2013 IEEE 7th International Symposium on Embedded Multicore/Manycore System-on-Chip

Mapping Non-trivial Network Topologies onto Chips

Ikki Fujiwara* and Michihiro Koibuchi*†
*National Institute of Informatics
†The Graduate University for Advanced Studies (SOKENDAI)
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, JAPAN 101-8430
Email: {ikki, koibuchi}@nii.ac.jp

***Abstract*—Adopting high-degree topologies is a promising way to reduce end-to-end latency in a network-on-chip (NoC). However, some high-degree topologies are not used in practice due to their complex layout on a chip. In this work we explore the way to systematically obtain the quasi-optimal mapping of those topologies onto a chip by modelizing the mapping problem as a quadratic assignment problem. Results show that the Robust Tabu Search algorithm achieves the mappings with the shortest link length for most topologies of up to 512 cores. The link length is reduced by up to 51% when compared to naive baseline method. We also tackle even larger topologies by means of clustering, and a promising results are obtained to apply those algorithms for topologies of 2,048 or more cores with a modest penalty on the link length.**

Keywords*—*Network topologies, systems on chip, Network-on-Chip, optimization.

I. Introduction

The advances in semiconductor technology allow us to integrate a number of processing cores on a single chip using enormous amount of wire resources [1]–[4]. It is expected to reach 1,000 cores on a chip [5], [6]. Enormous wire resources enable to use a large number of links on a chip, e.g. 32 or 256 wires per link. Thus, high-radix on-chip network topologies that effectively use a large number of links are widely proposed to make a low-latency network-on-chips (NoCs) [7]. Since traditional on-chip network topologies often exhibit both highly regular structure and low degree (e.g. k-ary 2-cubes), they naturally fit into a simple physical layout. In addition, recent specific high-radix topologies such as Flattened Butterfly [7] have their own layout that has a short link length. Other typical high-degree topologies such as k-ary n-cube with $n > 3$ are also attractive to attempt the task mapping for traditional parallel algorithms especially in chip multiprocessors. Many parallel algorithms have known and efficient mapping to such high-degree topologies. With those topologies, optimal physical layouts are no longer intuitive and system designers are now facing a difficult task of mapping routers to a physical layout so as to reduce the total link length on a chip.

In this context, given a topology, we study the optimization of the physical layout on a 2-D chip in a view to minimizing the total link length. Our approach maps on-chip routers onto a physical die by framing the problem and solving it as a quadratic assignment problem (QAP). Our goal in this work is to reduce the total link length not only for low-radix, layout-friendly topologies but also for more challenging high-radix topologies. Our main contributions and findings are as follows: (1) a quasi-optimal mapping of high-degree network topologies onto a chip can practically be obtained for a chip with up to 512 cores; (2) Robust Tabu Search algorithm achieves the best tradeoff between the quality of the solution and the execution time; and (3) a clustering technique enables our approach to deal with even larger topologies.

The rest of this paper is organized as follows. Section II rationalizes the mapping problem and introduces algorithms to solve it. Section III evaluates the performance of those algorithms. Section IV extends our approach to larger problems. Section V discusses related work. Finally, Section VI concludes the paper with a summary of our findings.

II. Quadratic Assignment Problem

A mapping of cores onto a chip can be modelized and optimized as a quadratic assignment problem (QAP). In this section we introduce a formal representation of QAP as well as applied optimization algorithms and their implementations.

A. Rationale

Assume a topology of N cores to be mapped onto N locations on a chip so that the total link length between cores is minimized. The QAP solution is represented by a permutation $\Phi = \phi(1), \ldots, \phi(N)$ such that

$$\text{Minimize} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} d_{\phi(i)\phi(j)} \quad (1)$$

where d_{ij} denotes the physical distance between locations i and j, w_{ij} denotes the number of links between cores i and j, and $\phi(i)$ denotes the location where core i is assigned.

QAP is considered as one of the "hardest of the hard" of all combinatorial optimization problems [8] as it is NP-hard and even an ϵ-approximation algorithm is proven not to exist unless P = NP [9]. In general, QAP instances of size $N > 30$ cannot be exactly solved in reasonable time [10]. Therefore, we need to use heuristic algorithms to practically solve a QAP. As we have no hope for exact optimization, we use the term "optimization" in the same meaning as "quasi-optimization" in the rest of this work.

