A Performance Model for Disintegrated Manycores

Jiaxiang Li, Mark C. Jeffrey, and Natalie Enright Jerger

***Abstract*—Chiplet-based architectures are being adopted to improve manufacturing yield and reduce system cost. Unfortunately, disintegration from a monolithic system to a chiplet-based one hurts performance, as inter-chiplet links are typically slower than on-chip links. We propose the first analytical model to derive chiplet performance from the monolithic design's performance. We focus on modeling the impact of inter-chiplet interconnection to explore different system-level configurations. We validate our model with gem5 using an average error rate of 5.4%.**

***Index Terms*—Chiplets, analytical models**

I. Introduction

MODERN high-performance monolithic chips are approaching the reticle limit. Large monolithic chips have lower manufacturing yields, resulting in higher costs. Breaking one large chip into multiple smaller chiplets improves yield and reduces cost. One chiplet design can also be reused across different product lines, which amortizes design costs. Due to these advantages, chiplet-based designs are being widely adopted by multiple companies, such as Intel [1] and AMD [2].

Although chiplets improve manufacturing for large systems, they introduce performance penalties compared to monolithic designs. Inter-chiplet links have longer latencies and lower bandwidths compared to on-chip network links, which hurt the performance of inter-chiplet packets. Simulation-based studies demonstrate that these packet penalties variably hurt total run time across different workloads [3]. However, simulation approaches are limited in the number of workloads and configurations they can evaluate due to long simulation times. Hence, we need a straightforward approach to quickly reason about chiplet performance at a high level. Such an approach helps hardware engineers do early stage design exploration and software engineers optimize programs for chiplet systems.

Analytical models help us understand the impact of different design choices on the system's performance and enable rapid design-space exploration. We propose an analytical model for CPU chiplet systems to model the performance penalty introduced by the slow inter-chiplet links. Our model reduces the total simulation time from $O(mn)$ to $O(m)$ to evaluate m workloads on n different configurations. Considering the combination of different chiplet sizes and inter-chiplet link configurations, n can be a large number. We validate our model against simulation and the results show an average error rate of 5.4% across 8 Splash-4 workloads [4] running on 64 cores. We also present a performance-per-cost exploration case study combining our model and an existing chiplet cost model.

II. Background and Motivation

Chiplet-based design can improve yield and reduce cost. Multiple small chips are integrated together and connected by advanced packaging technology, such as an MCM, interposer, etc. Fig. 1 shows a 64-core monolithic chip split into 16 4-core chiplets. This approach achieves comparable system scale at a lower cost. However, package-level interconnects do not have equivalent latency and bandwidth as on-chip interconnects, leading to a performance penalty for chiplets.

Received 23 September 2025; revised 29 October 2025; accepted 30 October 2025. The authors are with the University of Toronto, Canada

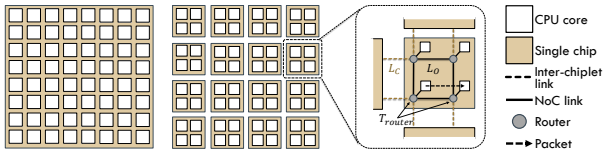


Fig. 1. Comparing 64-core monolithic and chiplet-based systems.

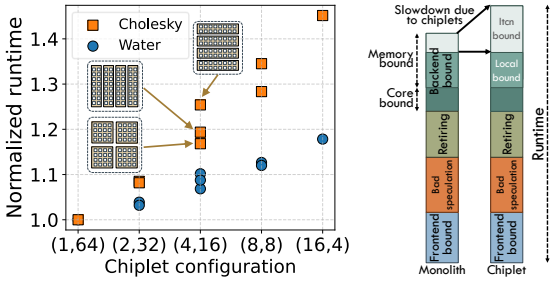


Fig. 2. (a) Normalized workload runtime across different chiplet size configurations on the x-axis (num chips, cores per chip). (b) An example Top-Down breakdown where chiplets increase backend-bound cyles.

As chiplet-based designs proliferate, it is important to understand the chiplet performance penalty. Prior work [3] simulates and correlates the performance degradation of running 29 benchmarks on two specific chiplet configurations, but is costly to extend to other workloads and configurations. Fig. 2a shows the runtime of Water and Cholesky from Splash-4 [4] under different configurations (see Sec. IV for methodology). We vary the chiplet row and column sizes between 2, 4, and 8. Each data point in Fig. 2a is labeled by the corresponding chiplet size configuration. For example, (4, 16) implies 4 chiplets with 16 cores per chiplet. For each chiplet size configuration, there can be multiple ways to split the system and Fig. 2a shows a (4, 16) system split in 3 different ways.

