RapidChiplet: A Toolchain for Rapid Design Space Exploration of Chiplet Architectures

Patrick Iff
patrick.iff@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Benigna Bruggmann
benigna.bruggmann@alumni.ethz.ch
ETH Zurich
Zurich, Switzerland

Blaise Morel
blaise.morel@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Maciej Besta
maciej.besta@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Luca Benini
lbenini@iis.ee.ethz.ch
ETH Zurich, Zurich, Switzerland
University of Bologna, Bologna, Italy

Torsten Hoefler
torsten.hoefler@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Abstract

Chiplet architectures are on the rise as they promise to overcome the scaling challenges of monolithic chips. A key component of such architectures is an efficient inter-chiplet interconnect (ICI). The ICI design space is huge as there are many degrees of freedom such as the number, size, and placement of chiplets, the topology and bandwidth of links, the packaging technology, and many more. While ICI simulators are important to get reliable performance estimates, they are not fast enough to explore hundreds of thousands of design points or to be used as a cost function for optimization algorithms or machine learning models. To address this issue, we present RapidChiplet, a fast and easy to use ICI latency and throughput prediction toolchain. Compared to cycle-level simulations, we trade 0.25%–30.15% of accuracy for 427×–137,682× speedup.

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

1 Introduction

Increasing the performance-per-cost of processors and accelerators has become more challenging than ever, due to a slow-down of Moore's law [1]. This slow-down is caused by the exponentially growing design and manufacturing costs when transitioning to a more advanced technology node [2] as well as by the diminishing return of this transition due to the scaling limits of IO-drivers, analog circuits, and, most recently, static random access memory (SRAM). A promising mitigation strategy for these challenges leverages 2.5D stacking, where multiple silicon dies, called chiplets, are integrated into the same package. The fact that a single chiplet design can be reused for multiple products helps to amortize the non-recurring costs. Furthermore, since 2.5D stacking allows integrating heterogeneous chiplets that are built in different technology nodes into the same package, only components that can take full advantage of technology scaling are manufactured in advanced and costly technology nodes. Components that have reached their scaling limits are manufactured in mature, lower-cost technologies. These advantages have driven industry-leading companies to adopt 2.5D stacking for their products, e.g., Intel's Ponte Vecchio CPU [3], AMD's EPYC and Ryzen CPUs [4], NVIDIA's Blackwell GPU [5], or Tesla's DOJO AI chip [6].

A crucial component of a 2.5D stacked chip is an inter-chiplet interconnect (ICI) to provide connectivity between chiplets. The ICI's latency and throughput are affected by a wide variety of design choices, such as the packaging technology [7–10], number and size of chiplets [11], chiplet placement [12], die-to-die (D2D) link implementation [13, 14], communication protocol [15, 16], ICI topology [17–21], and many more. Exploring this huge design space with the tools currently available is challenging due to two major reasons: Firstly, existing tools for ICI latency and throughput estimation [22–25] usually rely on cycle-based timing simulations, which are time-consuming and hence only allow the exploration of a limited number of designs. Secondly, these tool often fail to capture the complex interplay between different ICI design decisions. For example, choosing an ICI topology with a higher router radix requires more physical layers (PHYs) per chiplet, which increases the chiplet area, which in turn increases the link lengths, which has a packaging-technology-dependent effect onto the ICI latency and throughput.

To address these challenges, we introduce RapidChiplet, a fast and comprehensive toolchain for rapid design space exploration (DSE) of ICIs. RapidChiplet features high-level latency and throughput proxies that trade-off accuracy for speed, e.g., for transpose traffic, we achieve a speedup of 1,686× (latency) and 102,110× (throughput) compared to cycle-based timing simulations, at the cost of a 0.28% (latency) and 17.89% (throughput) deviation from the simulation results. These proxies are intended for large-scale DSE where hundreds of thousands of designs need to be evaluated, or as a cost-function for optimization algorithms and machine learning models. To thoroughly evaluate a smaller number of selected designs, RapidChiplet provides a seamless integration of the cycle-based BookSim2 [22] network simulator. Furthermore, to capture the complex interplay between different ICI design decisions, RapidChiplet incorporates a wide range of configurable parameters, including, but not limited to, chiplet size, PHY-placement, ICI topology, packaging technology, chiplet placement, and routing algorithm. To automate the DSE, we provide a rich toolbox that sweeps over user-defined parameter ranges, automatically generates the required inputs, runs RapidChiplet, and visualizes the results.

