Know Your Neighbor: Physically Locating Xeon Processor Cores on the Core Tile Grid

Hyungmin Cho
Department of Computer Science and Engineering
Sungkyunkwan University, Suwon, Republic of Korea
hyungmin.cho@skku.edu

***Abstract*—The physical locations of the processor cores in multi- or many-core CPUs are often hidden from the users. The current generation Intel Xeon CPUs accommodate many processor cores on a tile grid, but the exact locations of the individual cores are not plainly visible. We present a methodology for physically locating the cores in the Intel Xeon CPUs. Using the method, we collect core location samples of 300 CPU instances deployed in a commercial cloud platform, which reveal a wide variety of core map patterns. The locations of the individual processor cores are not contiguously mapped, and the mapping pattern can be different per each CPU instance.**

We also demonstrate that an attacker can exploit an inter-core thermal covert channel using the identified core locations. The attacker can increase the channel capacity by strategically placing multiple sender and receiver nodes. Our evaluation shows that up to 15 bps of data transfer is possible with less than 1% of bit error rate on a cloud environment, which is 3 times higher than the previously reported results.

***Index Terms*—topology, network on chip, covert channel**

I. Introduction

On a large-scale many-core CPU, identifying the physical locations of individual cores may not be a trivial problem. Some CPUs expose their core locations [1], but the Intel Xeon CPUs do not publicly report their core location patterns. The cores may be located on the CPU die in a non-contiguous fashion, and the IDs of the cores are usually not analogous to the IDs recognized by the operating system (*OS core ID*). The manufacturer may intentionally obfuscate the core locations for various reasons, such as to have a better power distribution by placing the threads to different positions of the core die. Our core location mapping results on 300 Intel Xeon CPU instances reveal that even the individual CPU instances of the same model may not share the same mapping pattern (Sec. III).

We monitor the on-chip interconnect traffic between the cores and reconstruct the *unknown* core map that satisfies the traffic observations. We mainly test our method using the first and second-generation Xeon Scalable processors (*i.e.*, the Skylake and the Cascade Lake server architectures), but we also confirmed that our mechanism works for the latest third-generation Xeon processors (*i.e.*, the Ice Lake server architecture) as well.

This work was supported in part by Institute of Information communications Technology Planning Evaluation (IITP) grants funded by the Korea government (MSIT) Research on CPU vulnerability detection and validation (No. 2019-0-00533), High-Potential Individuals Global Training Program (No. 2020-0-01550), and Development of high speed encryption data processing technology that guarantees privacy based hardware (No. 2021-0-00779).

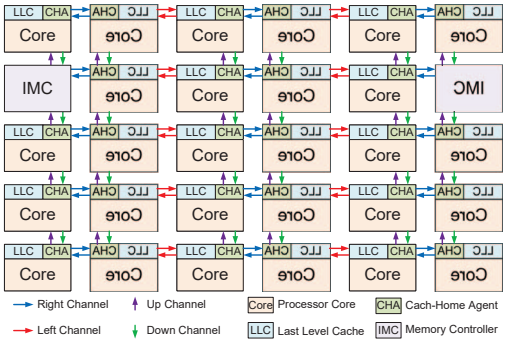


Fig. 1: Core tile grid on a Xeon CPU die.

The identified core locations can be a potential security threat if an attacker attempts location-based attacks, such as traffic contention side channel [2]. As a demonstration of such a location-based attack, we evaluate *inter-core* thermal covert channels on a commercial cloud platform. An inter-core thermal covert channel relies on the propagation of heat from a core to the neighboring cores as the mean of *unauthorized* communication. Therefore, identifying the relative positions of the cores is a crucial requirement. Previous work has demonstrated the possibility of the inter-core thermal channel but was limited to CPUs with a few cores only [3].