Under all configurations, Cholesky exhibits greater slowdown than Water—up to > 40% for Cholesky vs. < 20% for Water. They also show different sensitivity to the chiplet configuration. For the same chiplet sizes, Water shows consistent runtimes. However, for Cholesky, different chiplet configurations lead to significantly different runtimes, even when the size of chiplets is the same. These results show that the chiplet performance penalty cannot be explained by one or two factors. We need a systematic approach to understand and explain the chiplet performance penalty.

A. Motivation

An analytical performance model is a good candidate to understand performance penalties as it describes the hardware at a high level of abstraction [5], [6]. Traditionally, NoC analytical models use queueing theory [7]. These NoC models can be extended for chiplets, but they focus on packet latency and not workload runtimes. Moreover, applying this method to chiplets would be complex, due to the heterogeneity of on-chip and inter-chiplet links, and would require low-level details, like buffer sizes and traffic arrival rates.

A simple, high-level model would compare relative performance of chiplet and monolithic designs. We leverage the Top-Down method [8] for workload profiles. Fig. 2b shows an illustrative Top-Down breakdown of how cores spend their time (issue slots) across four top-level categories: *(i)* retiring, *(ii)* mispeculating, or stalled on the *(iii)* frontend or *(iv)* backend. By determining the extent to which parts are impacted by chiplets, we can calculate the relative performance of chiplet architectures. Assuming monolithic and chiplet-based multi-core designs share the same core IP, the CPU cycles associated with frontend, bad speculation, and retiring should remain the same. Hence, for a chiplet performance model that aims to derive chiplet performance from the performance of the monolith, the program runtime increase is approximately the increase of backend-bound CPU cycles. Within backend-bound cycles, the store-bound and L1 cache-bound cycles should also be the same, as this part of the traffic is still within a core and equivalent to the monolithic design. Based on this insight, our proposed model derives the ratio of chiplet performance (T_c) to monolith performance (T_m).

III. The Analytical Model

In this section, we describe our model. Table I shows the assumptions we make for the model. Our model supports homogeneous chiplet systems but does not apply to heterogeneous chiplets. We use subscripts c and m to represent terms for chiplets and monoliths. For example, T_c and T_m represent the workload runtimes on chiplet and monolithic systems.

A. How chiplets impact the latency of a packet

The NoC packet latency can be calculated as $T_{packet} = T_{wire} + T_{router} + T_{contention}$. The chiplet-induced increase in packet latency is therefore

$$\begin{aligned}\Delta T_{packet} &= T_{packet,c} - T_{packet,m} \\ &= T_{wire,c} - T_{wire,m} + T_{router,c} - T_{router,m} \\ &\quad + T_{contention,c} - T_{contention,m} \\ &= T_{wire,c} - T_{wire,m} \end{aligned} \tag{1}$$

where the final step follows from Assum. 4 and Assum. 6: for a given packet, T_{router} and $T_{contention}$ would not change from a monolith to a chiplet system. The wire latency for a packet that traverses h hops on a monolith is

$$T_{wire,m}(h) = L_o \cdot h + S/B_o \tag{2}$$

where L_o is the per-hop latency to traverse an on-chip link, B_o is the bandwidth of the link, and S is the size of the packet.

TABLE I
List of Assumptions

(1)	All chiplet and monolithic designs share the same core IP design and have the same number of cores.
(2)	The thread-to-core mapping is the same. We do not explore the impact of chiplet-specific mapping.
(3)	Cores are connected with the same mesh topology for chiplet and monolithic systems. We leave different chiplet topology comparisons as the future work.
(4)	Both on-chip and inter-chiplet links have the same bandwidth while inter-chiplet links have longer per-hop latency.
(5)	The total cycles the application is stalled by the interconnection is proportional to the average packet latency.
(6)	The interconnection networks operate in the non-saturation region and the change to $T_{contention}$ is negligible.