2 The RapidChiplet Toolchain

Fig. 1 outlines the RapidChiplet toolchain. The heart of the toolchain is the RapidChiplet core, which reads a wide range of user-defined inputs, validates them, and estimates the ICI's latency and throughput. In addition, we offer cycle-based flit-level simulations using BookSim2 [22]. Our toolchain automates the DSE s.t. the user only

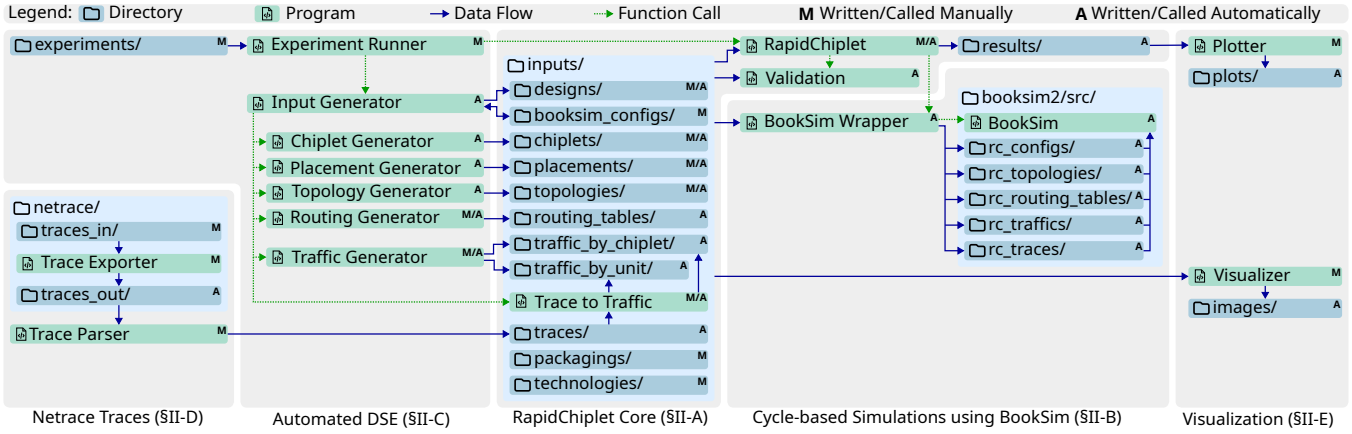

Figure 1: (§2) RapidChiplet architecture overview.

needs to specify parameter ranges (e.g., a list of topologies, chiplet-counts, packaging technologies, etc.) in an *experiments* file. The toolchain then iterates over all possible combinations of these parameters, generates the corresponding input files, and uses them to predict the ICI's latency and throughput. One of the few inputs that are not automatically generated are the traffic traces, which can be used for the cycle-based simulations in BookSim2. We provide functionalities to export network traces from the Netraces v1.0 trace collection [26] using Netrace [27, 28], and to parse them into the RapidChiplet format. The user can write custom parsing functions for arbitrary trace sources. Finally, we provide visualization functions for results and inputs.

2.1 RapidChiplet Core

2.1.1 Inputs and Validation. Fig. 2 visualizes the different inputs required by RapidChiplet. Each input directory can contain multiple input files, and to execute RapidChiplet, one file from each directory is required. To facilitate the handling of input files, a *design* file is used to specify the paths to the remaining input files. Input files can be combined at will as long as they are compatible (e.g., all chiplets used in the *placement* must exist in the *chiplet* file or chiplet-IDs in the *routing table*, *traffic*, or *trace* files must exist in the *placement* file). Our toolchain validates the inputs prior to each run.

2.1.2 Latency Proxy. We represent the ICI as an undirected, weighted graph $G = (V, E)$, where chiplets and on-interposer routers (for active silicon interposers only) are represented by vertices and the links between them are represented by edges. The weight $w(v)$ of a chiplet-vertex $v \in V$ is set to the chiplet's internal latency, and that of a router-vertex $v \in V$ is set to the router-latency. The weight $w(\{u, v\})$ of an edge $\{u, v\} \in E$ is set to the link latency, which can be constant or link-length dependent. In case of length-dependent link latencies, RapidChiplet computes all link-lengths, considering the chiplet positions and rotations, the placement of PHYs within the chiplets, and the link routing method (e.g., Manhattan, or direct). If the link is connected to a chiplet rather than an on-interposer routers, the PHY-latency is added to $w(\{u, v\})$. To estimate the average packet latency L under a given *traffic* $\mathcal{T}$, we iterate over all entries (s, d, a) in $\mathcal{T}$, where s and d are the source- and destination-chiplet-IDs and a is the amount of traffic between them. For each such entry, we calculate the path $route(s, d)$ from s to d using the *routing table* and we sum up all vertex- and edge-weights on the path. We compute the average latency over all entries $(s, d, a) \in \mathcal{T}$, weighted by the amount of traffic a between chiplets s and d:

$$L = \frac{\sum\limits_{(s,d,a)\in\mathcal{T}} a \cdot \left(w(s) + \sum\limits_{\{u,v\}\in \text{route}(s,d)} w(\{u,v\}) + w(v) \right)}{\sum\limits_{(s,d,a)\in\mathcal{T}} a}$$

