SetEdgeWeight`(IRG, 1);
24 **return** IRG;

At the annealing temperature K, we accept the neighbor solution with probability derived from

$$AP = min(e^{\frac{E_{current}-E_{neighbor}}{K}}, 1) \quad (13)$$

To make sure a good enough solution can be found in probabilistic search, we execute algorithm multiple times in parallel with different random seeds.

4.5 Network Topology Mapping on GIA

In this stage, the inter-chiplet network topology is mapped onto the configurable interposer, so that chiplets are finally connected into the custom network via the interposer.

The first step is to associate nodes in the inter-chiplet network to tiles of the interposer. The tiles of interposers NIs located at, e.g. NI-3 plugs into tile (1, 3) of the interposer in fig. 1 (d), can be obtained by the positions and orientations of corresponding chiplets. We then determine locations of routers by mapping some of configurable routers as normal routers. For active GIA, candidate routers are in the interposer. For passive GIA, candidate routers with normal channels are in the chiplet tiles containing NIs. In some cases, we can place additional auxiliary chiplets containing configurable routers on the passive interposer region not covered by chiplets if more router resources are needed to realize the topology. In this process, for each router in the network topology, we start searching for valid tiles containing unmapped candidate routers with enough normal input/output channels from the median of x/y coordinates of all its connected NIs. For example, the starting tile on the interposer of router-B in Fig. 1(c) is (1, 1) in Fig. 1(d). This rule aims at minimizing the sum of Manhattan distances between routers and their connected NIs to decrease the total routing length which helps reduce power and latency.

The next step is channel assignment. For each link in the network topology, we need to assign them a series of channels in configurable routers from the source tile to the destination tile. For example, the dotted path in Fig. 1(d) corresponds to the channel assignment from NI-0 to router-A in Fig. 1(c). Because links having more traffic cause greater impacts on power and performance, we sort links by their accumulated traffics and assign them in order. Note that every channel can only be assigned once. Since this process is similar to FPGA routing, we adopt the modified routability-driven algorithm described in [1]. This algorithm begins with building the resource graph including nodes representing available resources which are channels here and edge weights indicating the costs of using resources. Algorithm 1 illustrates the process of building the resource graph for the proposed interposer architecture. In the beginning, an interposer description graph IDG that connects tiles with bidirectionally horizontal and vertical edges is constructed (lines 1-7), and each edge is composed of a normal channel and a bypass channel. As stated in Subsection 3.2, C channel in the router might multiplex with BS channel. So the bypass channel of south input/output port is banned by `DelBC` if used by the connected NI (lines 8-14). `GetCoord` is used find the located interposer tiles of NIs. Then we transform channel resources in IDG to nodes in the interposer resource graph IRG (lines 15-22). When an edge connects one channel with one of its successor channels found by `GetSucEdge`, the packet is allowed to go from the former to the latter, e.g. packets in NW_{in} channel of router-A can go to NE_{out} in Fig. 1(d). Both normal channels and bypass channels are considered (lines 17-20). `HasBC` method is used to distinguish the bypass channels banned by `DelBC` that cannot be used by packets. An initial cost is given to each edge in IRG lastly (line 23). Once IRG is obtained, an iterative pathfinding algorithm [1] will be performed on it. All links in the network topology find the least cost path between tiles that endpoints located. Note that a link in the inter-chiplet network topology can only start from normal channels at the source router and end in normal channels at the destination router as bypass channels have no pipeline logic. If a channel gets used multiple times, its cost will be increased accordingly, which leads the pathfinding algorithm to avoid selecting frequently used channels. Subsequent iterations rip up the assignment and perform reassignment until no overuse exists. To reduce turn and channel switching of passive GIA when resurfacing is needed, we also add penalties to the edge weights in these cases.

In the last step, we generate detailed configurations(modes of routers, values of configuration registers, fuse positions, etc.) according to router mappings and channel assignments.

5 EVALUATION

5.1 Experimental Setup

Inter-Chiplet network setup. The network frequency is 1 GHz as in [2]. Every router has 4 VCs per normal input channel and a 4-flit

Table 2: Chiplet library examples.