The same packet on a chiplet system traverses h_o on-chip links and h_c inter-chiplet links. The latter have higher per-hop latency (L_c) and lower bandwidth (B_c). The serialization latency is bounded by the bottleneck link along the path:

$$T_{wire,c}(h_o, h_c) = L_o h_o + L_c h_c + S/\min(B_o, B_c) \tag{3}$$

Assum. 2 and Assum. 3 imply that $h_o + h_c = h$. Therefore, combining Eq. 1, Eq. 2, and Eq. 3:

$$\begin{aligned}\Delta T_{packet}(h_o, h_c) &= T_{wire,c}(h_o, h_c) - T_{wire,m}(h_o + h_c) \\ &= (L_c - L_o) \cdot h_c + S \cdot \left(B_c^{-1} - B_o^{-1}\right) \\ &= (L_c - L_o) \cdot h_c \end{aligned} \tag{4}$$

The final step above follows from Assum. 4: $B_c = B_o$.

B. How chiplets impact the communication part of a workload

Chiplets potentially increase the time that cores stall on shared-memory communication. For a given workload, this communication comprises multiple packets that traverse the NoC or inter-chiplet network. These packets have different chiplet hop counts (h_c) and sizes, which are determined by both workload and topology. Given Assum. 5, the increase in cycles stalled due to the network depends on the average packet latency for the monolith and chiplet, or specifically:

$$\begin{aligned}\mathbb{E}[\Delta T_{packet}] &= \Sigma_{h_c} \Pr[h_c] \cdot \Delta T_{packet}(h_c) \\ &= \Sigma_{h_c} \Pr[h_c] \cdot (L_c - L_o) \cdot h_c \\ &= (L_c - L_o) \cdot \mathbb{E}[h_c] \end{aligned} \tag{5}$$

where the expectation sums over the set of inter-chiplet hop counts that could be taken by all packets, and $\Pr[h_c]$ accounts for the fraction of packets that traverse exactly h_c inter-chiplet links. In other words, the average chiplet-induced increase in packet latency is simply a linear function of the difference in link latency, depending on the workload's average packet chiplet hop count ($\mathbb{E}[h_c]$). Calculating $\mathbb{E}[h_c]$ requires the workload's node-to-node traffic distribution. Given Assum. 2, the traffic distribution should be the same across monolithic and chiplet systems. The traffic distribution for the monolith can be collected either through simulation or software profiling.

C. How chiplets impact workload runtime

We now model the normalized chiplet-induced increase in runtime, i.e., $\frac{T_c}{T_m}$, naively assuming the program uses no synchronization. We relax that assumption in Sec. III-D. We use the Top-Down runtime breakdown [8] to quantify the

CPU stall cycles that may be increased by chiplets. Following Assum. 1 and Sec. II-A, the affected cycles are those that potentially stall on inter-chiplet packets: L2-cache-bound, L3-cache-bound (if applicable), and memory-bound cycles. We aggregate these three stall types into *combined interconnect-bound* (*itcn*) cycles.

As distinguished by the top grey bars of Fig. 2b, the runtime of a workload comprises interconnect-bound and non-interconnect-bound cycles, averaged across threads.

$$T_m = \mathbb{E}[T_{itcn,m}] + \mathbb{E}[T_{\neg itcn,m}]$$
$$T_c = \mathbb{E}[T_{itcn,c}] + \mathbb{E}[T_{\neg itcn,c}]$$

With Assum. 1 and no synchronization, $\mathbb{E}[T_{\neg itcn,m}]$ and $\mathbb{E}[T_{\neg itcn,c}]$ are the same. The performance difference should arise from interconnect-bound cycles:

$$T_c - T_m = \mathbb{E}[T_{itcn,c}] - \mathbb{E}[T_{itcn,m}] \tag{6}$$

Following Assum. 5 and substituting Eq. 5,

$$\frac{\mathbb{E}[T_{itcn,c}]}{\mathbb{E}[T_{itcn,m}]} = \frac{\mathbb{E}[T_{packet,c}]}{\mathbb{E}[T_{packet,m}]} = \frac{\mathbb{E}[T_{packet,m}] + \mathbb{E}[\Delta T_{packet}]}{\mathbb{E}[T_{packet,m}]}$$
$$= 1 + \frac{(L_c - L_o) \cdot \mathbb{E}[h_c]}{\mathbb{E}[T_{packet,m}]} \tag{7}$$