2.1.3 Throughput Proxy. We use the same graph representation as for the latency proxy, where we compute two new properties for each edge $\{u, v\} \in E$: the bandwidth $B(\{u, v\})$ and the flow $F(\{u, v\})$. The bandwidth $B(\{u, v\})$ depends on the number of bumps that are available to connect the link to its adjacent chiplets $c \in \{u, v\}$ and on the number of non-data wires N_{ndw} that are

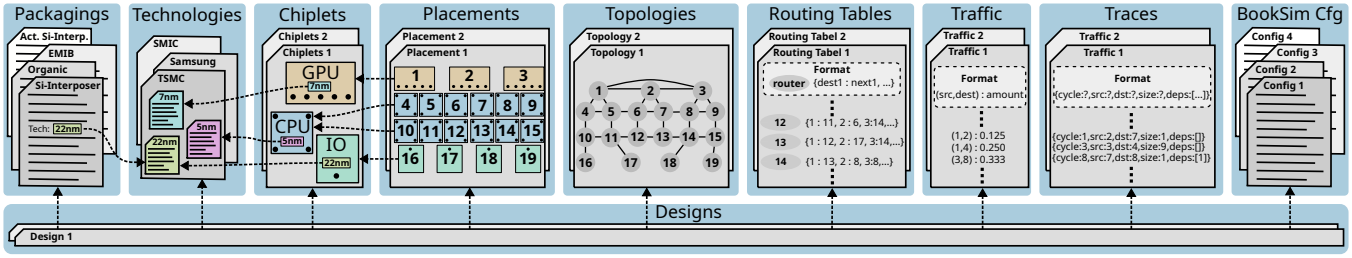

Figure 2: (§2.1.1) Overview of RapidChiplet input files and how they reference each other.

required for clock or handshaking signals. Said number of bumps depends on the chiplet area A_c, the bump pitch P_c, and the fraction $f_{c,\{u,v\}}$ of the chiplet-area available for the bumps of the link $\{u,v\}$. The flow $F(\{u,v\})$ is the sum of the traffic in $\mathcal{T}$ that is routed over the edge $\{u,v\}$:

$$B(\{u,v\}) = \left\lfloor \frac{A_c \cdot f_{c,\{u,v\}}}{P_c^2} \right\rfloor - N_{ndw} \quad \Bigg| \quad F(\{u,v\}) = \sum_{\substack{(s,d,a)\in\mathcal{T} \\ \{u,v\}\in \text{route}(s,d)}} a$$

Based on these two edge properties, we estimate the total throughput T under a given *traffic* $\mathcal{T}$. Notice that dividing $B(\{u,v\})$ by $F(\{u,v\})$ gives a metric for how much traffic the link $\{u,v\}$ can tolerate expressed as a fraction of the traffic that is currently routed over it, i.e., if this metric is smaller than 1, the link tolerates less traffic than it currently carries and if it is larger than 1, the link can tolerate more traffic. We estimate how much load the whole ICI can tolerate expressed as a fraction of the total traffic by taking the minimum of that metric over all edges $\{u,v\} \in E$: We multiply this metric by the total traffic to get the throughput T:

$$T = \min_{\{u,v\}\in E} \frac{B(\{u,v\})}{F(\{u,v\})} \cdot \sum_{(s,d,a)\in\mathcal{T}} a$$

This throughput proxy might look overly pessimistic, however, it turns out to be a good approximation for congestion since as soon as a link is saturated, the congestion starts to quickly spread across the network and also block flows that do not use the saturated link. In Section 3, we see that this proxy sometimes over- and sometimes underestimates the throughput.

2.1.4 Area, Power, and Cost Reports. RapidChiplet reports the total chiplet area, the interposer area, the power consumption, and an estimate for the manufacturing cost. The chiplet area report straightforwardly sums up the areas of all chiplets and the interposer area is that of the smallest possible rectangle that encloses all chiplets. These reports are intended for the automated DSE where the number, size, and placement of chiplets are automatically adjusted based on the design parameters. The same motivation applies to the power report, which, in addition to summing up the per-chiplet power and the interposer-power, can also incorporate the length-dependent power consumption of links. The manufacturing cost report uses a simple yield model to estimate the per-chiplet cost, which depends on the chiplet area and on technology-dependent inputs. These per-chiplet costs are summed up and added to the packaging cost to estimate the total system cost.