The contributions of this paper are as follows:

1) We present a method that can determine the physical locations of the cores of the latest Intel Xeon processors.
2) We collect a large number of processor core location map samples and revealed that a wide variety of map patterns exist even in the same CPU model.
3) Using the identified core locations, we evaluate a thermal covert channel on a commercial cloud platform. The results show that an inter-core thermal covert channel is feasible with a communication capacity of up to 15 bps with a bit error rate of less than 1%.

II. Locating Processor Cores

The Intel Xeon processors from the Skylake generation adopt a mesh interconnect that replaces the previous ring interconnect [4]. A simple dimension order routing is used on the Xeon mesh interconnect. A packet always travels through the vertical (*up* or *down*) channels first and then proceeds to the target using the horizontal (*left* or *right*) channels.

Each node in the mesh interconnect is a *core tile* that contains a processor core and a *slice* of the shared last level cache (LLC). The LLC slice is connected to the mesh interconnect through the Cache-Home Agent (CHA) module at each tile. Figure 1 shows a grid of Xeon core tiles that support up to 28 core tiles.

We assume the monitoring tool has access to the processor core's *Model Specific Registers* (MSRs) during the locating process (*i.e.*, requires root privileges). The Intel processors expose the *Protected Processor Inventory Number* (PPIN) that uniquely identifies each instance of the CPU chip. Once we map the core locations of a CPU instance, we can associate the core map with the PPIN.

Our mapping process consists of the following three steps:

1) OS core ID $\leftrightarrow$ CHA ID mapping
2) Inter-tile traffic generation and monitoring
3) Reconstructing the core map through an ILP problem

A. OS core ID $\leftrightarrow$ CHA ID mapping

To artificially generate noticeable data traffic between two core tiles, we need to place user-level worker threads to the source and sink tiles. Placing a thread to a specific core has to be done using the OS core ID.

On the contrary, mesh traffic monitoring has to be done based on the ID of the CHA node (*i.e.*, *CHA ID*). We use the *uncore performance monitors* (uncore PMON) to monitor the mesh traffic at each core tile [5]. The uncore PMON assigns different MSR address ranges per each CHA. To measure the amount of traffic that passes through a specific core tile, the monitoring program needs to know the CHA ID of the tile.

Unfortunately, the mapping between the OS core ID and the CHA ID is not contiguous nor monotonic. Even seriously, the mapping can be different per CPU instance. Therefore, the first step of the core location mapping should be identifying the mapping between the CHA ID and the OS core ID.

We find the mapping by monitoring data exchange between the cores and the LLC slices. If we select a core and an LLC slice from different tiles, they have to communicate through the mesh network. The generated traffic can be monitored by the uncore PMON. If we test all combinations of a core and an LLC slice, we can uniquely find the core-LLC pairs that do not generate inter-tile traffic between them. Such pairs are a core and an LLC slice that belong to the same tile, and they represent the OS core ID $\leftrightarrow$ CHA ID mapping.

We generate data exchange between a core and an LLC slice by repeatedly evicting cache lines from the core to the LLC slice. To generate such targeted evictions, we need to form a *slice eviction set*. A slice eviction set is a group of cache lines that belong to the same LLC slice as well as to the same L2 cache set. If the set contains more cache lines than the L2 cache set associativity, repeatedly accessing the cache lines in the set will cause evictions to the targeted LLC slice.

To form such slice eviction sets, we need to identify which of the LLC slices a cache line belongs to. However, such a slice mapping rule is undisclosed. Instead of fully deciphering the slice mapping rule, we may employ a similar strategy as previous work [6] to form slice eviction sets. In this work, we can use a simpler approach since we use the uncore PMON during the core locating process. We spawn a pair of worker threads that are pinned to two different cores. Those threads continuously perform simultaneous write accesses on the same cache line. We check the LLC look-up counts using uncore PMON and pick the CHA with the largest LLC lookup counts as the home of the tested cache line. We repeat the process until we collect enough cache lines to form a slice eviction set.

B. Inter-core Traffic Generation and Monitoring

We generate data traffic from a source to sink tile as follows.

