Chipletizer: Repartitioning SoCs for Cost-Effective Chiplet Integration

Fuping Li*‡§, Ying Wang†‡§¶, Yujie Wang†‡¶, Mengdi Wang†‡, Yinhe Han†‡¶, Huawei Li*‡‖, Xiaowei Li*‡§
*SKLP, Institute of Computing Technology, Chinese Academy of Sciences
†CICS, Institute of Computing Technology, Chinese Academy of Sciences
‡University of Chinese Academy of Sciences
§Zhongguancun National Laboratory, ¶Zhejiang Laboratory, ‖Peng Cheng Laboratory
{lifuping20s, wangyujie, wangying2009, wangmengdi, yinhes, lihuawei, lxw}@ict.ac.cn

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473888

***Abstract*—The stagnation of Moore's law stimulates the concept of breaking monolithic chips into smaller chiplets. However, tactic design partitioning remains an unaddressed issue despite its crucial role in chip product cost reduction. In this paper, we propose Chipletizer, a framework to guide the design partitioning for those who would benefit from chiplet reuse across a line of SoC products. The proposed generic framework supports the repartitioning of multiple SoCs into reusable chiplets economically and efficiently with user-specified parameters. Experimental results show that, compared with existing partitioning strategies, our proposed framework achieves notable cost improvement on realistic products with acceptable power and latency overheads.**

***Index Terms*—chiplet, design partitioning, MCM, InFO, 2.5D**

I. Introduction

In the post-Moore era, the conventional monolithic system-on-chip (SoC) design methodology becomes unsustainable for increasingly complicated computing systems. The growing die sizes pushed by insatiable computational demands result in poor yields at cutting-edge process nodes, which escalates the recurring engineering (RE) costs [1]. Moreover, the non-recurring engineering (NRE) costs including software, IP licensing, and masks, are rocketing for advanced technologies [2] and are prohibitive for low-volume products.

Chiplet technology has emerged as a promising alternative. As shown in Fig. 1, SoC designs composed of IPs are partitioned into chiplets with die-to-die (D2D) interfaces and then reintegrated on high-performance interconnect carriers. Due to smaller areas, chiplets reduce RE costs with improved yields and wafer utilization [3]. It also enables adopting expensive advanced processes and cost-effective mature processes in the same package to save fabrication costs further[1]. Moreover, the higher-level silicon reuse accelerates chip development and eliminates repaying one-time costs such as masks, thus allowing a more efficient amortization of NRE costs.

When chiplet technology reshapes the ecosystem of integrated systems, design partitioning plays a vital role in the efficiency and cost-effectiveness of the chiplet-based design flow. There might be three major players in the ecosystem [4] as shown in Fig. 1: chiplet providers decomposing designs and offering chiplet products only, chiplet integrators building systems with available chiplets on the market, and chiplet providers & integrators that design their own chiplets and reuse those in-house chiplets to amortize the development costs of their chip products. Since the ecosystem is still in its infancy and available third-party chiplets are limited, repartitioning of the SoC products is still necessary for decreasing the costs of product lines for those chip vendors in the near future. Meanwhile, the paybacks of chipletizing designs are greatly impacted by how designs are partitioned [3, 5, 6]. On the one hand, the chipletization of an SoC introduces additional overheads such as D2D interfaces, cross-die communication, and packaging [1], which might eclipse the benefits of adopting chiplets. On the other hand, a decent partitioning strategy is also indispensable for amortizing the NRE costs of one's product line via chiplet reuse. These factors differentiate chiplet partitioning from partitioning systems into smaller subsystems in 2D chip design flows, which primarily aims at mitigating design complexity [7].

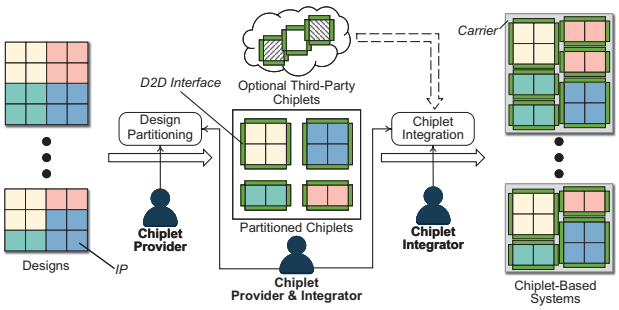

Fig. 1. Chiplet-based design flow and different roles in the chiplet ecosystem.

Hence, there is an increasing need for methodologies to disaggregate multiple SoCs for chip cost reductions with reasonable overheads in the chiplet context. At first, general abstractions for designs and chiplets are needed to model the partitioning process. Rather than relying on rigid partitioning methods like the IP-based policy [8], the partitioning method should adapt to diverse design parameters, evaluation models, and user-specified optimization goals. Furthermore, the design space of partitioning is immense even for a single SoC and explodes when trying to decompose multiple designs simultaneously to maximize chiplet reuse. To address these challenges, we develop an early-stage chiplet planning framework targeting multiple arbitrary SoC designs co-partitioning. In summary, this paper makes the following major contributions:

- We formulate the SoC repartitioning problem by using general representations of SoC designs and chiplets, which extract vital information for chipletization, including hardware components and communication demands.
- We propose *Chipletizer*, an adaptive framework for efficiently partitioning multiple monolithic SoCs into optimal chiplets. The framework can be practiced by designers to reduce the costs of SoC product lines, which hopefully alleviates the cost challenges faced by the current semiconductor industry.
- Compared with existing partitioning strategies, the proposed framework achieves at least 22.89% and 7.63% average cost reductions on low-volume and high-volume designs. Particularly, it also creates chiplets with unprecedented forms beyond existing experiences of partitioning.