2.2 Cycle-based Simulations using BookSim2

To thoroughly evaluate a small number of promising designs that are identified using RapidChiplet's performance proxies, we support a seamless integration of the cycle-based, flit-level simulator BookSim2 [22]. The topology, including potentially link-length-dependent latencies, as well as routing tables, traffic patterns, and traces are exported to BookSim2. We extended BookSim2 with the capabilities to load custom routing tables and traffic patterns, and to simulate network traces, including dependency-tracking (out-of-the-box BookSim2 does not support trace-based simulations). To perform simulations using BookSim2, the user is required to supply a *booksim_config* file with simulation parameters.

2.3 Automated Design Space Exploration

We provide a rich toolbox to automatically explore the design space of ICIs. The user can specify ranges of topologies, chiplet counts and sizes, traffic patterns, traces, technology nodes, packaging technologies, latency profiles, and many more parameters. Our toolchain then iterates over all possible combinations of those parameters, generates the corresponding input files, and runs RapidChiplet. We proceed by describing our input generation scripts, which the user can easily extend with custom topologies, routing algorithms, placements, etc.

2.3.1 Chiplets. The toolchain creates chiplets with a configurable base area and power, plus a configurable area and power overhead per PHY. The number of PHYs is determined by the topology. Our chiplet generator supports different PHY placements (see Fig. 3) where the toolchain automatically selects the most suitable PHY placement for a give topology.

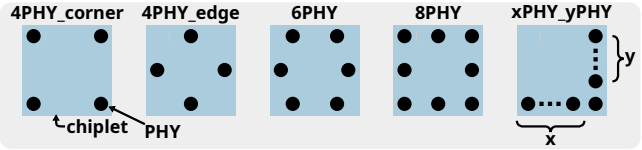

Figure 3: (§2.3.1) Available PHY placements

2.3.2 Placements. RapidChiplet offers two chiplet placements: A 2D grid that is used for most topologies and a hexagonal placement that is used for the HexaMesh [12] topology and its derivatives. The placement generator supports any chiplet size and a configurable spacing between chiplets.

2.3.3 ICI Topologies. RapidChiplet generates the following ICI topologies for configurable chiplet-counts: 2D Mesh, 2D Torus, Folded 2D Torus [29], Flattened Butterfly [30], HexaMesh [12], HexaTorus, Folded HexaTorus, OctaMesh, OctaTorus, Folded OctaTorus, Hypercube [31], Double Butterfly [17], ButterDonut [18], ClusCross [19], Kite [20], and SID-Mesh [21].

2.3.4 Routing Tables. Our routing table generator creates a routing table for each chiplet and on-interposer router. Notice that since RapidChiplet supports arbitrary topologies, routing algorithms for mesh network such as XY routing are not applicable. We support two different, deadlock-free routing algorithms for arbitrary topologies, which are both based on Dijsktra's algorithm [32]. The first one routes each packet along the shortest path and in the case of multiple shortest paths, it selects the next hop with the lowest ID to avoid deadlocks. This is the same deadlock-avoidance strategy as in BookSim2's routing algorithm for arbitrary topologies, however, this strategy fails to exploit path diversity. To alleviate this issue, we offer a second, randomized routing algorithm that uses the turn model [33], the simple cycle braking algorithm [34], and a dual graph construction [35] for deadlock-free, shortest-path routing.

2.3.5 Traffic. We offer generators for the synthetic *random-uniform*, *transpose*, *permutation*, and *hotspot* traffic patterns.

2.4 Exporting Traffic Traces using Netrace

We provide a script to export traces from the Netraces v1.0 trace collection [26] using Netrace [27, 28], and to parse them into the RapidChiplet format. These traces contain traffic between the L1

cache, L2 cache, and main memory, and they use source and destination addresses in the range 0 to 63.

2.5 Visualization

RapidChiplet can visualize its inputs (see Fig. 4 left) which is especially helpful when creating placements and topologies by hand, when extending the generator-scripts with new placements or topologies, or to analyze the solution that an automated DSE or optimization algorithm found. We also provide some functions to visualize results, e.g, latency-vs-load plots based on BookSim-simulations (see Fig. 4 right).

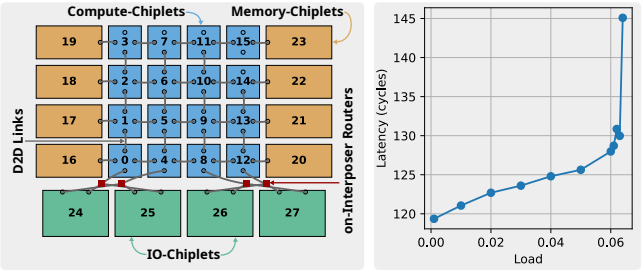

Figure 4: (§2.5) Visualized design (left); results-plot (right).

3 Evaluation

To evaluate the accuracy and speed of our latency and throughput proxies, we compare them against cycle-based flit-level simulations in BookSim2 [22].