B. Optimization Algorithms and Solvers

We select for this work three metaheuristic algorithms that have been successfully applied to QAP.

- Simulated annealing (SA) is a local search algorithm that exploits the analogy between combinatorial optimization algorithms and statistical mechanics [10]. We adopted

978-0-7695-5086-2/13 $26.00 © 2013 IEEE
DOI 10.1109/MCSoC.2013.10

TABLE I. DIMENSIONS.

(a) Physical dimensions

#cores	#tiles
32	6×6
64	8×8
128	12×11
256	16×16
384	20×20
512	23×23
640	26×25
768	28×28
896	30×30
1024	32×32

(b) Topological dimensions

#cores	4-D Torus	5-D Torus	6-D Torus	7-D Torus	Hypercube	Random Ring [degree]
32	2×2×2×4	2×2×2×2×2	—	—	2×2×2×2×2	5
64	2×2×4×4	2×2×2×2×4	2×2×2×2×2×2	—	2×2×2×2×2×2	6
128	2×4×4×4	2×2×2×4×4	2×2×2×2×2×4	2×2×2×2×2×2×2	2×2×2×2×2×2×2	7
256	4×4×4×4	2×2×4×4×4	2×2×2×2×4×4	2×2×2×2×2×2×4	2×2×2×2×2×2×2×2	8
384	4×4×4×6	2×3×4×4×4	2×2×2×3×4×4	2×2×2×2×2×3×4	—	9
512	4×4×4×8	2×4×4×4×4	2×2×2×4×4×4	2×2×2×2×2×4×4	2×2×2×2×2×2×2×2×2	9
640	4×4×5×8	2×4×4×4×5	2×2×2×4×4×5	2×2×2×2×2×4×5	—	10
768	4×4×6×8	2×4×4×4×6	2×2×2×4×4×6	2×2×2×2×2×4×6	—	10
896	4×4×7×8	2×4×4×4×7	2×2×2×4×4×7	2×2×2×2×2×4×7	—	10
1024	4×4×8×8	4×4×4×4×4	2×2×4×4×4×4	2×2×2×2×4×4×4	2×2×2×2×2×2×2×2×2×2	10

Taillard's implementation in C [11] of Connolly's algorithm [12].

- Tabu search (TS) is a local search algorithm that features an updated list of the best solutions that were found in the search process [10]. We adopted Taillard's implementation in C++ [11] of his Robust Tabu Search algorithm [13].
- Greedy randomized adaptive search procedure (GRASP) [14] is a random and iterative technique that obtains an approximate solution at each step and returns the best generated one [10]. We adopted Recende's implementation in Fortran [15] of GRASP for sparse QAP [16].

We defined the following baseline method for comparison. Assuming the locations are aligned on a 2-D grid, we consider two schemes: (1) a "wrap-around" method, which assigns the locations from left to right in every row, and (2) a "zigzag" method, which assigns the locations from left to right in the first row, from right to left in the second row, etc. The former can produce reasonable results for mesh/torus topologies whereas the latter is more reasonable for ring-based topologies. Our baseline method uses both schemes and returns the best of the two obtained results.

III. COMPARISON OF OPTIMIZATION ALGORITHMS

Forthcoming technology will integrate several hundreds of cores on a chip. No exact optimization algorithm is likely to be available for such a large QAP and different heuristics bring different tradeoffs between the quality of the solution (i.e. the resulting link length) and the execution time. In this section we evaluate those metaheuristic algorithms introduced in the previous section and try to pick the most preferable one for our purpose.