II. Background and Motivation

A. Chiplet-Based Heterogeneous Integration

Tremendous efforts have been made to facilitate the chiplet integration process. The optimizations of inter-chiplet interconnect fabrics include network topology [9], hybrid network architecture

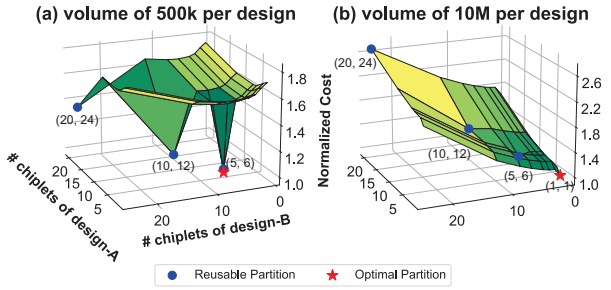

Fig. 2. Normalized average costs of chipletized design-A and design-B under different partitions obtained with the cost model in Subsection III-C3.

[10], and deadlock-free routing [11]. Standardized D2D interface protocols, such as BoW [4] and UCIe [12], are critical for building the chiplet ecosystem and have great impacts on die-to-die communication performance and overheads. Current multi-chiplet packaging technologies can be categorized into substrate-based multi-chip module (MCM), redistribution layer-based integrated fan-out (InFO) technology, and silicon interposer-based 2.5D integration [6]. They differ in package sizes, bandwidth demands, costs, etc. Overall, the chiplet integration process can be facilitated by system-level chiplet EDA [8, 13], cross-layer optimization technology [14], and existing single-chip EDA toolchain [15].

B. Chiplet Tax

Efficient system disintegration is crucial to minimize the power, performance, and area (PPA) overheads associated with chipletization, commonly referred to as the *chiplet tax*. First of all, D2D interfaces are needed to enable interactions between chiplets, which typically include transmitters (TXs), receivers (RXs), and electrostatic discharges (ESDs). They can result in considerable area overheads. For example, approximately 10% of silicon area is devoted to D2D interfaces for the AMD Zeppelin processor [1]. Besides, chiplet-based systems require more expensive packaging technologies and have lower packaging yields compared to monolithic systems [6]. Furthermore, off-die data transmissions entail an order of magnitude higher power and latency overheads than on-die ones [16]. To quantify the chiplet tax, several models have been proposed [3, 5, 6] to evaluate chiplet-based systems in terms of RE and NRE costs. They suggest that various factors, e.g. die sizes, manufacturing processes, and chiplet reusability, can impact the overall cost.

C. Design Partitioning Challenges

The partitioning stage is typically located at the beginning of the chiplet-based system design flow and has substantial impacts on performance and overheads [15]. Several naive strategies have been proposed. The uniform partitioning method [3, 6] divides simplistic multi-core systems into identical chiplets, which is not practical for realistic SoCs composed of diverse types and quantities of IPs. The balanced partitioning scheme [15] aims to increase die yields while minimizing inter-chiplet traffic but results in inferior intra-system chiplet reusability and higher NRE costs. The IP-based strategy [8] rigidly chipletizes every IP. It cannot scale to SoCs comprising plenty of IPs due to low packaging yields and insufficient chiplet reuse, which might cause higher costs than monolithic systems. Furthermore, it fails to cluster closely-interactive IPs into the same chiplet, leading to notable power and performance losses. Chopin [5] splits IPs of the same type into optimal chiplets following the cost model and encounters issues similar to IP-based partitioning. Besides the less efficiency of partitioning results, none of these prior proposals partition multiple systems simultaneously to explore inter-system silicon reuse, making them miss the opportunity to further amortize NRE costs. To motivate this, take the partitioning of hypothetical 20-core design-A and 24-core design-B as an example. The area of the 7-nm core is 5mm^2 and the silicon interposer-based packaging is assumed. The costs of all available partitions are shown in Fig. 2(a), where the amortized NRE costs significantly drop when design-A and design-B are divided into reusable chiplets owning 1/2/4 cores. However, reuse is not a silver bullet. When the product volume increases to 10M, NRE costs are well-amortized and RE costs become dominant. As depicted in Fig. 2(b), fewer chiplets in packages are preferred to reduce the costs of wasted known-good-dies (KGDs) caused by bonding failures, and the non-reusable monolithic partitions are optimal. At this time, the optimal partition under 500k volume, i.e. 4-core chiplet, achieves 1.06x higher cost. Our proposed Chipletizer overcomes the limitations of prior works and bridges the gap between SoC product lines and cost-effective partitioned chiplets.

III. Chipletizer Methodology

A. Problem Formulation

To support arbitrary SoCs, designs are assumed to be composed of different types and quantities of *blocks*, i.e. the user-defined smallest granularity of partitioning. The hardware and communication of design D is described by a unified *design characterization graph*.

Definition 1. The design characterization graph is a directed graph $DCG(B, E)$. Vertex $b_i \in B$ is a block tagged with its block type t_i. Edge $e_{i,j} \in E$ denotes the directed communication from b_i to b_j, and its weight $comm_{i,j}$ is the data transmission rate.

Definition 2. The result of partitioning a design is given by a collection of disjoint block subsets: $B' = \{B'_1, B'_2, ...\}$, where B'_i corresponds to a chiplet and $\cup B'_i = B$. Binary $\chi_{i,j}$ indicates cross-chiplet traffic, which is set to 1 when $b_i \in B'_p$, $b_j \in B'_q$, and $p \neq q$.

The process of design partitioning is to find the optimal B' for each given design characterization graph. We also use multisets to quantitatively describe blocks contained in designs and chiplets.

Definition 3. A design D or a chiplet C is described by $\{t_1^{m_1}, t_2^{m_2}, ...\}$ where m_i denotes the number of blocks with type t_i in $t_i^{m_i}$.

The blocks of a design can be extracted from its DCG. For each chiplet C_i, in addition to extracting blocks from its associated B'_i, D2D interfaces satisfying the bandwidth demand are inserted.