3.1 Experiment Setup and Methodology

We evaluate RapidChiplet using the Mesh, Torus, Folded Torus [29], and SID-Mesh [21] topologies, the random-uniform, transpose, permutation, and hotspot[1] traffic patterns, and chiplet counts ranging from 9 up to 100. We set the latency of chiplets and PHYs to 3 and 12 cycles respectively, and we set the link-latency to 0.25 cycles/mm. Furthermore, we use square chiplets with a base-area of 74mm^2 and an area overhead of 0.85mm^2 per PHY. BookSim2 models input-queued routers with a pipeline of routing-, VC allocation-, switch allocation-, and crossbar traversal-stages. We use four virtual channels with 16-flit buffers each, and we adjust the number of simulated cycles to the number of chiplets. While a single BookSim-simulation is sufficient to get the zero-load latency, we need to run multiple simulations with increasing injections rates to determine the saturation throughput. We first increase the injection rate in 10%-steps and once saturation is reached, we go back to the last stable injection rate and proceed with 1%-steps and finally with 0.1%-steps. E.g., determining a saturation throughput of 12.3% requires 9 simulations with the injection rates 10%, 20%, 11%, 12%, 13%, 12.1%, 12.2%, 12.3%, 12.4%. The full list of parameters can be found in our repository[2]. All experiments are performed on a laptop PC with an Intel Core i7-1165G7 CPU running Arch Linux with kernel version 6.10.6.

[1]Four hotspot nodes and 50% of the traffic is directed towards these hotspots.
[2]https://github.com/spcl/rapidchiplet

3.2 Experimental Results

Fig. 5 shows the deviation of RapidChiplet's latency and throughput proxies from the BookSim2 simulation results (relative error) and RapidChiplet's speedup over BookSim2.

3.2.1 Latency Proxy. On average, our latency proxy is 1075× faster than simulations in BookSim2, which comes at the cost of an average error of 2.57%. The error mainly depends on the traffic pattern: For random-uniform and transpose traffic, the average error is lower than 0.3%, for hotspot traffic, it is 1.66% and for permutation traffic, it is 8.11%. The speedup depends on the number of chiplets and the traffic pattern. For the random-uniform and hotspot traffic patterns, the speedup grows linearly with the number of chiplets, while for the transpose and permutation traffic patterns, the speedup grows quadratically with the number of chiplets. This behavior can be explained by analyzing the runtime[3] of RapidChiplet and BookSim2. The runtime of BookSim2 grows quadratically with the number of chiplets, ranging from about 0.05 s for 9 chiplets to about 13 s for 100 chiplets (independently of the traffic pattern). The runtime of RapidChiplet grows linearly in the number of communicating chiplet-pairs, which grows quadratically in the number of chiplets for the random-uniform and hotspot traffic patterns and linearly for the transpose and permutation traffic patterns, hence, the difference in speedup between the traffic patterns. The simulated ICI topology does not seem to affect the error or speedup of RapidChiplet's latency proxy.

3.2.2 Throughput Proxy. Our throughput proxy achieves an average speedup of 69,079× over BookSim2 with an average error of 25.12%. The runtime of RapidChiplet's throughput proxy follows the same trends as its latency, proxy, i.e., it grows linearly with the number of chiplets for the transpose and permutation traffic patterns and quadratically for the random-uniform and hotspot traffic patterns. The combined runtime of the multiple BookSim2-simulations needed to identify the saturation throughput grows more or less linearly with the number of chiplets, ranging from about 10 s for 9 chiplets to about 300 s for 100 chiplets. Therefore, the speedup of our throughput proxy declines with an increasing chiplet count for the random-uniform and hotspot traffic patterns, while it stays more or less constant for transpose and permutation traffic.

Another interesting observation is that the average speedup of the throughput proxy is 64× higher than that of the latency proxy. The reasons for this are twofold: Firstly, as explained in Section 3.1, we need to run multiple simulations with increasing injection rates to determine the saturation throughput. Secondly, and most importantly, simulations close to saturation (as required to determine the saturation throughput) are significantly more time-consuming than simulations at low injection rates (as required to determine the zero-load latency).

The huge speedup of RapidChiplet's throughput proxy comes at the cost of a rather high approximation error. These results fortify our approach of using performance proxies for a large-scale DSE where simulations are not feasible, and using simulations to accurately evaluate a set of selected design points.