A. Setup

We employ an abstract model of an interconnection network on a 2-D chip as described below. A chip consists of many tiles. A tile hosts a core. A core has its own router (the term "core" includes its router in this work) and is connected to other cores with physical links. All tiles have a uniform-sized square shape (e.g. 1 mm × 1 mm) and are aligned in a grid-like pattern on a chip (e.g. 64 tiles are aligned on a chip of 8 mm × 8 mm). We also use the term "tile" as a unit of length, which is equal to the grid size (e.g. 1 *tile* = 1 mm if a tile is 1 mm × 1 mm large). We assume the number of cores to be $N \in \{32, 64, 128, 256, 384, 512, 640, 768, 896, 1024\}$ and the number of tiles to be $x \times y$, where $x = \lceil \sqrt{N} \rceil$ and $y = \lceil N/x \rceil$, as shown in Table I(a). In case of $N < xy$, cores are located on N tiles selected from left to right and then from top to bottom of the chip, and the remainder $xy - N$ tiles are left unused. We pick several typical topologies to be mapped onto a chip, namely n-dimensional Torus with $n \in \{4, 5, 6, 7\}$, n-dimensional Hypercube with $n \in \{5, 6, 7, 8, 9, 10\}$ and d-degree Random Ring with $d \in \{5, 6, 7, 8, 9, 10\}$, as shown in Table I(b). A Random Ring is a ring topology with $d - 2$ additional shortcut links at every core which randomly connect to other cores [17].

We executed those solvers introduced in Section II-B as single-threaded programs on our Linux server with 3.47 GHz Intel Xeon X5690 processor and 144 GB main memory. We run SA for 100 million iterations and pick the best solution out of 10 trials.

B. Link Length vs. Chip Size

Figures 1 and 2 show the average and the maximum link length, respectively, vs. the number of cores. Results indicate that both SA and TS successfully reduce the link length, and TS slightly outperforms SA at most cases. For example, the average link length of 4-D Torus topology of 640 cores is reduced by 49.6% and 51.3% by SA and TS, respectively, when compared to the baseline. The difference between SA and TS is less than 10% of the best solution as far as the average link length is concerned. GRASP also reduces the link length, but its effectiveness is not as much as SA and TS. Some plots of GRASP are missed due to runtime errors. The baseline method exceptionally works well for Torus/Hypercube of 64, 256 and 1,024 cores (see also Fig. 6). They are "lucky" cases in which their topological dimensions fit the physical dimensions. In other general cases, the baseline method leads to unreasonably long link lengths, and proves the necessity of a systematic method to optimally map a complex topology onto a chip.

Cumulative distributions of the link length for some particular topologies are shown in Fig. 3. The baseline has some "steps" or intermediate horizontal parts on its plots. This indicates that the topological boundary matches up with the physical boundary. The steps can clearly be seen in the "lucky" cases including Torus/Hypercube of 256 cores (as shown in the two charts on the left-hand side) and not in general cases

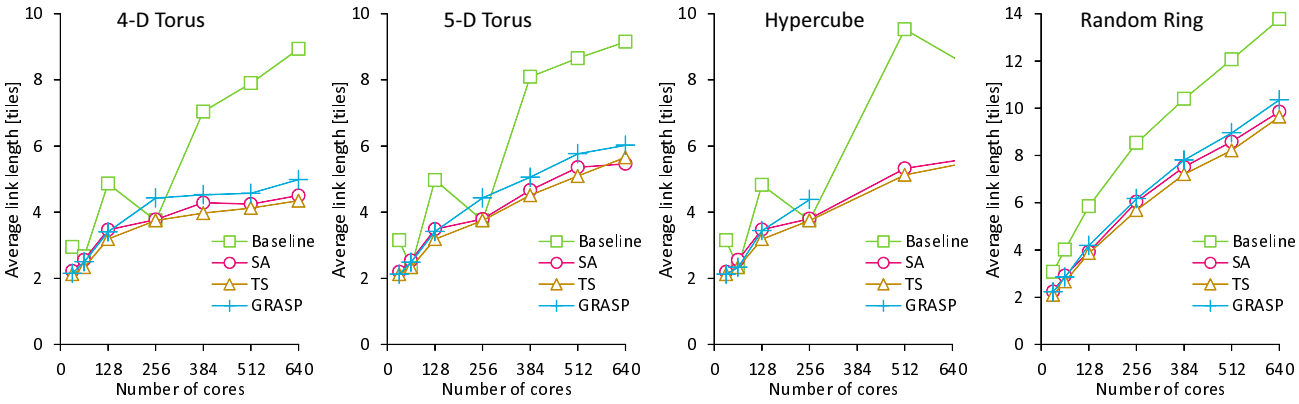

Fig. 1. Average link length vs. number of cores.