Chiplet	Size (mm)	Power (W)	Bandwidth (GB/s)	Resource
CPU [23]	2.4 * 3.15	0.35	9.6	14 cores
DSP [25]	2.5 * 2.5	0.5	61.6	196 cores
GPU [32]	8.6 * 8.6	30	48.06	256 cores
SRAM [23]	1.1 * 3.15	0.35	6	640 KB
DRAM [17]	8.75 * 8.75	20	25.6	512 MB

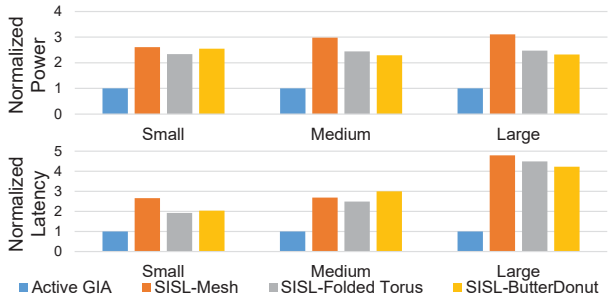

Figure 5: Normalized power and latency comparisons of active GIA and SISL.

buffer per VC. Flit size is 128-bit [11] and packet size is 8-flit. Prohibitive turns are introduced to guarantee network-level deadlock-free [26]. We use modified BookSim 2.0 [13] to simulate communication latency of networks and use DSENT [30] to evaluate the power and area overheads.

GIA setup. We consider 1-*mm* tile in this paper. The microbump pitch is 20 μm which has been realized in [33]. To obtain the maximum distance that flits can traverse in one cycle, we perform simulation using LTSPICE with parameters in [19, 28]. HotSpot [37] is used to estimate system temperature. We assume 45 *nm* technology node is used for both interposer side and chiplet side to make passive interposers comparable with active interposers. Three sizes of square interposers: 20 *mm* (*small-scale*), 30 *mm* (*medium-scale*), 40 *mm* (*large-scale*), are used in experiments to examine systems with different scales.

Framework setup. We assume 12 chiplets are available in the library, including CPU, DSP, GPU, SRAM, and DRAM. Table 2 lists the parameters of some used chiplets. 50 synthetic workloads are generated with TGFF [4] for each size of interposers. In the stage of chiplet selection, we limit the system power density to 0.6 W/mm^2 which meets the majority of power demands [31], and decrease the upper bound of area which is treated as a hard constraint until a valid initial chiplet placements on the target interposer is found. k_P, k_{FT}, and k_{Co} is set to 0.33 and k_A is 0. The presented ILP is solved by IBM CPLEX. In placement generation, we use the temperature threshold of 85°C and set $k_{P'} = k_L = 0.5$. The total number of iterations in chiplet placement generation is 2000.

Baselines setup. We use SISL [27] and SiPterposer [5] as the baselines for active GIA and passive GIA respectively. In baselines, most parameters are same as GIA. In SISL and SiPterposer, five-port routers with four-stage pipepline are placed in 1-*mm* tiles and unused ones are powered down. For SISL, three common inter-chiplet network topologies [15] including Mesh, Folded Torus, and ButterDonut are considered. The proposed desgin automation framework are applied to both SISL and SiPterposer for fair comparisons.

5.2 Experimental Results

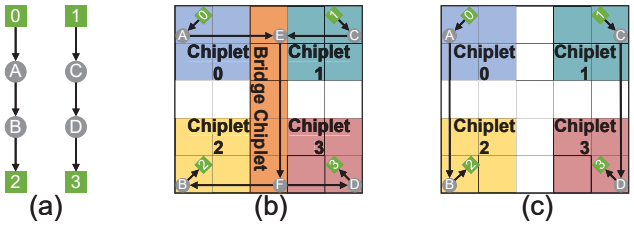

Figure 6: (a) target inter-chiplet topology. (b) configuration of SiPterposer. (c) configuration of passive GIA.

Active GIA vs. SISL. The comparison of active GIA and SISL on three sizes of interposers are shown in Fig. 5, where results are normalized to that of active GIA. As we can see, SISL suffers from large power consumption, which is at least 2.29x and average 2.57x higher than the active GIA counterpart. The main reason is SISL has larger network diameters compared with generated application-specific network topologies in GIA, and its intermediate routers on paths bring a large amount of energy consumption. Furthermore, SISL incurs at least 1.93x packet latency in small-scale systems, and as the interposer size increases the gap grows to at least 4.23x in large-scale systems. The packet latency of SISL is average 3.15x higher than active GIA. Among all topologies of SISL, Mesh has the highest average packet latency. Folded Torus and ButterDonut enable faster packet transmission by adding horizontal skip connections and diagonal links, however, they are for general multi-core systems and not effective for application-specific heterogeneous systems. The chiplet placer also improves performance of SISL by adjusting mapping of NIs to the inter-chiplet network in our framework. Nevertheless, the adjustment of mapping is limited by the fixed relative positions of NIs on the same chiplet and cannot compensate for the drawback of the fixed topology. The performance of GIA benefits from the application-specific inter-chiplet network architecture by clustering NIs with intensive mutual communications into the same router to avoid congestions.