[3]Runtime results are omitted due to space constraints. They can be found in our repository: https://github.com/spcl/rapidchiplet

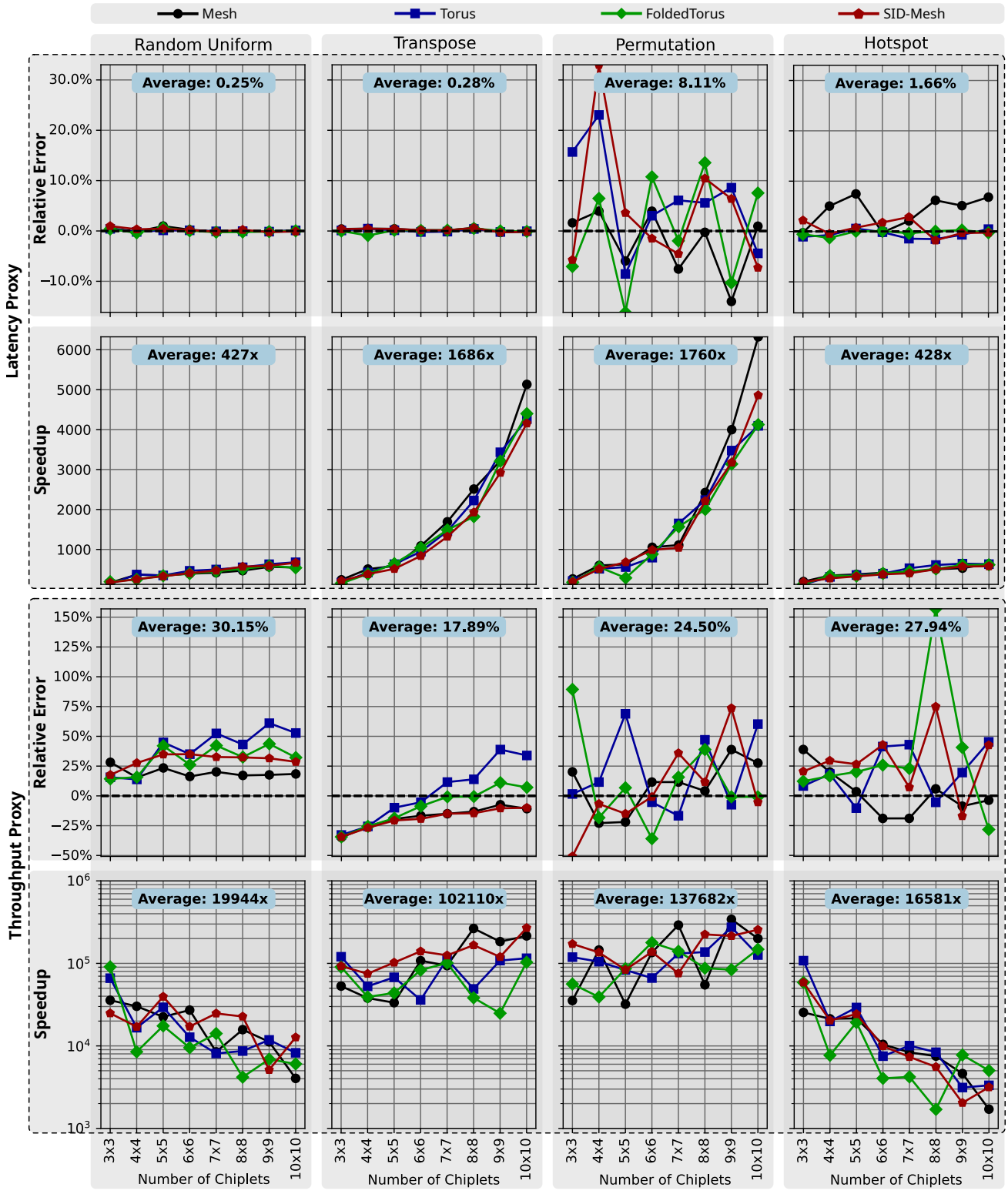

Figure 5: (§3.2) Evaluation of RapidChiplet's latency and throughput proxies compared to simulations in BookSim2.

4 Case Study

To showcase the possibilities that RapidChiplet unveils, we perform a case study in which we analyze the performance of the sparse Hamming graph (SHG) network-on-chip (NoC) topology, if used as an ICI. SHG [36] is a parametrizable NoC topology that spans the design space between a 2D mesh (low area but low performance) and a flattened butterfly (high area but high performance). For a grid of $R \times C$ chiplets, there are 2^{R+C-4} different parametrizations. Since for large designs, thousands of possible SHG parametrizations exist, the authors propose a manually driven selection strategy that does not explore the whole design space. We exploit the unprecedented speed of RapidChiplet to evaluate the 65'536 possible parametrizations of SHG for a 10×10 chiplet grid, which takes less than half a day on a consumer-grade laptop.

4.1 Experiment Setup

We use a 10×10 grid of homogeneous chiplets with the same specifications as in our evaluation (see Section 3.1) and we estimate the latency and throughput of each SHG parametrization under the random-uniform traffic pattern.