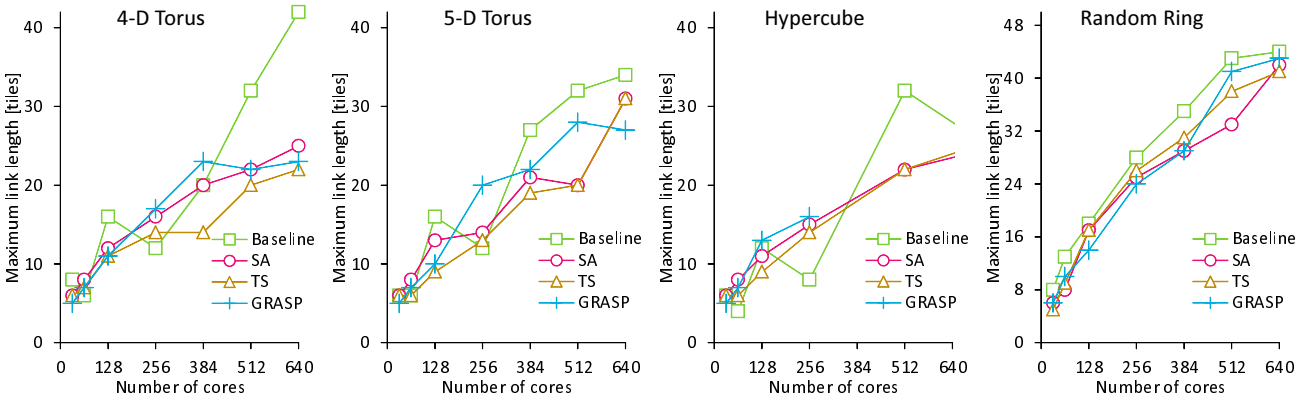

Fig. 2. Maximum link length vs. number of cores.

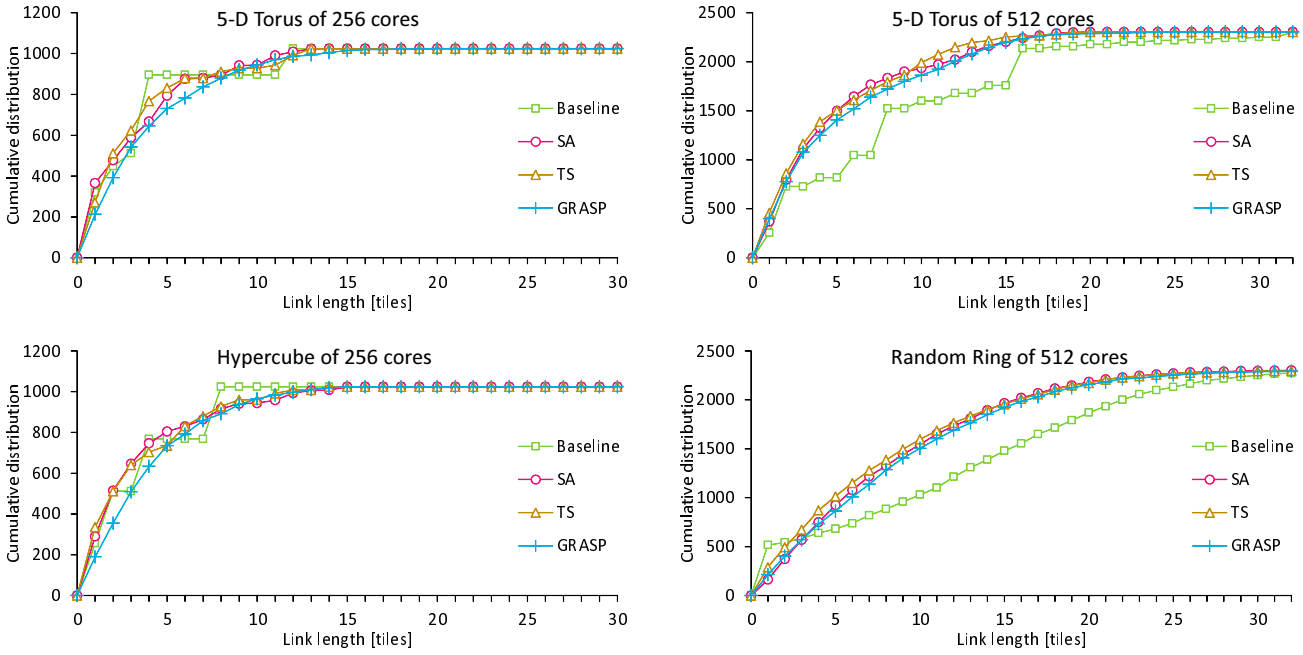

Fig. 3. Cumulative distribution of link length.