Let $f_{itcn,m} = \frac{\mathbb{E}[T_{itcn,m}]}{T_m}$ be the fraction of total workload time spent stalled on interconnect-bound cycles by the monolith. Let $\alpha_{itcn} = \frac{\mathbb{E}[T_{itcn,c}]}{\mathbb{E}[T_{itcn,m}]}$. Then, combining Eq. 6 and Eq. 7, the relative increase in workload runtime on the chiplet is

$$\frac{T_c}{T_m} = 1 + \frac{\mathbb{E}[T_{itcn,c}]}{T_m} - \frac{\mathbb{E}[T_{itcn,m}]}{T_m} = 1 + f_{itcn,m}\,(\alpha_{itcn} - 1)$$
$$= 1 + f_{itcn,m} \cdot \frac{(L_c - L_o) \cdot \mathbb{E}[h_c]}{\mathbb{E}[T_{packet,m}]} \tag{8}$$

In words, Eq. 8 scales the monolith-normalized increase in average packet latency (the fraction term) by the ratio of time a monolith's cores stall on interconnect-bound cycles ($f_{itcn,m}$). $\mathbb{E}[h_c]$, $\mathbb{E}[T_{packet,m}]$, and $f_{itcn,m}$ are empirically measured through simulation or profiling each workload on the monolithic system. The difference in link latencies is constant.

D. How synchronization impacts workload runtime

Sec. III-C naively assumes the workload does not use synchronization but this can underestimate the performance degradation of chiplets. Load imbalance or high contention causes some threads to wait at a barrier or on a lock, respectively. For example, Fig. 3 shows four threads using one lock to grant exclusive access to a critical section. Thread 1 acquires the lock first, then runs other thread-private code afterward. It would likely stall with interconnect-bound cycles for lock acquire (transfer), accesses to shared mutable data, and lock release. Meanwhile, the other threads wait on the lock. With local[1] spinning synchronization, most cycles spent waiting are accounted in the Top-Down method as retiring or L1-bound stalls; with blocking synchronization, the thread yields. In both cases, a thread's waiting will not be counted as interconnect-bound cycles.

[1]E.g., test-and-test-and-set instead of test-and-set

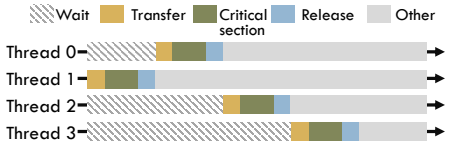


Fig. 3. An example of 4 threads accessing one critical section using a lock.

Synchronization invalidates Eq. 6. On one hand, chiplets increase the cycles a thread spends acquiring/releasing a contended lock as well as the time it spends in the critical section, because these can cause interconnect-bound stalls. Eq. 7 accounts for this. On the other hand, chiplets increase the cycles spent by waiting threads, but these would fall under $T_{\neg itcn}$. We address this by augmenting Eq. 6 to further decompose $T_{\neg itcn}$ into time spent waiting on synchronization and the other non-interconnect, non-waiting cycles:

$$T_c - T_m = \mathbb{E}[T_{itcn,c}] + \mathbb{E}[T_{wait,c}] - \mathbb{E}[T_{itcn,m}] - \mathbb{E}[T_{wait,m}] \tag{9}$$

$\mathbb{E}[T_{wait,m}]$ is determined both by how many threads are waiting for each critical section and the average synchronization period [9] (latency of transfer, critical section, and release). Following Assum. 2, the number of waiting threads for each critical section is the same for the monolith and chiplets. Consequently, the relative increase in time spent waiting ($\frac{\mathbb{E}[T_{wait,c}]}{\mathbb{E}[T_{wait,m}]}$) is proportional to the relative increase of time in critical sections. We make a simplifying assumption to determine this relative increase of time in critical sections: critical section code has similar interconnect-bound behavior as thread-private code. Mathematically, we assume the ratio of interconnect-bound cycles to non-waiting and non-interconnect-bound cycles is the same within and outside critical sections. Let $f_{wait,m} = \frac{\mathbb{E}[T_{wait,m}]}{T_m}$. Taken all together,

$$\frac{\mathbb{E}[T_{wait,c}]}{\mathbb{E}[T_{wait,m}]} \approx \frac{T_c - \mathbb{E}[T_{wait,c}]}{T_m - \mathbb{E}[T_{wait,m}]}$$
$$= \frac{\mathbb{E}[T_{itcn,c}] - \mathbb{E}[T_{itcn,m}] + T_m - \mathbb{E}[T_{wait,m}]}{T_m - \mathbb{E}[T_{wait,m}]}$$
$$= \frac{f_{itcn,m}}{1 - f_{wait,m}}\,(\alpha_{itcn} - 1) + 1 \tag{10}$$

where the second step substitutes $T_c - \mathbb{E}[T_{wait,c}]$ from Eq. 9.