4.2 Experiment Results

Fig. 6 shows the results of the case study. Each point in the plot corresponds to one of the 65'536 SHG parametrizations, with the 2D mesh and flattened butterfly topologies highlighted. The color of the points represents the area overhead, ranging from 0% for the 2D mesh to 16% for the flattened butterfly. Lines show latency-throughput Pareto-fronts under different area-constraints (labeled with the area overhead vs. a 2D mesh).

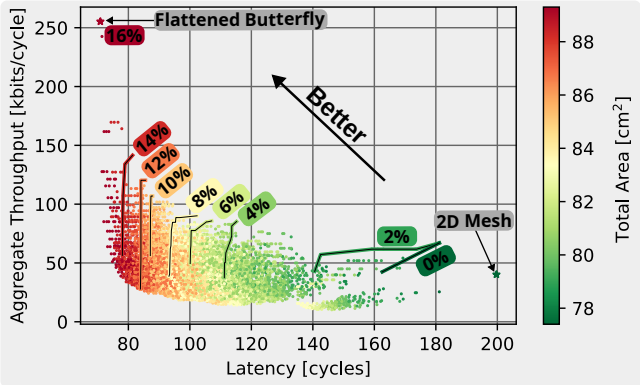

Figure 6: (§4.2) Design space exploration of the SHG topology. Points are configurations; lines are latency-throughput pareto-frontiers for different area-overheads.

We observe that a high area overhead is a necessary condition for both low latency and high throughput. While a larger area budget almost seems to be a sufficient condition for low latency, this does not hold for the throughput as there are many points which, despite having a high area overhead, suffer from a low throughput. We conclude that in order to achieve a high throughput under a given area-budget, finding a good parametrization is crucial. The best way to find this parametrization is through exhaustive search, which is enabled by RapidChiplet's fast latency and throughput proxies.

5 Related Work

Early-stage latency and throughput predictions for ICIs are often performed using cycle-based simulators such as BookSim2 [22], Noxim [23], Nostrum [24] or Garnet [25]. While these simulators were originally conceived to simulate NoC of monolithic chips, methodologies [37] to use them for simulations of 2.5D stacked chips have been proposed. As we have shown, such simulations take orders of magnitude longer than our novel, high-level latency and throughput proxies.

There exist numerous DSE-tools for other metrics, such as the Orion 2.0 [38] power and area model, the ChipletActuary [39] cost model, or the HotSpot [40] thermal simulator. RapidChiplet focuses on the latency and throughput of the ICI and only provides very high-level power, area, and cost estimates to assess the influence of the ICI onto those metrics.

Numerous works perform a DSE of chiplet architectures. Pal et al. [41] present a DSE for chiplet systems, focusing on finding a minimum set of chiplets as building blocks for a processor-family. Chiplet-Gym [42] is a reinforcement learning-based framework to explore the design space of chiplet-based accelerators for artificial intelligence workloads. Kim et al. [43] present a design flow that covers the architecture, circuit and packaging design phases of a 2.5D integrated circuit. Our work focusses on the DSE of the ICI specifically, rather than the overall architecture.

6 Conclusion

The design space for inter-chiplet interconnects (ICIs) is huge, as there are countless options for packaging, chiplet placement, D2D link implementation, inter-chiplet interconnect (ICI) topology and many more, which all influence the performance and cost of the ICI. Exploring this large design space requires a fast framework capable of capturing the complex interplay between these design choices.

As we are not aware of an existing toolchain satisfying these requirements, we propose RapidChiplet, a fast and comprehensive toolchain that provides high-level latency and throughput proxies for ICIs based on a multitude of configurable parameters. Compared to cycle-based simulations, our latency proxy achieves an average speedup of $1,075\times$ at an average error of 2.57%. Our throughput proxy achieves a higher average speedup of $69,079\times$ at a larger average error of 25.12%.

RapidChiplet can be used for DSEs where hundreds of thousands of design points are evaluated, or as a cost function to optimization algorithms or machine learning models. By providing RapidChiplet with its novel, high-level latency and throughput proxies, we facilitate DSE of ICIs, sparking future advancements in this active research area.

Acknowledgements

This work was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. It also received funding from the European Research Council (Project PSAP, No. 101002047), the European Union's HE research and innovation programme (Project GLACIATION, No. 101070141), and the European PILOT project which has received funding from the European High-Performance Computing Joint Undertaking (No. 101034126).

References

[1] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[2] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun, "Chiplet heterogeneous integration technology-status and challenges," *Electronics*, 2020.

[3] W. Gomes, A. Koker, P. Stover, D. Ingerly, S. Siers, S. Venkataraman, C. Pelto, T. Shah, A. Rao, F. O'Mahony *et al.*, "Ponte vecchio: A multi-tile 3d stacked processor for exascale computing," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2022.