- Pick a cache line that belongs to the sink tile LLC.
- Place a thread on the source tile and repeatedly perform writes on the selected cache line. The modified data will be stored in the private L2 cache of the source tile.
- Place another thread on the sink node and repeatedly perform reads on the same cache line. The modified data in the source node has to be transferred to the sink node.

We use the VERT_RING_BL_IN_USE counters for the up and the down channels and the HORZ_RING_BL_IN_USE counters for the left and right channels. These counters record the number of cycles the data channel is occupied.

If we can fully observe the utilized paths between all pairs of tiles, mapping the relative tile locations would be a trivial problem. Unfortunately, we can obtain only partial observations due to the following reasons.

1) **Input-only monitoring**: The counters monitor the number of occupied *ingress* cycles. We can monitor which *input* channel (*i.e.*, direction) is utilized at each CHA, but we do not know which *output* channel is utilized.
2) **None-core tiles**: Not all tiles contain a pair of a core and an LLC slice. For instance, some core tiles are occupied by the integrated memory controller (IMC).
3) **Unused (disabled) core tiles**: The Xeon processors provide many different core count configurations, but there are only a few tile grid sizes. Depending on the core count, there could be unused core tiles. For example, the 20-core configuration is manufactured using a CPU die with 28 core tiles, with eight unused core tiles. The unused tiles are disabled cores that may have been deactivated due to defects or process variations. A tile with a disabled core is still a valid mesh node that serves the traffic that passes through the tile. However, we cannot observe the traffic on those disabled tiles since the uncore PMON counters are disabled as well.
4) **LLC-only tiles**: Some Xeon configurations have core tiles consists of an active LLC slice and a disabled core. We can monitor the traffic that passes through these tiles since the uncore PMON of these tiles are not disabled, but we cannot utilize these tiles as a source or sink node.

Figure 2 depicts a situation where the partial observation caused by the disabled cores hinders the core mapping. If we monitor the traffic pattern between A and D (the red and blue arrows), we only observe that the horizontal input channels are activated. The traffic actually passed through the disabled tiles B and C, but the activated vertical channels are

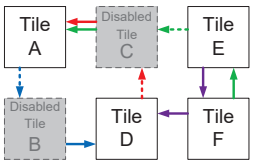


Fig. 2: An example of a partially observed inter-tile traffic pattern. Arrows with dotted lines represent unobservable channels due to the disabled core tiles.

not observable. We can identify that A and D are located in different tile columns (since a horizontal channel is activated), but the observation does not reveal any information on the relative row positions of A and D.

The relative row positions of A and D can be determined based on traffic patterns involving other core tiles. We can identify that E and F are located in different rows by the traffic between the two tiles. If we generate traffic from E to D (the purple arrows), it will create traffic that involves a vertical (down) input channel to F. This means that D is located on the same row as F or on a row below F. Similarly, the traffic from F to A (the green arrows) reveals that A is on the same row as E or on a row above E. By combining these observations, we can conclude that A and D are on different core tile rows.

C. ILP formulation

To reconstruct the core mapping that satisfies all traffic pattern observations, we formulate an integer linear programming (ILP) problem. We map N core tiles on a $T_h \times T_w$ tile grid.

1) Variables for Core Tile Positions: The R and C integer variables indicate the row and column indices of the core tiles.

$$0 \leq R_i < T_h \quad and \quad 0 \leq C_i \leq T_w, \quad i \in [0...N-1]$$

2) Alignment with Source and Sink: In the Xeon's dimension order routing, all vertical routes are fulfilled before making any horizontal movement. This implies that the intermediate tile nodes that receive the vertical inputs have the same tile column index as the source node (s). Similarly, the tile nodes that receive horizontal inputs have the same tile row index as the sink node (e). These constraints are expressed as follows.

$$C_i = C_s \quad and \quad R_j = R_e$$

where i and j represent all intermediate tile nodes that receive vertical and horizontal channel input, respectively.