Let $\alpha_{wait} = \frac{\mathbb{E}[T_{wait,c}]}{\mathbb{E}[T_{wait,m}]}$. Combining Eq. 9 and Eq. 10:

$$\frac{T_c}{T_m} = 1 + \frac{\mathbb{E}[T_{itcn,c}]}{T_m} - \frac{\mathbb{E}[T_{itcn,m}]}{T_m} + \frac{\mathbb{E}[T_{wait,c}]}{T_m} - \frac{\mathbb{E}[T_{wait,m}]}{T_m}$$
$$= 1 + f_{itcn,m}\,(\alpha_{itcn} - 1) + f_{wait,m}\,(\alpha_{wait} - 1)$$
$$\approx 1 + \left(f_{itcn,m} + \frac{f_{wait,m} \cdot f_{itcn,m}}{1 - f_{wait,m}}\right) \cdot (\alpha_{itcn} - 1)$$

Substituting Eq. 7 and simplifying, we present our analytical model for the performance degradation of a chiplet system, normalized to the monolith:

$$\boxed{\frac{T_c}{T_m} \approx 1 + \frac{f_{itcn,m}}{1 - f_{wait,m}} \cdot \frac{(L_c - L_o) \cdot \mathbb{E}[h_c]}{\mathbb{E}[T_{packet,m}]}} \tag{11}$$

Eq. 11 generalizes Eq. 8 by normalizing $f_{itcn,m}$ by non-waiting time. Similar to Eq. 8, L_o and L_c are constant and the rest are empirically measured on a monolithic system.

TABLE II
SIMULATION CONFIGURATION

Cores	64 2GHz OoO x86 cores
Cache hierarchy	MESI two-level protocol 64 KB L1D cache per core; 32 KB L1I cache per core; 1 MB L2 cache per core
Memory	15 ns latency, 40 GB/s per channel
Topology	8×8 Mesh with 16 memory controllers on both sides.
Interconnection config	1 cycle for all routers; 1 cycle for NoC links 9 cycles for chiplet links; Use X-Y routing for all configs

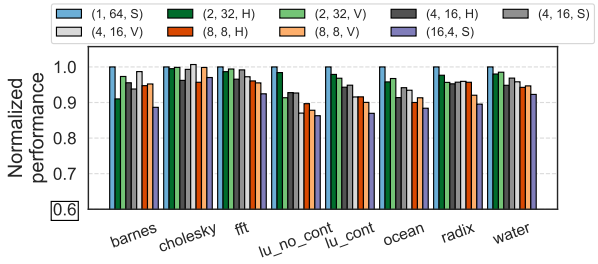


Fig. 4. Model-predicted chiplet runtime normalized by simulated runtime. H, V, and S indicate whether the chiplets are laid horizontal or vertical rectangulars or squares.

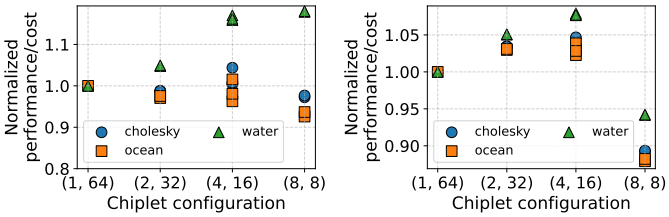
IV. EVALUATION

Validation. We employ the gem5 simulator [10] with Garnet to validate the accuracy of our model. Table II shows our configuration. We sweep the chiplet row and column size between 2, 4, and 8. For Top-Down profiling, we integrated an existing approach [11] to gem5. We use 8 Splash-4 [4] workloads and measure the runtime for region of interest.