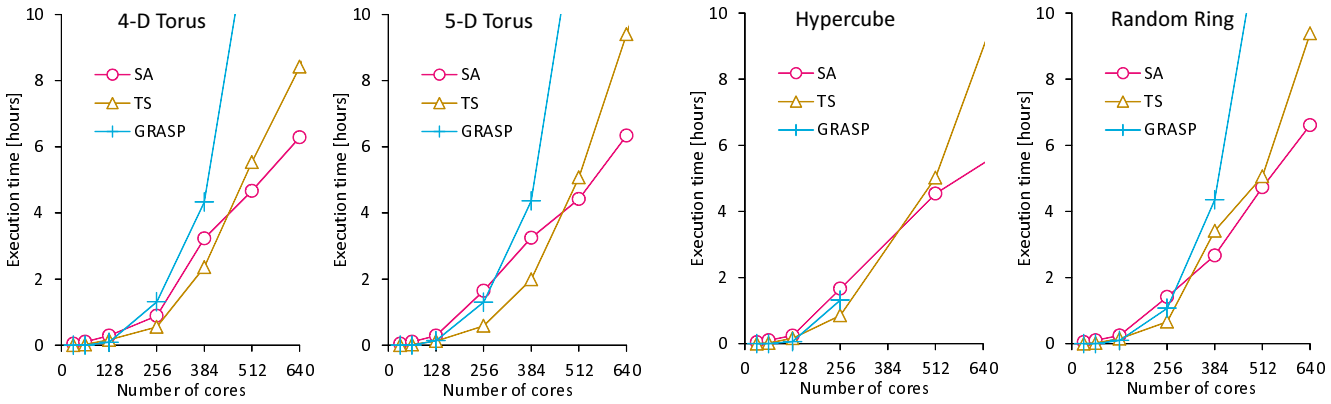

Fig. 4. Execution time of solvers vs. number of cores.

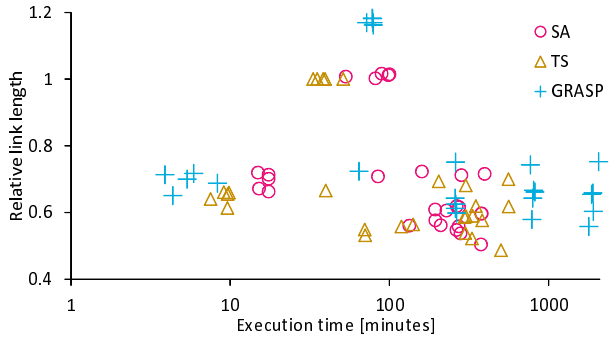

Fig. 5. Link length relative to baseline vs. execution time of solvers. Horizontal axis is logarithmic.

(on the right-hand side). The plots of SA, TS and GRASP are pulled up toward the upper left corner and have no clear steps. This means that those algorithms successfully shorten the link lengths without depending on the relationship between physical and topological dimensions.

C. Execution Time vs. Chip Size

Figure 4 shows the execution time of the solvers vs. the number of cores. The execution time of SA is the longest among those three algorithms with a small chip; while it becomes the shortest with a large chip. This is because SA repeats a fixed number of trials (10 times in this work) to search the optimal solution and each trial has a pre-defined cap on the number of iterations (100M iterations in this work). Indeed the cap eliminates the execution time while possibly misses a better solution, but the opportunity loss caused by the cap is not very significant, as far as the resulting link length is concerned (see Section III-B). The execution time of TS, in contrast, straightly fits a quadratic function to the chip size. It is the shortest among those three algorithms when $128 < N < 512$ (except for Random Ring) and the difference between TS and GRASP is less than 10 minutes when $N \leq 128$. The crossover point of SA and TS exists at around $384 < N < 512$ (except for Random Ring) and the difference between SA and TS is less than 16% (70 minutes) at $N = 512$. Consequently we consider TS to be advantageous or comparable to SA as long as $N \leq 512$. The execution time of GRASP grows near-exponentially with the chip size and exceeds 12 hours at $N = 512$. We thus consider GRASP as an impractical algorithm to optimize the mapping of topology of more than 256 cores.