3) Vertical Bounding Box Constraints: All intermediate tile nodes are enclosed within the bounding box determined by the source and the sink nodes as Fig. 3 shows. As the example in Fig. 3, suppose the observed traffic pattern has vertical routes through the *up* channels. This means that the generated traffic is traveling upwards to the sink node while passing through the intermediate nodes. For all intermediate nodes i in each observed paths p, the following constraints hold.

$$R_s > R_k \geq R_e \tag{1}$$

where s and e are the source node and the sink node of the path p, respectively, and k represents all intermediate nodes in the path p. If we observe a traffic path that uses *down* channels, the constraints (1) are applied with the reversed inequalities.

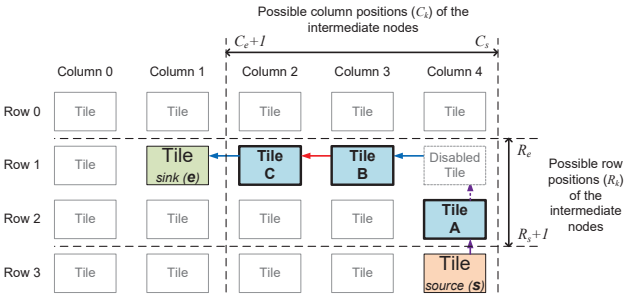


Fig. 3: An example of bounding box constraints.

4) Horizontal Bounding Box Constraints: The core tiles in every odd column are flipped horizontally on the Xeon tile grid [4]. Packets that travel horizontally will encounter alternating channel types (left and right) regardless of the travel direction (either westbound or eastbound). That is, we cannot distinguish the true direction of the horizontal movement just by a traffic path observation only.

We need an ILP problem that encompasses constraints for *both* directions while making the constraint that corresponds to the actual direction is exclusively enforced. To do so, we introduce new binary variables NE_p and NW_p per each observed path p. The role of these binary variables is nullifying the effect of a constraint. For each horizontal path observation, we add the following set of constraints.

$$C_s \leq C_k + b \cdot NE_p \quad and \quad C_k < C_e + b \cdot NE_p \tag{2}$$

$$b \cdot NW_p + C_s \geq C_k \quad and \quad b \cdot NW_p + C_k > C_e \tag{3}$$

(2) and (3) enforce the bounding box constraints assuming the observed path is eastbound and westbound, respectively. s, e, and k represent the source, sink, and intermediate nodes as before. In (2), the column indices increase as move from the source to the sink whereas the indices in (3) are the opposite.

The coefficient b of the constraint-nullifying variables is an arbitrarily large constant. If NE_p or NW_p becomes 1, the inequality trivially holds regardless of the column indices. This effectively voids the corresponding constraint. The following constraint enforces only one of (2) or (3) is nullified and the other is applied in the solution.

$$NE_p + NW_p = 1$$

5) Indicator variables: Additional variables are required to form the objective function. The following binary variables express the assigned row and column indices in one-hot encoding.

$$\sum_r OHR_{i,r} = 1 \quad and \quad \sum_c OHC_{i,c} = 1$$

$$i \in [0...N-1], \quad r \in [0...T_h-1], \quad c \in [0...T_w-1]$$

The following constraints keep the conversion between the index variables and the corresponding one-hot variables.

$$R_i = \sum_r r \cdot OHR_{i,r} \quad and \quad C_i = \sum_c c \cdot OHC_{i,c}$$

We create a set of binary indicator variables, one for each core tile row and column.

$$RI_r \quad and \quad CI_c, \quad r \in [0...T_h - 1], \quad c \in [0...T_w - 1]$$

The RI indicator variables become 1 if and only if the corresponding row is occupied by at least one core tile. The following constraint enforces such a relationship.

$$RI_r \leq \sum_i OHR_{i,r} \leq b \cdot RI_r$$

where b is a large constant value. The left inequality prevents the indicator variables from being 1 if no core tile is assigned to the row. The right inequality forces the indicator variable to be 1 if at least one core tile is assigned to the row. The same rule is applied to the CI variables for the column assignments.