We normalize the model prediction to the gem5 runtime for each configuration as shown in Fig. 4; effectively, this gives the error of the model for each configuration and workload. We also calculate the error rate by $\frac{|prediction-simulation|}{simulation}$. Across all the simulated configurations, our model shows a 5.4% average error rate with a worst case of 13.7%. Both versions of LU show relatively low prediction accuracy due to the uneven distribution of workload to threads. Threads that have finished the assigned job idle and wait for the last threads to finish. This irregularity results in a non-uniform packet pattern which breaks our Assum. 5. Also, larger chiplet configurations have higher prediction accuracy. We also sweep the inter-chiplet latency from 3 cycles to 18 cycles to validate the generality of our model. The results shows an average error rate of 6.3%.

Case Study. We demonstrate the utility of our model through a design space exploration of performance-per-cost (perf/$) under 8 chiplet configurations (excluding (16,4)) with 3 different workloads. We evaluate two different packaging technology: multi-chip module (MCM) and silicon interposer (SI). Both configurations follow the parameters in Table II, except the per-chiplet hop for SI is set to 3 cycles. We calculate the recurring engineering cost using the Chiplet Actuary [12]. We follow the die area assumption ($76mm^2$ per 8 cores) [12] and scale it by number of cores on the chiplet.

Fig. 5a and Fig. 5b show the normalized perf/$ for two different packaging technologies. We classify whether a workload is chiplet sensitive based on $\beta = \frac{f_{itcn,m}}{1-f_{wait,m}}$. Chiplet-insensitve workloads (e.g., Water, β=0.11) shows the best perf/$ on small chiplets as they save cost. However, chiplet-sensitive workloads (e.g., Ocean, β=0.61) prefer the monolithic design due to the increase of relative chiplet runtime. From the packaging technology perspective, MCM and SI show two different trends. MCM shows a big variation of perf/$, ranging from 1.18× improvement at best and less than 0.92× at worst. Cost saving overweights performance degradation for chiplet-insensitive workloads, However, for chiplet-sensitive workloads like ocean, it shows worse perf/$ compared to the monolithic configuration. For SI, the perf/$ does not vary much. The cost saving is not as significant as MCM due to the higher packaging costs. However, the performance penalty is also relatively low. As a result, the SI has perf/$ improvement for all workloads in (2,32) and (4,16) configurations.



(a) Multi-chip module (b) Silicon interposer

Fig. 5. Normalized performance-per-cost (higher is better).

V. CONCLUSION

We introduce an analytical model to understand the performance degradation from a monolithic to a chiplet systems. We validate our model through simulation and have an average error rate of 5.4%. Our case study demonstrates the utility of our model for rapid design space exploration.

ACKNOWLEDGEMENTS

We sincerely thank reviewers and research group members for their helpful feedback. We gratefully acknowledge the support of NSERC and CRC.

REFERENCES

[1] N. Nassif *et al.*, "Sapphire Rapids: The next-generation Intel Xeon Scalable Processor," in *ISSCC*, vol. 65, 2022.
[2] R. Bhargava *et al.*, "AMD next-generation "Zen 4" core and 4th gen AMD EPYC server cpus," *IEEE Micro*, vol. 44, no. 3, 2024.
[3] I. R. Brkić *et al.*, "Disintegrating manycores: Which applications lose and why?" in *NoCArc*, 2023.
[4] E. J. Gómez-Hernández *et al.*, "Splash-4: A modern benchmark suite with lock-free constructs," in *IISWC*, 2022.
[5] M. D. Hill *et al.*, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, 2008.
[6] S. Eyerman *et al.*, "Modeling critical sections in Amdahl's law and its implications for multicore design," in *ISCA*, 2010.
[7] U. Y. Ogras *et al.*, "An analytical approach for network-on-chip performance analysis," *TCAD*, vol. 29, no. 12, 2010.
[8] A. Yasin, "A top-down method for performance analysis and counters architecture," in *ISPASS*, 2014.
[9] A. Kägi *et al.*, "Efficient synchronization: Let them eat QOLB," in *ISCA*, 1997.
[10] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint*, 2020.
[11] J. M. Cebrian *et al.*, "Semi-automatic validation of cycle-accurate simulation infrastructures: The case for gem5-x86," *FGCS*, 2020.
[12] Y. Feng *et al.*, "Chiplet actuary: A quantitative cost model and multi-chiplet architecture exploration," in *DAC*, 2022.