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

[5] A. Patrizio, "Nvidia launches blackwell gpu architecture," Networkworld, 2024, accessed: 2024-08-19. [Online]. Available: https://www.networkworld.com/article/2066534/nvidia-launches-blackwell-gpu-architecture.html

[6] E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, V. Samant *et al.*, "The microarchitecture of dojo, tesla's exa-scale computer," *IEEE Micro*, 2023.

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

[8] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, C. Fuguet, I. Miro-Panades, G. Moritz, J. Durupt, C. Bernard, D. Varreau *et al.*, "IntAct: A 96-core processor with six chiplets 3D-stacked on an active interposer with distributed interconnects and integrated power management," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, 2020.

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

[10] K. Sikka, R. Bonam, Y. Liu, P. Andry, D. Parekh, A. Jain, M. Bergendahl, R. Divakaruni, M. Cournoyer, P. Gagnon *et al.*, "Direct bonded heterogeneous integration (dbhi) si bridge," in *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*. IEEE, 2021.

[11] A. Graening, S. Pal, and P. Gupta, "Chiplets: How small is too small?" in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

[12] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler, "Hexamesh: Scaling to hundreds of chiplets with an optimized chiplet arrangement," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[13] B. Dehlaghi and A. C. Carusone, "A 0.3 pj/bit 20 gb/s/wire parallel interface for die-to-die communication," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 11, 2016.

[14] Y. Nishi, J. W. Poulton, W. J. Turner, X. Chen, S. Song, B. Zimmer, S. G. Tell, N. Nedovic, J. M. Wilson, W. J. Dally *et al.*, "A 0.297-pj/bit 50.4-gb/s/wire inverter-based short-reach simultaneous bi-directional transceiver for die-to-die interface in 5-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, 2023.

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

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

[17] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014.

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

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

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

[21] B. Sharifpour, M. Sharifpour, and M. Reshadi, "Sid-mesh: Diagonal mesh topology for silicon interposer in 2.5 d noc with introducing a new routing algorithm," in *2021 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*. IEEE, 2021.

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

[23] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *2015 IEEE 26th international conference on application-specific systems, architectures and processors (ASAP)*. IEEE, 2015.

[24] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "Nnse: Nostrum network-on-chip simulation environment," *Proc. of SSoCC*, 2005.

[25] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009.

[26] J. Hestness, "Netraces v1.0," The University of Texas at Austin, 2024, accessed: 2024-08-21. [Online]. Available: https://www.cs.utexas.edu/~netrace/

[27] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, 2010.

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

[29] P.-H. Pham, P. Mau, and C. Kim, "A 64-pe folded-torus intra-chip communication fabric for guaranteed throughput in network-on-chip based applications," in *2009 IEEE Custom Integrated Circuits Conference*. IEEE, 2009.

[30] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007.

[31] A. Sahba and J. J. Prevost, "Hypercube based clusters in cloud computing," in *2016 World Automation Congress (WAC)*. IEEE, 2016.

[32] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022.

[33] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, 1992.

[34] L. Levitin, M. Karpovsky, and M. Mustafa, "Deadlock prevention by turn prohibition in interconnection networks," in *2009 IEEE international symposium on parallel & distributed processing*. IEEE, 2009.

[35] T. Caldwell, "On finding minimum routes in a network with turn penalties," *Communications of the ACM*, vol. 4, no. 2, pp. 107–108, 1961.

[36] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler, "Sparse hamming graph: A customizable network-on-chip topology," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[37] H. Zhi, X. Xu, W. Han, Z. Gao, X. Wang, M. Palesi, A. K. Singh, and L. Huang, "A methodology for simulating multi-chiplet systems using open-source simulators," in *Proceedings of the Eight Annual ACM International Conference on Nanoscale Computing and Communication*, 2021.

[38] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009.

[39] Y. Feng and K. Ma, "Chiplet actuary: A quantitative cost model and multi-chiplet architecture exploration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[40] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 14, no. 5, 2006.

[41] S. Pal, D. Petrisko, R. Kumar, and P. Gupta, "Design space exploration for chiplet-assembly-based processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, 2020.

[42] K. Mishty and M. Sadi, "Chiplet-gym: Optimizing chiplet-based ai accelerator design with reinforcement learning," *arXiv preprint arXiv:2406.00858*, 2024.

[43] J. Kim, G. Murali, H. Park, E. Qin, H. Kwon, V. Chaitanya, K. Chekuri, N. Dasari, A. Singh, M. Lee *et al.*, "Architecture, chip, and package co-design flow for 2.5 d ic design enabling heterogeneous ip reuse," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019.