6) Objective Function: Using the indicator variables, we create an objective function that has to be minimized.

$$minimize \left(\sum_{r=0}^{T_h-1} k \cdot RI_r + \sum_{c=0}^{T_w-1} k \cdot CI_c \right)$$

The objective function is a weighted sum of the indicator variables, with the higher weights for the larger row or column positions. The ILP problem finds the tightest core mapping that satisfies the constraints.

D. Mapping Failures

Our ILP problem fails to find the exact row or column indices if there is a fully vacant tile row or column. However, even for those cases, the relative location between the core tiles is correctly mapped.

III. Core Map Locating Results

We test our mapping technique on the AWS cloud to collect a large number of samples. To access the MSRs, we use the "*bare metal* " type EC2 instances. We collect core mapping of 100 CPU instances from each of the Xeon Platinum 8124M, 8175M, and 8259CL models. The 8124M CPUs have 18 cores, and the 8175M and 8259CL CPUs provide 24 cores each.

A. OS core ID $\leftrightarrow$ CHA ID Mapping Results.

Table I presents the CHA ID mapping results of the evaluated CPU instances. The ID mapping for the 8124M and the 8175M CPUs are rather regular, although not continuous. The numbers are grouped with strides of 4. Also, all 100 CPU instances share the same OS core ID $\leftrightarrow$ CHA ID mapping. However, the mapping results for the 8259CL CPUs are not uniform across the instances. We obtained 7 different ID mapping cases out of the 100 CPU instances. The 8259CL CPUs have a non-contiguous CHA ID space, resulting from the LLC-only tiles. For example, in the most frequently observed mapping that has 62 instances, the core tile with the CHA ID 3 is not mapped to any OS core ID. This core tile is used as the LLC-only core tile. The core tile with the CHA ID 25 is another LLC-only core tile. These LLC-only core tiles might be the reason why the 8259CL CPUs have non-uniform CHA ID mapping.

B. Core Mapping Results on the Core Tile Grid

Table II provides the statistics of the observed location patterns to show the diversity of the core location maps. Some patterns are more frequently observed than others. For example, 53 out of the 100 8124M CPU instances use the same core location mapping. However, many CPU instances use different core location mappings. For the 8124M CPUs, we observed a total of 14 different mapping patterns out of the tested 100 CPU instances. For the 8175M and the 8259CL CPUs, we observed 26 and 53 unique location patterns, respectively.

Figure 4 shows the three most frequently observed core location patterns of the 8259CL CPUs. The OS core ID and CHA ID are indicated on each tile. The mapping results do expose some degree of regularity: the CHA IDs are numbered in the column-major order, skipping disabled tiles. The other two CPU models also exhibit the same pattern rule. However, to draw such a generalization, a large number of mapping samples are required. Also, a new mapping rule may have to be identified on every different CPU model or generation.

We test our mapping mechanism on the latest Xeon processor as well. The third-generation "*Ice Lake*" Xeon Scalable processors support up to 40 cores in a CPU. Due to the current limited availability of the Ice Lake processors, we mapped the core locations of 10 Xeon 6354 CPU instance provided by the Oracle Cloud Infrastructure. Out of the evaluated 10 CPU instances, we found 6 unique core mapping patterns. Figure 5 shows a mapping example of the Xeon 6354 CPU, which has 18 cores mapped on an 8×6 tile grid. The new CHA ID location pattern is clearly different from the previous Xeon generations.

IV. Thermal Covert Channel

Our approach for creating the inter-core thermal covert channels is similar to the previous work [3], [7]. The sender node creates observable thermal fluctuations by controlling the CPU's load, and the receiver node observes the heat using the readings from the temperature sensor of the receiver core. The temperature sensors provide the current temperature of individual processor cores at 1°C granularity. Although the `lm-sensors` utility commonly found in many Linux distributions provides user-level access to *all* processor core temperature readings, we conservatively assume that the attacker can only access the thermal sensor of the core that is currently executing the application thread, as in [3], [7].