Passive GIA vs. SiPterposer. The resulted SiPterposer solutions show an average 60.92x higher network latency and 2.99x higher power than that of passive GIA. Particularly, 64.67% of applications show packet transmission latencies over 1000 cycles, implying most of inter-chiplet networks are saturated in Sipterposer solutions. The lack of vertical wires in interposer is the main cause of inter-chiplet network congestion in SiPterposer. An example of how passive GIA with vertical wires in the interposer outperforms SiPterposer is shown in Fig. 6, where (a) is the network topology to configure and the selected four chiplets in (b) and (c) have the same placement. In SiPterposer, bridge chiplets are used to add vertical connections. For example, packets from NI-0 to NI-2 need to go through the bridge chiplet in Fig. 6(b). This brings considerable channel assignment conflicts in bridge chiplets. Unlike GIA, SiPterposer allows one single channel to be shared by multiple links of network topology via chiplet-side logic. As depicted in Fig. 6(b), router-E and router-F in the bridge chiplet can multiplex and demultiplex traffic flows, i.e. A→B and C→D. It then results in huge load at the channel from router-E to router-F, and congestion inevitably occurs when the load on a shared channel exceeds the bandwidth. In contrast, in Fig. 6(c), the vertical wires in passive GIA connect routers directly without sharing channels in the bridge chiplet, and bring significant performance improvement over Sipterposer.

Table 3: Comparisons of designs on realistic workloads.

APP	Interposer Design	Power in W	(x)	Latency in $cycle$	(x)
MMS	Active GIA	**1.14**	**(1)**	**19.49**	**(1)**
	SISL-Mesh	3.90	(3.42)	53.78	(2.76)
	SISL-Folded Torus	3.00	(2.63)	41.39	(2.12)
	SISL-ButterDonut	2.77	(2.43)	60.55	(3.11)
	Passive GIA	**1.02**	**(1)**	**20.27**	**(1)**
	SiPterposer	3.60	(3.53)	126.09	(6.22)
CMD	Active GIA	**1.30**	**(1)**	**27.18**	**(1)**
	SISL-Mesh	4.52	(3.48)	53.63	(1.97)
	SISL-Folded Torus	4.10	(3.15)	48.44	(1.78)
	SISL-ButterDonut	3.54	(2.72)	48.74	(1.79)
	Passive GIA	**1.28**	**(1)**	**27.81**	**(1)**
	SiPterposer	4.94	(3.86)	766.96	(27.58)

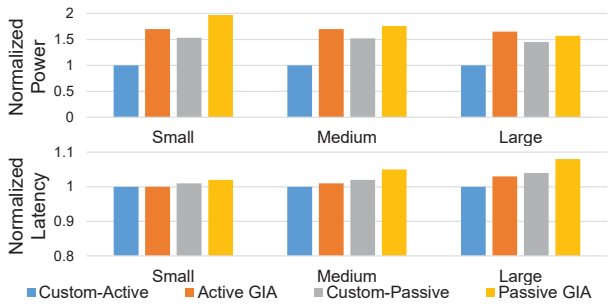

Figure 7: Normalized power and latency comparisons of custom interposers and GIA.

Experiments on real-world applications. We use realistic benchmarks of MultiMedia System (MMS) [10] and Cholesky Matrix Decomposition (CMD) [36] to examine the proposed GIA. The synthesized 2.5D system of MMS contains chiplets of CPU, DSP, and DRAM, which are listed in Table 2, and CMD uses CPU and DRAM. The results are shown in Table 3. Compared with the best power and performance of SISL, active GIA achieves average improvements of 2.58x and 1.95x respectively. Besides 3.7x power improvement, passive GIA takes 105.82 and 739.15 fewer cycles for network packet transmission. All interposer designs satisfy the given thermal constraint.