B. Framework Overview

In this paper, we consider three vital metrics of power (P), latency (L), and cost (Co). Other metrics such as thermal can also be included simply as in [14]. For m input designs, the optimization objective is:

$$\min \quad O = \sum_{i=1}^{m} w_i \cdot (\alpha_i \cdot \frac{P_i}{P'_i} + \beta_i \cdot \frac{L_i}{L'_i} + \gamma_i \cdot \frac{Co_i}{Co'_i}) \qquad (1)$$

Here w_i represents the weight of design D_i. Coefficients α_i, β_i, and γ_i are used to achieve user-specified trade-offs between power and latency overheads versus cost reduction. For example, β is set to 0 for those throughput-oriented systems which are insensitive to inter-chiplet latency. To obtain the normalized coefficients of metrics, P'_i, L'_i, and Co'_i, we perturb the initial partition multiple times and have the average evaluation results before the optimization starts [17].

Partitioning graphs without constraints is NP-hard, and the excessive design space poses challenges in exploring reusable chiplets. To address this, we adopt the two-level hierarchical partitioning method illustrated in Fig. 3. At the type level, blocks in design D_i are first

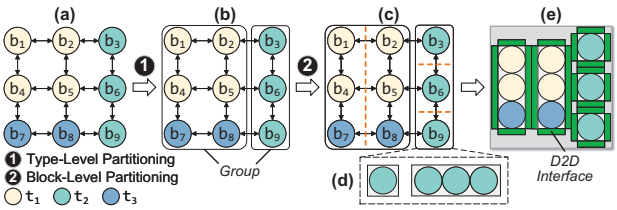

Fig. 3. Example of two-level hierarchical partitioning.

partitioned into a set of *groups* $S_i = \{G_1^i, G_2^i, ...\}$ with disjoint block types, e.g. two groups described by multisets $\{t_1^4, t_3^2\}$ and $\{t_2^3\}$ as shown in Fig. 3(b). Each group corresponds to a subgraph in the design characterization graph. Then each subgraph is partitioned at the block level and split into chiplets containing identical blocks using min-cut partitioning [15] to minimize inter-chiplet traffics, e.g. two chiplets with $\{t_1^2, t_3^1\}$ and three chiplets with $\{t_2^1\}$ as shown in Fig. 3(c). For the group $G = \{t_1^{m_1}, t_2^{m_2}, ...\}$, we use *candidate chiplet set* (CCS) in (2) to include all its valid partitioned chiplets.

$$CCS(G) = \{\{t_1^{m_1/k}, t_2^{m_2/k}, ...\} \\ |k \in \mathbb{Z}^* \wedge (\forall t_u^{m_u} \in G : m_u \% k = 0)\} \quad (2)$$

For example, $CCS(\{t_2^3\}) = \{\{t_2^1\}, \{t_2^3\}\}$ in Fig. 3(d). D2D interfaces are inserted at the end as shown in Fig. 3(e), which should conform to the maximum bandwidth demands among chiplets containing the same blocks to ensure the chiplet reusability. The complexity of enumerating every partition in the search space for m input designs is approximated as $O(n^m)$, where n is the number of partitions per design. This is still considered prohibitive when dealing with multiple large designs, thus we adopt the simulated annealing (SA)-based method (see Subsection III-D) to adjust partitions iteratively.

C. Chipletization Overhead Estimation

In the previous system design flows [8, 15], tedious stages including place-and-route of packages and chiplets, parasitic extraction, and timing analysis are performed to attain PPA. They incur significant time overheads for the optimization. To accelerate evaluation, we develop a simplified method for estimating power, latency, and cost. Firstly, we calculate the area A_p of each chiplet C_p consisting of the total block area A_p^{block} and D2D interface area A_p^{D2D} according to:

$$A_p = A_p^{block} + A_p^{D2D} = (\sum\nolimits_{t_i^{m_i} \in C_p} m_i \cdot A_{t_i}) + \frac{BW_p}{\sigma} \quad (3)$$

where A_{t_i} is the area of the block with block type t_i. A_p^{D2D} is estimated by its required bandwidth BW_p in (4) where $b_i \in B'_p$, $b_j \notin B'_p$, and symmetric TX and RX structures [12] are assumed. The area bandwidth density σ depends on factors such as the pitch of microbumps, percentage of microbumps for I/O, and data rate [16].

$$BW_p = \max(\sum\nolimits_{e_{i,j} \in E} comm_{i,j}, \sum\nolimits_{e_{j,i} \in E} comm_{j,i}) \cdot 2 \quad (4)$$

With the size of each chiplet derived from its area and an assumed aspect ratio of 1, we perform chiplet placement. Considering the significant impact of the inter-chiplet interconnect on power and latency, we then utilize the wire length-driven placement algorithm [13] to search for non-overlapping locations of the chiplets. The wire length of two chiplets is approximated by their Manhattan distance [14]. We impose constraints on minimum inter-die spacing [16] and maximum inter-chiplet wire length for passive packages [3] to generate valid placements. After obtaining the chiplet placement and wire lengths, we evaluate system metrics with the following models.

1) Power Model: The power overhead of inter-chiplet communication can be estimated based on cross-die traffic with:

$$P = \sum\nolimits_{e_{i,j} \in E} \chi_{i,j} \cdot comm_{i,j} \cdot (E_{I/O} + E_{i,j} + N_{i,j}^{FF} \cdot E_{FF}) \quad (5)$$

where $E_{I/O}$ represents the energy efficiency of I/O circuits including RXs, TXs, ESDs, and microbumps. $E_{i,j}$ is the energy efficiency of the wires that connect chiplet C_p and C_q, where $b_i \in B'_p$, $b_j \in B'_q$. It is modeled based on the technology nodes and lengths of wires [14]. For active interposers, energy consumed by $N_{i,j}^{FF}$ flip-flops for long-distance transmissions [3] are also taken into account. E_{FF} is the energy efficiency of the flip-flop scaling with the interposer technology node. $N_{i,j}^{FF}$ is computed by (6), where $WL_{i,j}$ is the wire length, and D_{max} is the maximum reachable distance per cycle modeled based on the node and frequency of the interposer.