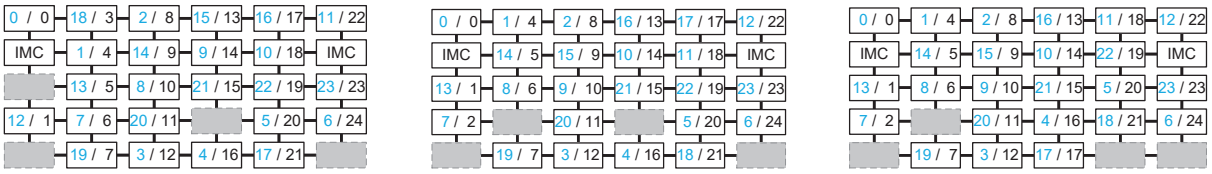
Bartolini *et al.* used the `lstopo` utility to identify the neighboring cores [7]. However, the `lstopo` utility only provides *logical* topology information. There is only a small chance that the cores with consecutive IDs reported by `lstopo` are actually located next to each other on a large-scale Xeon.

We use the identified core locations to select the sender and the receiver cores. Although our core mapping process requires root privileges, the identified core locations are permanent on a CPU instance. Other than the identified core locations, we assume the attackers only have user-level access.

A simple way to defend against a thermal covert channel would be blocking the user-level access to the temperature sensors. Reducing the resolution or the update frequency of the

TABLE I: The OS core ID to the CHA ID mapping results

CPU model	# of instances	OS core ID:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
8124M	100	**CHA ID:**	0	4	8	12	16	2	6	10	14	1	5	9	13	17	3	7	11	15	-					
8175M	100	**CHA ID:**	0	4	8	12	16	20	2	6	10	14	18	22	1	5	9	13	17	21	3	7	11	15	19	23
8259CL	62	**CHA ID:**	0	4	8	12	16	20	24	2	6	10	14	18	22	1	5	9	13	17	21	7	11	15	19	23
	33	**CHA ID:**	0	4	8	12	16	20	24	6	10	14	18	22	1	5	9	13	17	21	3	7	11	15	19	23
	1	**CHA ID:**	0	4	8	12	16	20	24	2	6	10	14	18	22	1	9	13	17	21	3	7	11	15	19	23
	1	**CHA ID:**	0	4	8	12	16	20	24	2	6	10	14	18	22	1	5	9	13	17	21	25	7	11	15	19
	1	**CHA ID:**	0	4	8	12	20	24	6	10	14	18	22	1	5	9	13	17	21	25	3	7	11	15	19	23
	1	**CHA ID:**	0	4	8	12	16	20	2	6	10	14	18	22	1	5	9	13	17	21	25	7	11	15	19	23
	1	**CHA ID:**	0	4	8	12	20	24	2	6	10	14	18	22	1	5	9	13	17	21	25	7	11	15	19	23



(a) Pattern #1 (b) Pattern #2 (c) Pattern #3

Fig. 4: The three most frequently observed core location mappings on the Xeon Platinum 8259CL CPUs. The IDs in the tiles indicate the following ID pairs: OS core ID / CHA ID.

TABLE II: Observed core location pattern statistics.

CPU Model		8124M	8175M	8259CL
Top-4 frequently observed location patterns	Pattern #1	53 insts.	52 insts.	19 insts.
	Pattern #2	18 insts.	7 insts.	5 insts.
	Pattern #3	5 insts.	7 insts.	4 insts.
	Pattern #4	5 insts.	6 insts.	4 insts.
Total number of unique patterns		14 patterns	26 patterns	53 patterns

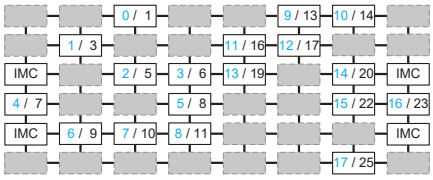


Fig. 5: Core location mapping example of a third-generation Xeon 6354 CPU.

temperature sensors can reduce the channel capacity. However, even if the internal channel is blocked, our mechanism can help to create a stronger *external* thermal covert channel. An attacker who has physical access to the hardware can externally probe the temperature of the desired core tiles on the CPU die [8].