D. Link Length vs. Execution Time

To summarize the observations in this section, we plot the link length (relative to baseline) vs. execution time of the solvers in Fig. 5. From the chart we can reconfirm that SA and TS are comparable in terms of the resulting link length. TS tends to consume shorter execution time than SA when the problem size is small (on the left-hand side); while TS can cause a longer execution time than SA when the problem size is large (on the right-hand side). The plots of GRASP tend to form a vertical distribution on the horizontal axis, which means that the execution time of GRASP depends not on the essential difficulty of the problem but sorely on the problem size. When there are 640 cores (the rightmost part of the plots), GRASP can cause more than seven times longer execution time than SA.

From the overall observations in this section, we conclude that TS is the most preferable algorithm to optimize the mapping of a topology onto a chip, as long as the chip size does not exceed 512 cores.

IV. Further Scaling by Clustering

We found that naive implementations of the optimization algorithms face a practical upper limit on the chip size at around 512 cores. Indeed more sophisticated solvers are proposed to tackle larger problems, but QAP is essentially too difficult to solve such a large problem directly. Instead we should go through another way to reduce the complexity of the problem, and divide-and-conquer is a promising strategy to do so. In this section we try to divide the problem by grouping several cores into a cluster and to optimize the mapping on a per-cluster basis.

A. Setup

Given a chip, on one hand, we divide it so that the adjacent four tiles (2×2 on a chip) form a clustered tile. We assume that

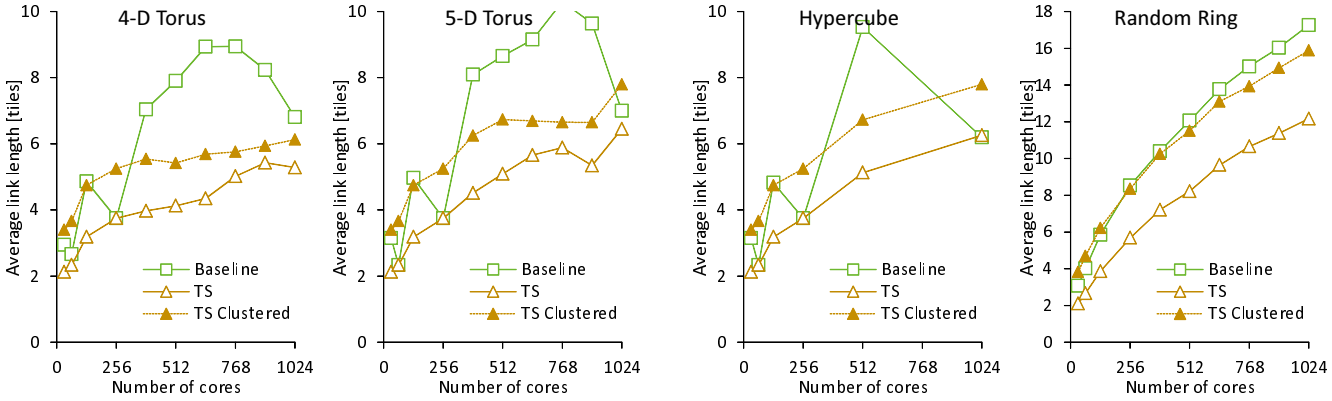

Fig. 6. Average link length vs. number of cores. Clustered vs. non-clustered.

the size of a cluster is 2 tiles $\times$ 2 tiles, the link length between two cores within the same cluster is 1 tile, and the link length between two cores on different clusters has an overhead of 2 tiles (1 tile at each end) in addition to the Manhattan distance between two clusters. Given a topology of cores, on the other hand, we pick every four cores in the canonical topological order to form a clustered core. For example, given a 5-D Torus of 128 cores ($2\times2\times2\times4\times4$ cores), the first 2×2 cores form the first cluster, the next 2×2 cores form the second cluster, and so on. As a result, in this example, the first two dimensions remain within each cluster and the last three dimensions exit the clusters. We then optimize the mapping of the clustered cores onto the clustered tiles using the same methods as the previous section.