$$N_{i,j}^{FF} = \lfloor WL_{i,j} / D_{max} \rfloor \quad (6)$$

2) Latency Model: The model in (7) measures the global inter-chiplet communication latency. $L_{i,j}$ is the data transmission delay from the chiplet of b_i to that of b_j. For active interposer-based systems shown in (8), $L_{i,j}$ comprises the latency of the I/O interface, and the delay of inserted flip-flops. Equation (9) assumes that the weight $w_{i,j}^L$ depends on the relative traffic rate [18]. Users can also use a large $\kappa_{i,j}$ to penalize the cross-die critical path manually, e.g. the communication between the core block and its L1 cache block, which avoids partitioning them into different chiplets. The protocol-level logic might bring extra delay [12] and is not included here.

$$L = \sum\nolimits_{e_{i,j} \in E} \chi_{i,j} \cdot L_{i,j} \cdot w_{i,j}^L \quad (7)$$

$$L_{i,j} = L_{I/O} + N_{i,j}^{FF} \quad (8)$$

$$w_{i,j}^L = \frac{comm_{i,j}}{\sum_{e_{u,v} \in E} comm_{u,v}} \cdot \kappa_{i,j} \quad (9)$$

3) Cost Model: The cost of a chiplet-based system consists of the RE cost and the amortized NRE cost. Here we exemplify with an MCM system containing n chiplets with RE cost given by (10) [3].

$$Co^{RE} = \frac{Co_{sub} + \sum_{i=1}^{n} (Co_{die_i} + Co_{bond_i})}{Y_{bond}^n} \quad (10)$$

Here Co_{sub} is the substrate cost relevant to package size, Co_{bond_i} is the bonding cost of the chiplet, Y_{bond} is the chiplet bonding yield, and Co_{die_i} is the cost of KGD obtained with (11) [3].

$$Co_{die} = \frac{\frac{Co_{wafer}}{N_{die}} + Co_{test}}{Y_{die}} \quad (11)$$

Here Co_{wafer} is the process-dependent wafer cost, N_{die} is the number of dies per wafer related to die size, Co_{test} is the die testing cost, and Y_{die} is the die yield influenced by process and die size.

In order to provide a comprehensive analysis of the RE cost, we further divide (10) into five parts following [6]. These are: (i) The cost of raw packages, i.e. Co_{sub}. (ii) The cost of wasted packages due to packaging failures. (iii) The cost of raw dies, whose main component is Co_{wafer}/N_{die}. (iv) The cost of die defects for the imperfect die yield Y_{die}. (v) The cost of wasted KGDs caused by packaging failures, which is $(\sum_{i=1}^{n} Co_{die_i}) \cdot (1/Y_{bond}^n - 1)$.

The amortized NRE cost of the system is:

$$Co^{NRE} = \frac{Co_{sub}^{NRE}}{Q_{sub}} + \sum\nolimits_{i=1}^{n} \frac{Co_{die_i}^{NRE}}{Q_{die_i}} \quad (12)$$

where Co_{sub}^{NRE} and $Co_{die_i}^{NRE}$ are NRE costs of the substrate and chiplet, and Q_{sub} and Q_{die_i} are their respective quantities. The NRE

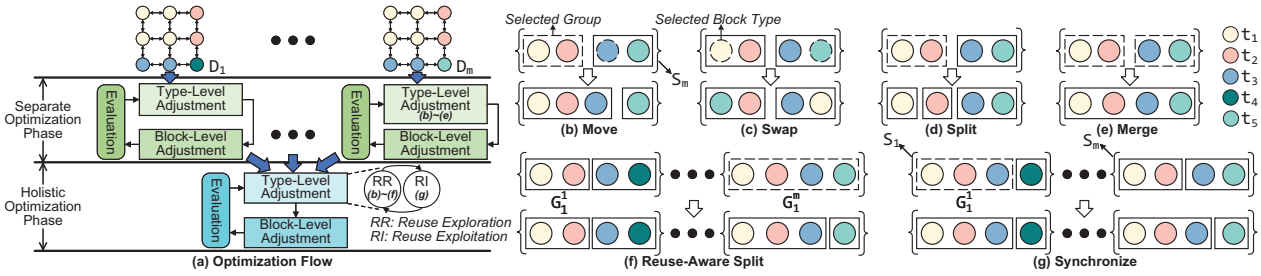

Fig. 4. (a) Optimization flow of Chipletizer. (b)-(g) Perturbation operations used in type-level adjustment which only show the block types of groups.

costs consist of chiplet area-dependent expenses such as verification, and area-independent costs like full masks [6]. Q_{die_i} indicates the reusability of chiplet C_i and takes all the chiplets owning identical blocks with C_i into account. Thus, chipletization costs are influenced not only by intra-system factors including die sizes, system scales, and processes, but also by inter-system factors of chiplet reusability.

D. Two-Phase Optimization Flow

We employ the SA-based method to effectively handle the large and complex search space of design partitioning. This model-agnostic approach also allows users to flexibly integrate customized evaluation models into the framework. To accelerate the optimization process of multiple designs, we divide it into two phases. In the separate optimization phase, each design is individually optimized, which provides high-quality initial solutions for the next phase. In the holistic optimization phase, we aim at reducing the amortized NRE costs further by seeking reusable chiplets among various designs.