A. Encoding and Decoding Scheme

We use the `stress-ng` program to control the sender node's temperature. Among the stress tests in `stress-ng`, we found the repeated branch misses cause the most heat. We use the *Manchester encoding* suggested in [7] to minimize the thermal bias caused by a monotonic bit pattern. The measured temperature traces at the receiver node is decoded offline. The decoder is synchronized to the sender node phase using a designated signature bit sequence. The decoder determines the offset in the measurement that can correctly decode the signature bit sequence and decode the actual payload.

Figure 6 shows an example of the data transmission through the thermal channel between the cores. The propagated heat

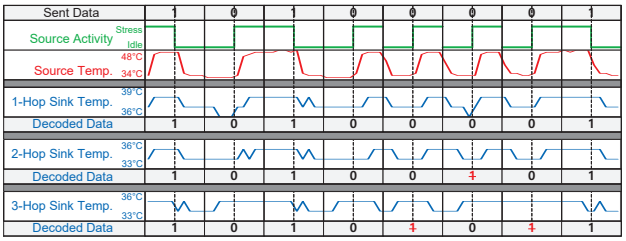


Fig. 6: Inter-core thermal covert channel measurements.

is measured by the receiver nodes that are located 1, 2, and 3 tile hops away from the sender in the vertical direction. The measurement from the 1-hop receiver observes dampened fluctuations, but the transmitted data can be correctly decoded with a high probability. If the receiver is placed on a core further away than one tile hop, the thermal measurements start to show unstable behavior and result in higher bit error rates.

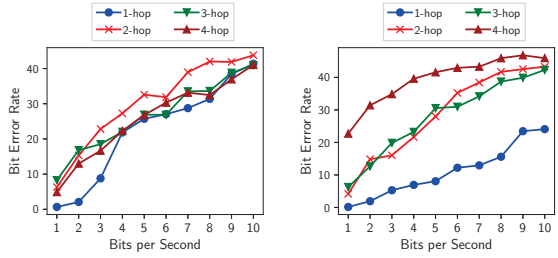
V. Thermal Covert Channel Measurements

We evaluate the thermal covert channel using the 8259CL CPUs. We measured 10 Kbits of random bitstream transfer for each of the measurements. The reported error probability does not employ any additional error correction scheme.

A. Effects of Core Tile Hop Counts

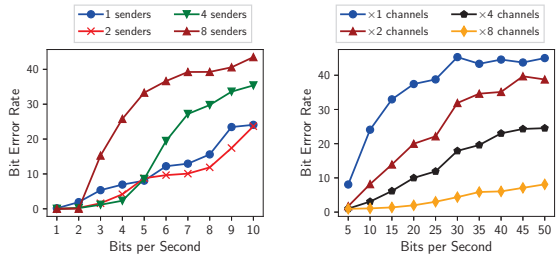
Figure 7 compares the resulting bit error rate during the transmission on the varying distances between the sender and receiver nodes. If the sender and the receiver are not directly located to next each other, the resulting bit error rate is too high to be utilized as a reliable data communication channel. For 1-hop sender-receiver pairs, 1 bps of transfer rate can be reliably achieved with almost zero bit error rate.

For the same 1-hop distance, the sender-receiver pair located next to each other in the vertical direction creates a better



(a) Horizontal hops (b) Vertical hops

Fig. 7: Bit transfer rate and the resulting bit error probability for different sender-receiver hop counts.



(a) Multiple sender nodes (b) Multiple channels

Fig. 8: Bit transfer rate and the resulting bit error probability of the strengthened thermal covert channels.

communication channel than the sender-receiver pair located horizontally. In Fig. 7a, the horizontal 1-hop channel's error rate worsens to more than 20% at 4 bps, but the error rate at the same transfer rate is less than 10% on the vertical 1-hop channel in Fig. 7b. A Xeon core tile is a horizontally long rectangle [4], and the shorter distance between the vertical neighbors is better for inter-tile heat propagation.