Overhead of GIA. Though our modified router owns a larger crossbar and additional input registers, power and silicon area overheads are not serious by multiplexing C channel and BS channel. The analysis with DSENT shows that under 0.5 flit/cycle injection rate, the router in GIA consumes 4.87% more power and 16.48% more silicon area than those of traditional five-port router in 45 nm technology node. For the defect-prone active GIA, with the largest interposer and defect density of 0.2 cm^{-2}[29], yield is 3.66% lower than interposers built by five-port routers. Therefore, the overhead associated with routers in GIA is acceptable. To measure how the reusable GIA interposer approaches the fully-customized solution in terms of performance and power, we compare GIA with custom interposers containing fixed optimized topologies which cannot be reused in other applications. The results are shown in Fig. 7. Compared to custom active interposer, active GIA introduces 68.33% and 1.33% additional power and performance overhead on average. To support assembly-time configuration, active GIA needs configurable routers that have higher energy and larger wire delay, which is absent from the custom active interposer. Passive GIA takes 17.61% and 2.59% additional power and packet latency than the custom passive interposer, besides the overhead of configurable router, it also needs to go through more microbumps and Electrostatic Discharge (ESD) protection circuits due to more frequent wire resurfacing as illustrated in Subsection 3.3. Note that in our experiments, that maximum power of active GIA and passive GIA is up to 8.83 W and will not seriously infringe the power budget. Though custom interposers achieve slight power and performance improvement, high non-amortizing NRE costs due to the sacrifice of reusability makes them impractical for low-volume products.

5.3 Discussion of GIA

Active GIA vs. passive GIA. As shown in Fig.7, compared with passive GIA, active GIA decreases the latency by 3.47% owning to packets passing through 1.71x more tiles in one cycle with the help of repeaters in the interposer. Note that the latency advantage of active GIA are not proportional to tile per cycle increment. On the one hand, the adopted topology generation algorithm tends to generate low diameter network with three hops on average, so the packet delay is dominated by contention delay in routers. On the other hand, chiplets having more communications can be moved closer in chiplet placement stage. Analysis shows that passive GIA transmits 40.66% of link distances in a single cycle and 88.02% of link distances within three cycles. We also find that active GIA consumes less power than passive GIA in small-scale systems but more in large-scale systems. Contrary to passive GIA, packets in active GIA need to traverse all crossbars along paths. As the interposer size grows, link distances get longer and active GIA spends more and more power on crossbars that exceeds the power consumption of microbumps and ESDs in passive GIA.

Effectiveness of bypass channels. To determine the maximum number of NIs supported by GIA, we synthesize multiple groups of 50 application-specific systems containing specific numbers of chiplets that possess one NI and one tile size. Each system is placed on one medium-scale GIA and another one with bypass channels removed, and we search for the placement solutions that is able to achieve successful GIA mapping within 50 try-runs at most. The maximum number of NIs that enable successful GIA mapping with and without bypass channels is 98 and 34, respectively. This implies that GIA with bypass channels offers bandwidth of 3.14 TB/s ideally, 2.88x more than the one without bypass channels, which can meet the bandwidth demands of various chiplet systems more easily.

6 CONCLUSION

In this paper, we propose GIA, a reusable interposer architecture to avoid the high NRE and silicon time-to-market. Based on GIA, we present an end-to-end design automation flow to optimally assemble the application-specific chiplet system. The experimental results show that our proposed active GIA and passive GIA can achieve 3.15x and 60.92x performance boost with 2.57x and 2.99x power saving over baselines respectively.

ACKNOWLEDGMENTS

This paper is supported in part by the National Key Research and Development Program of China under grant 2018AAA0102705, and in part by the National Natural Science Foundation of China (NSFC) under grant No.(61876173, 62004198, 62222411, 62090024, 61874124), Zhejiang Lab under Grants 2021PC0AC01. The corresponding authors are Ying Wang and Huawei Li.

REFERENCES

[1] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. 2012. *Architecture and CAD for deep-submicron FPGAs*. Vol. 497.

[2] Chia-Hsin Owen Chen, Sunghyun Park, Tushar Krishna, Suvinay Subramanian, Anantha P. Chandrakasan, and Li-Shiuan Peh. 2013. SMART: A single-cycle reconfigurable NoC for SoC applications. In *Design, Automation Test in Europe Conference Exhibition*. 338–343.