1) Separate Optimization: The SA-based partitioning algorithm consists of initial partition generation, perturbation of the current partition, and acceptance of the neighbor partition. A random partition is used as the initial solution. For adopting the two-level partitioning method in Subsection III-B, the partition perturbation in each iteration consists of type-level and block-level adjustment, as shown in Fig 4(a). At the type level, we adjust blocks in groups according to their block types. The move operation (Fig. 4(b)) randomly picks a block type and moves it to another randomly selected group. The swap operation (Fig. 4(c)) entails swapping two randomly chosen block types across different groups. The split operation (Fig. 4(d)) randomly selects a group and then divides it into two equal-sized groups. With the merge operation (Fig. 4(e)), two randomly selected groups are combined. Then we can directly decide the optimal chiplets of adjusted groups at the block-level adjustment. There are at most two groups G_p^i and G_q^i in design D_i that are affected by an aforementioned operation. From their candidate chiplet sets, we select chiplets with the smallest objective function value for a single design defined in (1) by traversing the Cartesian product $CCS(G_p^i) \times CCS(G_q^i)$. To generate feasible chiplets, we exclude solutions that contain chiplets exceeding the reticle size limit [1]. Because the maximum bandwidth offered by a chiplet is limited by its perimeter [11], we also discard solutions violating the constraint:

$$BW_p \leq (\sqrt{A_p^{block}/\omega} + \sqrt{A_p^{block} \cdot \omega}) \cdot 2 \cdot \mu \tag{13}$$

where A_p^{block} and BW_p are computed by (3) and (4). ω is the aspect ratio of chiplets assumed as 1. μ is the shoreline bandwidth density modeled similarly to the area bandwidth density σ in (3). The neighbor partition is accepted as the current solution with probability:

$$AP = \min(e^{\frac{O_{current} - O_{neighbor}}{K \cdot O_{norm}}}, 1) \tag{14}$$

where $O_{current}$ and $O_{neighbor}$ are objective function values of the current and neighbor solution defined in (1) of a single design. K is the annealing temperature. To get the normalized coefficient O_{norm}, we perturb the initial partition multiple times and have the average evaluation results before the optimization starts [17].

2) Holistic Optimization: For m designs, we use *reuse score* $r(G_p^i)$ to quantify the reusability of the partitioned group G_p^i:

$$r(G_p^i) = \sum_{j=1 \wedge j \neq i}^{m} \psi(G_p^i, D_j) \tag{15}$$

Here $\psi(G_p^i, D_j)$ is 1 indicates at least one of the candidate chiplets of G_p^i can be reused in design D_j and is derived from:

$$\psi(G_p^i, D_j) = \begin{cases} 1, & \text{if } CCS(G_p^i) \cap CCS(G') \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

where $G' = \{t_u^{m_u} | t_u \in G_p^i \wedge t_u^{m_u} \in D_j\}$ is the group extracted from design D_j owning same block types as G_p^i, e.g. $\psi(G_1^1, D_m)$ is 1 in Fig. 4(f) with common candidate chiplets $\{t_1^4, t_2^2\}$ and $\{t_1^2, t_2^1\}$.

In this phase, we continue to use the SA-based method and derive the initial partition of each design from the outputs of separate optimization. As shown in Fig. 4(a), the type-level adjustment owns alternate states of *reuse exploration* and *reuse exploitation*, which perturb groups with different operations. In each iteration of the reuse exploration state, we randomly choose a design and perform the operations used in separate optimization on it. To accelerate the discovery of reusable partitions, we also employ the reuse-aware split operation to obtain the largest reusable group. For the randomly selected group G_p^i, we initialize a new empty group G_q^i. $t_u^{m_u} \in G_p^i$ is moved into G_q^i if no $r(G_q^i)$ reduction is caused by its movement. As shown in Fig. 4(f), group $G_1^m = \{t_1^4, t_2^2, t_3^2, t_5^1\}$ is split into $\{t_1^4, t_2^2, t_3^2\}$ which is reusable in D_1, and $\{t_5^1\}$. In the reuse exploitation state, for m designs, we randomly select a group G_p^i with the possibility SP in (17). The selection tends to choose groups showing stronger reusability. Each design D_j then synchronize with the selected G_p^i if $\psi(G_p^i, D_j) \neq 0$, which removes $t_u \in G_p^i$ from all $G_q^j \in S_j$ and reinsert them into S_j as a new group. As shown in Fig. 4(g), $S_m = \{\{t_1^4, t_2^2\}, \{t_3^2, t_5^1\}\}$ turns into $\{\{t_1^4, t_2^2, t_3^2\}, \{t_5^1\}\}$ after synchronizing with $G_1^1 = \{t_1^4, t_2^2, t_3^2\}$, producing chiplets of $CCS(\{t_1^4, t_2^2, t_3^2\})$ that can be used in both D_1 and D_m.

$$SP = r(G_p^i) / \sum_{j=1}^{m} \sum_{G_q^j \in S_j} r(G_q^j) \tag{17}$$

The block-level adjustment of holistic optimization is based on Algorithm 1, where the groups altered in type-level adjustment are contained in S^A and are adjusted one by one (lines 1-3). As chiplets might be reused across designs, groups in S^R that own the common candidate chiplets with G_p^i (line 2) need to be adjusted simultaneously. The timing complexity scales exponentially with

Algorithm 1 Block-Level Adjustment for Multiple Designs

Require: Set of altered groups: S^A.
1: **for** $G_p^i \in S^A$ **do**
2: $S^R \leftarrow \{G_q^j | \forall G_q^j \in S_j, CCS(G_p^i) \cap CCS(G_q^j) \neq \emptyset, 1 \leq j \leq m\}$;
3: BLA(S^R);
4: **function** BLA(S)
5: $O, M \leftarrow \emptyset$, EMPTYMAP();
6: **for** $G \in S, C \in CCS(G)$ **do**
7: **if** $C \notin M.keys$ **then** $M[C] \leftarrow \emptyset$;
8: $M[C] \leftarrow M[C] \cup \{G\}$;
9: **for** $C \in M.keys$ **do**
10: $o_1 \leftarrow$ EVALUATE($M[C], C$);
11: $o_2 \leftarrow$ BLA($S - M[C]$);
12: $O \leftarrow O \cup \{o_1 + o_2\}$;
13: **return** MIN(O);

$O(|CCS(G)|^{|S^R|})$ for traversing the Cartesian products of their candidate chiplet sets. Thus, we use the heuristic depth-first search BLA (lines 4-13). In lines 6-8, we set up the map M where $M[C]$ contains groups that can be composed of chiplet C. For each candidate chiplet C (line 9), we greedily apply it to all applicable groups to prune the search space and obtain part of the objective function value o_1 with EVALUATE based on (1) (line 10). Note that some other groups in designs of $M[C]$ might not be adjusted at this time. And we estimate them with per-design optimal block-level partitions obtained using the same way as separate optimization. Besides, we recursively search the solution o_2 for those designs that chiplet C is not applicable to (line 11). We record the sum of o_1 and o_2 in O (line 12), whose minimum corresponds to the optimal partition of S^R (line 13). After the adjustment, we accept the neighbor partition similar to the separate optimization phase.