B. Multiple Senders

We can use multiple sender nodes to amplify the thermal activity. We may use up to *eight* sender nodes that surround the receiver node. Those sender nodes are synchronized and send the same heat activity to the receiver node. The strengthened thermal signal reduces the bit error rate. In Fig. 8a, the bit error rate at 4 bps is lowered to 2% when four sender nodes are used.

C. Multiple Covert Channels

On a many-core CPU, we can create multiple sender-receiver pairs to increase the overall throughput. Figure 8b shows the achieved *aggregated* throughput from the multi-channel setting. For instance, the 40 bps throughput from the ×8 setting is achieved by eight 5 bps channels. Using the multi-channel transmission, the inter-core thermal covert channel achieved up to 15 bps of aggregated transmission rate with less than 1% of error rate by the ×8 setting. This is three times higher than the previously reported 1-hop channel capacity of 5 bps [7]. Please note that the experiment in [7] was conducted in a controlled environment, unlike our cloud-based experiments.

D. Core Location Mapping Verification

To confirm that our core map reveals the true core locations, we conduct thermal transmission between all core pairs. As expected, the lowest error rates are achieved between the neighboring cores identified by our mechanism except for a few cases. Those exceptions are the core tiles that have no adjacent vertical neighbor (*e.g.*, CHA ID 1 in Fig. 4a).

VI. Related Work

Independent of our work, a recent report by McCalpin discuss a different mechanism that identifies the core locations of the Xeon processors [9]. Instead of creating a fully autonomous algorithm for core location mapping, McCalpin's approach is generalizing the observed core map patterns of the tested CPU instances. This approach is not directly applicable to different CPU models that use a different mapping pattern, such as the latest third-generation Xeon CPUs. In contrast, our mechanism works for the newer Xeon cores as well (Sec. III-B).

Horro et al. found the core mapping pattern of the Xeon Phi KNL processors based on the memory access latency [10]. The latency-based mechanism is not sufficient for the Xeon CPUs with only two DRAM memory controllers.

VII. Conclusion

Our mechanism provides an automated way to identify the physical locations of processor core tiles connected through a mesh interconnect. The mechanism is applicable to the current generation Intel Xeon processors including the latest models.

We demonstrated that the identified core locations can be a potential security threat by demonstrating an inter-core thermal covert channel. The observed thermal covert channels also indirectly verify the correctness of our mapping results.

The source code of our mapping mechanism is available at `https://github.com/hyungmin2/core_map` .

References

[1] Arm® corelink™ cmn-600 coherent mesh network technical reference manual. [Online]. Available: https://developer.arm.com/documentation/100180/0201

[2] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical," in *USENIX Security*, 2021.

[3] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *USENIX Security*, 2015.

[4] S. M. Tam, H. Muljono, M. Huang, S. Iyer, K. Royneogi, N. Satti, R. Qureshi, W. Chen, T. Wang, H. Hsieh, S. Vora, and E. Wang, "SkyLake-SP: A 14nm 28-core xeon® processor," in *ISSCC*, 2018.

[5] "Intel® Xeon® processor scalable memory family uncore performance monitoring," 2017.

[6] M. Yan, R. Sprabery, B. Gopireddy, C. W. Fletcher, R. H. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *S&P*, 2019.

[7] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores," in *EuroSys*, 2016.

[8] A. Smith. (2016) Measuring small objects with infrared pyrometers. [Online]. Available: https://www.calex.co.uk/measuring-small-objects-with-infrared-pyrometers

[9] J. D. McCalpin, "Mapping core and l3 slice numbering to die location in intel xeon scalable processors," in *ACELab Technical Report TR-2021-01b*, 2021.

[10] M. Horro, M. T. Kandemir, L.-N. Pouchet, G. Rodríguez, and J. Touriño, "Effect of distributed directories in mesh interconnects," in *DAC*, 2019.