[3] Ayse Coskun, Furkan Eris, Ajay Joshi, Andrew B. Kahng, Yenai Ma, Aditya Narayan, and Vaishnav Srinivas. 2020. 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* 39, 12 (2020), 5183–5196.

[4] Robert P Dick, David L Rhodes, and Wayne Wolf. 1998. TGFF: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*. IEEE, 97–101.

[5] Pete Ehrett, Todd Austin, and Valeria Bertacco. 2019. SiPterposer: A Fault-Tolerant Substrate for Flexible System-in-Package Design. In *Design, Automation Test in Europe Conference Exhibition*. 510–515.

[6] Furkan Eris, Ajay Joshi, Andrew B. Kahng, Yenai Ma, Saiful Mojumder, and Tiansheng Zhang. 2018. Leveraging thermally-aware chiplet organization in 2.5D systems to reclaim dark silicon. In *Design, Automation Test in Europe Conference Exhibition*. 1441–1446.

[7] Yinxiao Feng and Kaisheng Ma. 2022. Chiplet Actuary: A Quantitative Cost Model and Multi-Chiplet Architecture Exploration. *arXiv preprint arXiv:2203.12268* (2022).

[8] Ilya K Ganusov, Mahesh A Iyer, Ning Cheng, and Alon Meisler. 2020. Agilex™ generation of intel® fpgas. In *IEEE Hot Chips Symposium*. IEEE Computer Society, 1–26.

[9] J Hruska. 2018. As chip design costs skyrocket, 3nm process node is in jeopardy. *ExtremeTech https://www.extremetech.com/computing/272096-3nm-process-node (22 June 2018)* (2018).

[10] Jingcao Hu and Radu Marculescu. 2005. Energy-and performance-aware mapping for regular NoC architectures. *IEEE Transactions on computer-aided design of integrated circuits and systems* 24, 4 (2005), 551–562.

[11] Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li, and Gabriel H. Loh. 2014. NoC Architectures for Silicon Interposer Systems: Why Pay for more Wires when you Can Get them (from your interposer) for Free?. In *Annual IEEE/ACM International Symposium on Microarchitecture*. 458–470.

[12] Natalie Enright Jerger and Li-Shiuan Peh. 2009. On-chip networks. *Synthesis Lectures on Computer Architecture* 4, 1 (2009), 1–141.

[13] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, D. E. Shaw, John Kim, and William J. Dally. 2013. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*. 86–96.

[14] Jukka Jylänki. 2010. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. *http://clb. demon. fi/files/RectangleBinPack. pdf* (2010).

[15] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H. Loh. 2015. Enabling interposer-based disintegration of multi-core processors. In *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 546–558.

[16] Martha Mercaldi Kim, Mojtaba Mehrara, Mark Oskin, and Todd Austin. 2007. Architectural implications of brick and mortar silicon manufacturing. In *Proceedings of the 34th annual international symposium on Computer architecture*. 244–253.

[17] Yenai Ma, Leila Delshadtehrani, Cansu Demirkiran, José L. Abellan, and Aiav Joshi. 2021. TAP-2.5D: A Thermally-Aware Chiplet Placement Methodology for 2.5D Systems. In *Design, Automation Test in Europe Conference Exhibition*. 1246–1251.

[18] Radu Marculescu, Jingcao Hu, and Umit Y. Ogras. 2005. Key research problems in NoC design: a holistic perspective. In *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. 69–74.

[19] Peter Moon, Vinay Chikarmane, Kevin Fischer, Rohit Grover, Tarek A Ibrahim, Doug Ingerly, Kevin J Lee, Chris Litteken, Tony Mule, and Sarah Williams. 2008. Process and Electrical Results for the On-die Interconnect Stack for Intel's 45nm Process Generation. *Intel Technology Journal* 12, 2 (2008).

[20] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo. 2006. Designing Application-Specific Networks on Chips with Floorplan Information. In *IEEE/ACM International Conference on Computer Aided Design*. 355–362.

[21] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. 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*. 57–70.

[22] Jawad Nasrullah, Zhiquan Luo, and Greg Taylor. 2019. Designing Software Configurable Chips and SIPs using Chiplets and zGlue. In *International Symposium on Microelectronics*, Vol. 2019. International Microelectronics Assembly and Packaging Society, 27–32.