E. Extensions of Chipletizer

1) Hybrid Process Technologies: To partition designs into chiplets with distinct process nodes, users need to designate the most mature node required by each block type, e.g. 7nm for core logic and 12nm for I/O [1]. The fabrication node of the chiplet is the most advanced one of the nodes required by the contained blocks. In addition to inter-chiplet communication overheads, the evaluation models also need to consider the varied PPA of blocks under diverse nodes [2].

2) Framework Used by Different Roles: The above framework is for the chiplet provider & integrator. For the chiplet provider solely supplying chiplets, the revenue comes from partitioned chiplets only. However, removing integration-relevant parameters such as Y_{bond}^n in (10) from the cost model will lead to undesirable undersized chiplets for integrators. We believe the cost models considering expenses of integration are also needed by chiplet providers to fabricate chiplets friendly to downstream integrators.

IV. EVALUATION

A. Baseline Methods

We compare Chipletizer (CI) with the following strategies, excluding uniform partitioning due to its limited support for realistic SoCs.
(1) Monolithic (M): implementing designs on monolithic dies.
(2) Reuse First (RF): partitioning multiple designs into the least types of chiplets to minimize total NRE costs. For design D_i, its *building chiplet set* (BCS) is given by:

$$\begin{aligned} BCS_i &= \{C_1^i, C_2^i, ...\} \\ &= \{\{t_a^{m_a/k'}, ...\} | \forall \{t_a^{m_a}, ...\} \in \mathcal{P}(D_i) \\ &\qquad \wedge k' = gcd(m_a, ...)\} \end{aligned} \tag{18}$$

TABLE I
DESCRIPTIONS OF BENCHMARKS

Company	# Designs	Markets	Block Types	Nodes (nm)
AMD [1]	6	server/desktop processor	CPU core, L3 $ PCIe controller, ...	7, 12
HiSilicon [19]	6	processor, base station chip	CPU cluster, accelerator core, PCIe controller, ...	7, 16
Intel [20]	10	server processor	CPU core, DDR controller, PCIe controller, ...	10
Rockchip [21]	10	IoT	CPU, GPU, ISP, ...	28
Nvidia [22]	6	server/desktop GPU	streaming multiprocessor, L2 $, DDR controller, ...	7

where $\mathcal{P}(D_i)$ denotes all non-empty groups of partitioning D_i at the type level. Since the chiplet NRE costs can be further reduced by smaller die area [6], we split each group $\{t_a^{m_a}, ...\}$ into k' chiplets at the block level, where k' is the greatest common divisor of $(m_a, ...)$. The building chiplet set of all input designs is $BCS = \cup BCS_i = \{C_1, C_2, ...\}$. Then we construct m input designs with the minimal subset of BCS, which is formulated as a binary integer programming problem. The objective function (19) optimizes the total area of distinct chiplets when their number minimizes, where binary y_p is 1 when chiplet C_p is used, M is a large constant, and A_p is the area of chiplet C_p. Equation (20) ensures the block types contained in partitioned groups are disjoint for each design (see the definition of group in Subsection III-B). Binary x_p^i is 1 when chiplet $C_p^i \in BCS_i$ is used in D_i. Equation (21) describes that chiplet $C_p \in BCS$ is used if any of the designs use identical chiplet C_q^j.

$$\min. \quad M \cdot \sum\nolimits_{C_p \in BCS} y_p + \sum\nolimits_{C_p \in BCS} y_p \cdot A_p \tag{19}$$

$$\sum\nolimits_{C_p^i \in BCS_i \wedge t_u \in C_p^i} x_p^i = 1, \forall t_u \in D_i, 1 \leq i \leq m \tag{20}$$

$$x_q^j \leq y_p \leq \sum x_q^j, \forall C_q^j \in BCS_j, C_p = C_q^j, 1 \leq j \leq m \tag{21}$$

(3) Balanced Partitioning (BP) [15]: minimizing the RE costs and inter-chiplet traffics. For each $DCG(B, E)$, the partitioner [15] generates k-partition $B' = \{B_1', ..., B_k'\}$ that minimizes $\sum_{e_{i,j} \in E} \chi_{i,j} \cdot comm_{i,j}$ and keeps area of B_i' balanced. We traverse k from 1 to $|B|$ and select the partition with the minimal RE cost.
(4) Finest Granularity (FG): equivalent to the IP-based partitioning [8] and chipletizing every block in the design. Design with block set $B = \{b_1, b_2, ...\}$ is partitioned into $\{\{b_1\}, \{b_2\}, ...\}$.
(5) Chopin (CO) [5]: partitioning each type of block individually. At the type level, design $D_i = \{t_1^{m_1}, t_2^{m_2}, ...\}$ is partitioned into the set of groups $S_i = \{\{t_1^{m_1}\}, \{t_2^{m_2}\}, ...\}$. Instead of optimizing multiple designs simultaneously to get accurate Q_{die} in (12), Chopin estimates it using $Q \cdot r$ to evaluate the amortized NRE costs for block-level partitioning. Here Q is the chiplet volume of a single design and r is the reuse factor inversely proportional to the area of the chiplet.