B. Link Length vs. Chip Size

Figure 6 shows the average link length vs. the number of cores. The number of clusters is one fourth of the number of cores. In this section we only show the results of TS, since the results of the other optimization algorithms follow essentially the same trend.

From the result we can observe that the link length increases by clustering, and the differences between the clustered and the non-clustered cases are almost constant. The average penalties on the link length is 1.41 tiles in 4-D Torus, 1.44 tiles in 5-D Torus and 1.46 tiles in Hypercube, which are smaller than the overhead of the inter-cluster links (2 tiles long), because cores are densely connected inside a cluster using shorter intra-cluster links (1 tile long). An exception to this observation is Random Ring, in which the average penalty is 2.66 tiles, because many shortcut links jumps beyond a cluster.

C. Link Length vs. Execution Time

Figure 7 shows the optimized link length (relative to non-clustered baseline) vs. the execution time of the TS solver. A line corresponds to a topology. The leftmost and the rightmost points of a line correspond to the clustered and the non-clustered cases, respectively.

From the result we can observe that the execution time decreases drastically by clustering (note that the horizontal axis is logarithmic), whereas the link length increases modestly. This observation applies to all the cases with no exception. For example, in the case of 640-core 5-D Torus, the execution time decreases by 98% (from 33,875 sec to 728 sec) whereas the link length increases by 18% (from 16,296 tiles to 19,280 tiles).

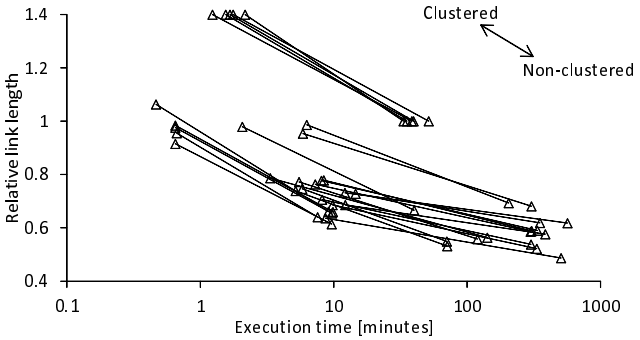

Fig. 7. Link length relative to non-clustered baseline vs. execution time of TS solver. Leftmost point is clustered and rightmost point is non-clustered. Horizontal axis is logarithmic.

Since clustering four cores reduces to one fourth the size of QAP, we confidently suppose that the mapping of a 2,048-core topology can be optimized in essentially the same execution time as a 512-core non-clustered counterpart. Further scaling can also be considered by clustering 9 cores (3×3), 16 cores (4×4) etc., at a possible cost of wiring overhead inside a cluster.

V. Related Work

We focus on existing on-chip topologies and their layouts in NoCs. Needless to say, k-ary 2-meshes and folded k-ary 2-tori have intuitive layouts that make each link length uniform and short. The butterfly network (k-ary n-fly) can be efficiently mapped onto a 2-D VLSI by utilizing high-radix routers [18]. In the flattened butterfly [7], routers in each row of a conventional butterfly are combined into a single router. It has a large diversity of router degrees for each network size and its low-degree case is equivalent to hypercube. Spidergon topology, which is a ring topology with links that connect

diagonal counterparts in the ring, has been discussed for cost-effective on-chip networks [19]. It can be efficiently mapped onto a chip in which almost all links have minimum length as well as k-ary 2-meshes. However, its average hop counts considerably increase as the number of nodes increases, even though it provides diagonal links to mitigate the increase of diameter compared to a conventional ring. To the best of our knowledge, the above specific topologies have good layouts on a chip; however, optimal physical layouts are no longer intuitive for other typical high-radix networks evaluated in the previous sections.

In addition to the above direct networks in which each router has a local core, NoCs sometimes employ indirect networks in which some routers do not have local cores. The simplest indirect network topology is H-Tree, in which each router (except for the top-rank router) has one upward and four downward connections. The links and routers around the root of the H-Tree are frequently congested due to its poor bisection bandwidth. Fat Tree enhances the number of connections toward the root to mitigate the congestion. Another variation of H-tree is Fat H-tree that includes tori and tree by using two H-trees [20]. Our work can be applied to any indirect networks for their layout; however, their evaluation is out of our scope in this work.