[23] Saptadeep Pal, Jingyang Liu, Irina Alam, Nicholas Cebry, Haris Suhail, Shi Bu, Subramanian S. Iyer, Sudhakar Pamarti, Rakesh Kumar, and Puneet Gupta. 2021. Designing a 2048-Chiplet, 14336-Core Waferscale Processor. In *ACM/IEEE Design Automation Conference*. 1183–1188.

[24] Heechun Park, Jinwoo Kim, Venkata Chaitanya Krishna Chekuri, Majid Ahadi Dolatsara, Mohammed Nabeel, Alabi Bojesomo, Satwik Patnaik, Ozgur Sinanoglu, Madhavan Swaminathan, Saibal Mukhopadhyay, Johann Knechtel, and Sung Kyu Lim. 2020. Design Flow for Active Interposer-Based 2.5-D ICs and Study of RISC-V Architecture With Secure NoC. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 10, 12 (2020), 2047–2060.

[25] Uneeb Rathore, Sumeet Singh Nagi, Subramanian Iyer, and Dejan Marković. 2022. A 16nm 785GMACs/J 784-Core Digital Signal Processor Array With a Multilayer Switch Box Interconnect, Assembled as a 2×2 Dielet with 10um-Pitch Inter-Dielet I/O for Runtime Multi-Program Reconfiguration. In *IEEE International Solid- State Circuits Conference*, Vol. 65. 52–54.

[26] David Starobinski, Mark Karpovsky, and Lev A Zakrevski. 2003. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking* 11, 3 (2003), 411–421.

[27] Dylan Stow, Itir Akgun, Russell Barnes, Peng Gu, and Yuan Xie. 2016. Cost Analysis and Cost-Driven IP Reuse Methodology for SoC Design Based on 2.5D/3D Integration. In *Proceedings of the 35th International Conference on Computer-Aided Design*. Article 56, 6 pages.

[28] Dylan Stow, Itir Akgun, and Yuan Xie. 2019. Investigation of Cost-Optimal Network-on-Chip for Passive and Active Interposer Systems. In *ACM/IEEE International Workshop on System Level Interconnect Prediction*. 1–8.

[29] Dylan Stow, Yuan Xie, Taniya Siddiqua, and Gabriel H. Loh. 2017. Cost-effective design of scalable high-performance systems using active and passive interposers. In *IEEE/ACM International Conference on Computer-Aided Design*. 728–735.

[30] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *IEEE/ACM Sixth International Symposium on Networks-on-Chip*. 201–210.

[31] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David Kaeli. 2019. Summarizing CPU and GPU design trends with product data. *arXiv preprint arXiv:1911.11313* (2019).

[32] TechPowerUp. 2022. NVIDIA GeForce GT 1010. https://www.techpowerup.com/gpu-specs/geforce-gt-1010.c3762#:~:text=The%20GeForce%20GT%201010%20was,run%20on%20GeForce%20GT%201010.

[33] P Vivet et al. 2020. A 220GOPS 96-core processor with 6 chiplets 3D-stacked on an active interposer offering 0.6 ns/mm latency, 3TBit/s/mm2 inter-chiplet interconnects and 156mW/mm2@ 82% Peak-Efficiency DC-DC Converters. In *Proc. IEEE Int. Conf. Solid-State Circuits*.

[34] Xiaoxia Wu, Guangyu Sun, Xiangyu Dong, Reetuparna Das, Yuan Xie, Chita Das, and Jian Li. 2010. Cost-Driven 3D Integration with Interconnect Layers. In *Proceedings of the 47th Design Automation Conference*. 150–155.

[35] Jing Xia, Chuanning Cheng, Xiping Zhou, Yuxing Hu, and Peter Chun. 2021. Kunpeng 920: The First 7-nm Chiplet-Based 64-Core ARM SoC for Cloud Services. *IEEE Micro* 41, 5 (2021), 67–75.

[36] Lilia Zaourar, Massinissa Ait Aba, David Briand, and Jean-Marc Philippe. 2017. Modeling of Applications and Hardware to Explore Task Mapping and Scheduling Strategies on a Heterogeneous Micro-Server System. In *IEEE International Parallel and Distributed Processing Symposium Workshops*. 65–76.

[37] Runjie Zhang, Mircea R Stan, and Kevin Skadron. 2015. Hotspot 6.0: Validation, acceleration and extension. *University of Virginia, Tech. Rep* (2015).