B. Experimental Setup

We use the design characterization graphs extracted from multiple realistic SoCs in Table I as benchmarks. For D2D interfaces, we adopt the power model in [14, 23] with 20μm microbump pitch, 50fF ESD capacitance, and 1GT/s data rate. The I/O bump map in [12] is also used to get the bandwidth densities, i.e. σ in (3) and μ in (13). $L_{I/O}$ in (8) is 2 cycles [12]. We use the 65nm active interposer with 1GHz frequency for multi-chiplet integration. Technology node parameters of interposers, such as wire and flip-flop energy efficiencies, are based on [14]. The cost model parameters, e.g. wafer costs and bonding yields, refer to [6]. For Chipletizer, we consider hybrid process

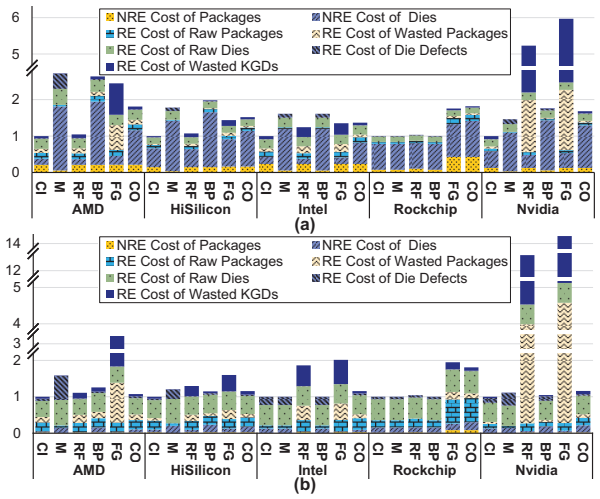

Fig. 5. Normalized cost comparison among different strategies with quantities of (a) 500k and (b) 10M. All costs are normalized to the results of Chipletizer. Details of each component refer to Subsection III-C3.

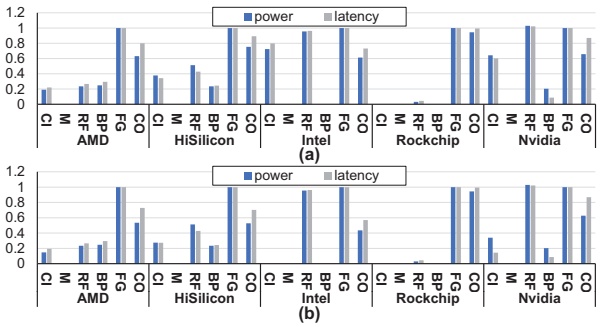

Fig. 6. Normalized power and latency comparison among different strategies with quantities of (a) 500k and (b) 10M. Due to the zero values of Chipletizer, we normalize the results to those of Finest Granularity instead.

technologies (see Subsection III-E1). The most mature nodes of blocks are assumed to be identical to their reported implementation, i.e. column of nodes in Table I. Quantities of 500k and 10M per design [6] are used to exhibit essential characteristics of each strategy for cases dominated by NRE costs and RE costs respectively. We set $\alpha_i/\beta_i/\gamma_i$ to 0.1/0.1/0.8 in (1) to prioritize the cost optimization [14]. K in (14) decays from 1 to 0.01 with a factor of 0.99.

C. Experimental Results

1) Cost Comparison: In the NRE costs dominant cases shown in Fig. 5(a), Chipletizer obtains a 22.89% total cost reduction compared to the best-performing Reuse First. One notable shortcoming of most baselines is the inadequate exploration of reusable chiplets. Since chiplet reusability is totally missed in Monolithic and Balanced Partitioning, they suffer from quite high NRE costs of dies. Finest Granularity scales the number of tape-out chiplets with the number of block types, which does not bring NRE cost reduction for Rockchip owning 20 distinct blocks and incurs 26.52% higher NRE costs of dies than Monolithic. Despite considering inter-system chiplet reuse, Chopin achieves inferior and unstable chiplet NRE cost reduction due to the inaccurate estimation of the number of reusable chiplets. Comparatively, Reuse First builds systems with the least and smallest tape-out chiplets and takes 62.32% less NRE costs of dies than Monolithic. For Reuse First and Finest Granularity, another common defect is the excessive number of generated chiplets which leads to low bonding yields according to (10). Especially for Nvidia owning plenty of streaming multiprocessor blocks, they generate 171 and 179 chiplets at most respectively, which brings a steep increase in RE costs of wasted packages and KGDs. Chipletizer not only produces highly reusable chiplets with 58.70% less NRE costs than Monolithic but also controls their quantities. It only uses 11 chiplets at most on Nvidia, bringing 29.79× higher bonding yields than Reuse First.

Fig. 5(b) depicts a diminishing cost advantage of Chipletizer when RE costs dominate, averaging 7.63% lower than the best-performing Balanced Partitioning. This is because RE costs consist of inevitable parts like wafer and package costs and have limited optimization potential. Note that this improvement is still significant when taking total cost savings into account. Monolithic encounters severe die yield issues on large SoCs, e.g. 41.84% of total costs spent on die defects for AMD with an average area of 596mm^2. Balanced Partitioning improves the die yields by smaller chiplets but has two weaknesses compared to Chipletizer. One weakness is the poor chiplet reusability. Even for massive volumes, this results in high NRE costs on products using leading-edge technologies such as HiSilicon. The other is the absence of heterogeneous technologies. On AMD, Balanced Partitioning combines blocks of PCIe and CPU core into the same chiplets, which avoids manufacturing PCIe blocks with more cost-effective 12nm technologies and introduces 1.16× higher raw die costs than Chipletizer. For Reuse First and Finest Granularity, the well-amortized NRE costs magnify the negative impacts of packaging yield issues on total costs, resulting in 2.04× and 1.96× higher proportions of costs for packaging failures compared to low-volume cases, respectively. On Intel and Rockchip, the cost reduction of Reuse First, Finest Granularity, and Chopin that split SoCs into smaller chiplets, such as increased die yields and wafer utilization, cannot offset chipletization expenses like interposers. In contrast, Chipletizer is able to generate the optimal monolithic systems.