Custom topology design for a traffic patterns generated by a target application has been widely discussed [21], [22]. Their resulting topology usually becomes irregular. Our works can be naturally extended to such irregular topologies for their layout; however, their evaluation is also out of our scope in this work.

VI. Conclusions

In this work we tried to obtain a quasi-optimal mapping of high-degree network topologies onto a chip for upcoming low-latency network-on-chips (NoCs). We modelized the mapping as a quadratic assignment problem in a straightforward way and compared three metaheuristic algorithms to solve it. The Robust Tabu Search algorithm among them achieves the best tradeoff between the quality of the solution and the execution time as long as the chip size does not exceed 512 cores. For even larger chips we modelized the mapping in a sophisticated way that clusters adjacent cores to reduce the complexity, and confirmed the feasibility of our approach for large-scale NoCs with 2,048 or more cores with a modest penalty on the link length. The advantage of our approach is that it makes it possible to decouple layout concerns from topology concerns, while remaining effective even for high-degree topologies, which are not amenable to intuitive physical layouts.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Numbers 25280018 and 25730068, and by NII Joint Research Fund.

References

[1] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of the Design Automation Conference (DAC'01)*, Jun. 2001, pp. 684–689.

[2] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, John F. Brown III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.

[3] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, vol. 27, no. 5, pp. 41–50, Sep. 2007.

[4] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep. 2007.

[5] T. M. Pinkston and J. Shin, "Trends Toward On-Chip Networked Microsystems," *International Journal of High Performance Computing and Networking*, vol. 3, no. 1, pp. 3–18, Sep. 2005.

[6] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge, "Scaling Towards Kilo-Core Processors with Asymmetric High Radix Topologies," in *Proc. of the International Symposium on High Performance Computer Architecture(HPCA)*, 2013.

[7] J. Kim, J. Balfour, and W. J. Dally, "Flattened Butterfly Topology for On-Chip Networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO'07)*, Dec. 2007, pp. 172–182.

[8] M. Bayat and M. Sedghi, "Quadratic Assignment Problem," in *Facility Location*, R. Z. Farahani and M. Hekmatfar, Eds. Physica-Verlag, 2009, ch. 6, pp. 111–143.

[9] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," *Journal of the ACM*, vol. 23, no. 3, pp. 555–565, Jul. 1976.

[10] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European Journal of Operational Research*, vol. 176, no. 2, pp. 657–690, Jan. 2007.

[11] [Online]. Available: http://mistic.heig-vd.ch/taillard/

[12] D. T. Connolly, "An improved annealing scheme for the QAP," *European Journal of Operational Research*, vol. 46, no. 1, pp. 93–100, May 1990.

[13] E. D. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Computing*, vol. 17, no. 4-5, pp. 443–455, Jul. 1991.

[14] Y. Li, P. M. Pardalos, and M. G. Resende, "A greedy randomized adaptive search procedure for the quadratic assignment problem," in *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, P. M. Pardalos and H. Wolkowicz, Eds. AMS, 1994, vol. 16, pp. 237–261.

[15] [Online]. Available: http://www2.research.att.com/~mgcr/

[16] P. M. Pardalos, L. S. Pitsoulis, and M. G. C. Resende, "Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP," *ACM Transactions on Mathematical Software*, vol. 23, no. 2, pp. 196–208, Jun. 1997.

[17] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu, and H. Casanova, "A Case for Random Shortcut Topologies for HPC Interconnects," in *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2012, pp. 177–188.

[18] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a High-radix Router," in *Proceedings of the International Symposium on Computer Architecture (ISCA'05)*, Jun. 2005, pp. 420–431.

[19] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *Proceedings of the International Symposium on System-on-Chip (ISSOC'04)*, Nov. 2004, p. 15.

[20] H. Matsutani, M. Koibuchi, and H. Amano, "Performance, Cost, and Energy Evaluation of Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'07)*, Mar. 2007.

[21] W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'03)*, Feb. 2003, pp. 377–388.

[22] J. Hu and R. Marculescu, "Communication and Task Scheduling of Application-Specific Networks-on-Chip," in *IEE Proceedings Computers and Digital Techniques*, Sep. 2005.