2) Overhead Comparison: The power and latency overheads of inter-chiplet communication for each strategy are presented in Fig. 6. Balanced Partitioning outperforms or matches Chipletizer on all products except for AMD, where the unnecessary constraint of chiplet area balancing hinders the reduction of cross-die traffic. Reuse First, Finest Granularity, and Chopin do not optimize inter-chiplet communication and bring 2.85×/3.50×, 4.69×/6.28×, and 2.58×/4.93× higher power/latency overheads than Chipletizer, respectively. Increasing quantities from 500k to 10M, Chipletizer reduces average power and latency by 68.70% and 76.17% due to distinct partitions. Chipletizer tends to generate smaller and more reusable chiplets when NRE costs dominate, which results in higher inter-chiplet communication overheads. In contrast, when RE costs dominate, it produces larger and fewer chiplets with higher packaging yields. The maximum overhead of Chipletizer is 2.08-cycle latency with 19.14W power, which takes 4.57% of the total system power with a considerable cost reduction of 20.24% compared to the monolithic design. Thus, the overheads achieved by Chipletizer are acceptable, and users can increase α_i and β_i in (1) for those designs sensitive to power and latency overheads. All strategies own low runtime overheads and are able to obtain the outputs within 10 minutes.

3) Innovative Forms of Chiplets: For AMD at 10M quantities, compared with the commercial large I/O die and eight-core CPU die solution [1], Chipletizer adopts smaller I/O dies and 16-core CPU dies to boost die yields and packaging yields respectively, which achieves

a 4.11% cost reduction. For Nvidia at 500k quantities, canonical I/O dies [1] are split into fine-grained HBM controller dies and PCIe controller dies to be reused in products with diverse I/O demands.

4) Ablation Study: We study the impacts of two optimization phases by removing them respectively. Removing the separate optimization phase results in higher costs for designs with little inter-design chiplet reuse, e.g. a 63.04% cost increase for low-end Rockchip at 500k quantities whose optimal partitions are monolithic. This is because the separate optimization phase optimizes each design in parallel and thus is more efficient than the serial holistic optimization phase. At 500k quantities, removing the holistic optimization phase has no impact on Rockchip. However, other benchmarks take an average of 25.86% higher costs as they rely on exploring reusable chiplets to amortize the high NRE costs. The impact of removal diminishes at 10M quantities when NRE costs are well-amortized.

V. Conclusion

In this paper, we make an experimental attempt towards partitioning multiple SoCs into more cost-effective chiplets. The proposed Chipletizer can adapt to diverse design parameters, evaluation models, and goals. Validated by real-world products, our proposed framework is approved to be effective on various designs in concept.

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] S. Naffziger *et al.*, "Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product," in *ISCA*, 2021, pp. 57–70.
[2] M. Khazraee *et al.*, "Moonwalk: Nre optimization in asic clouds," *ACM SIGARCH CAN*, vol. 45, no. 1, pp. 511–526, 2017.
[3] D. Stow *et al.*, "Investigation of cost-optimal network-on-chip for passive and active interposer systems," in *SLIP*, 2019, pp. 1–8.
[4] B. Vinnakota *et al.*, "The open domain-specific architecture," *IEEE Micro*, vol. 41, no. 1, pp. 30–36, 2021.
[5] P. Ehrett *et al.*, "Chopin: Composing cost-effective custom chips with algorithmic chiplets," in *ICCD*, 2021, pp. 395–399.
[6] Y. Feng *et al.*, "Chiplet actuary: A quantitative cost model and multi-chiplet architecture exploration," in *DAC*, 2022, pp. 121–126.
[7] S.-J. Chen *et al.*, "Tutorial on vlsi partitioning," *VLSI design*, vol. 11, no. 3, pp. 175–218, 2000.
[8] J. Kim *et al.*, "Architecture, chip, and package co-design flow for 2.5 d ic design enabling heterogeneous ip reuse," in *DAC*, 2019, pp. 1–6.
[9] S. Bharadwaj *et al.*, "Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling," in *DAC*, 2020, pp. 1–6.
[10] N. E. Jerger *et al.*, "Noc architectures for silicon interposer systems," in *Micro*, 2014, pp. 458–470.
[11] J. Yin *et al.*, "Modular routing design for chiplet-based systems," in *ISCA*, 2018, pp. 726–738.
[12] D. D. Sharma *et al.*, "Universal chiplet interconnect express (ucie)®: An open industry standard for innovations with chiplets at package level," *TCPMT*, 2022.
[13] S. Osmolovskyi *et al.*, "Optimal die placement for interposer-based 3d ics," in *ASP-DAC*, 2018, pp. 513–520.
[14] A. Coskun *et al.*, "Cross-layer co-optimization of network design and chiplet placement in 2.5-d systems," *TCAD*, pp. 5183–5196, 2020.
[15] M. A. Kabir *et al.*, "Chiplet-package co-design for 2.5d systems using standard asic cad tools," in *ASP-DAC*, 2020, pp. 351–356.
[16] S. Pal *et al.*, "A case for packageless processors," in *HPCA*, 2018, pp. 466–479.
[17] T.-C. Chen *et al.*, "Modern floorplanning based on b/sup */-tree and fast simulated annealing," *TCAD*, vol. 25, no. 4, pp. 637–650, 2006.
[18] F. Li *et al.*, "Noception: a fast ppa prediction framework for network-on-chips using graph neural network," in *DATE*, 2022, pp. 1035–1040.
[19] J. Xia *et al.*, "Kunpeng 920: The first 7-nm chiplet-based 64-core arm soc for cloud services," *IEEE Micro*, vol. 41, no. 5, pp. 67–75, 2021.
[20] I. E. Papazian, "New 3rd gen intel® xeon® scalable processor (codename: Ice lake-sp)." in *Hot Chips Symposium*, 2020, pp. 1–22.
[21] "Rockchip products." [Online]. Available: https://www.rock-chips.com
[22] J. Choquette *et al.*, "3.2 the a100 datacenter gpu and ampere architecture," in *ISSCC*, vol. 64, 2021, pp. 48–50.
[23] P. Ehrett *et al.*, "Analysis of microbump overheads for 2.5 d disintegrated design," *UMich. Ann Arbor Tech. Rep.*, 2017.