Application Mapping for Express Channel-Based Networks-on-Chip

Di Zhu, Lizhong Chen, Siyu Yue, and Massoud Pedram
University of Southern California
Los Angeles, California, USA 90089
{dizhu, lizhongc, siyuyue, pedram}@usc.edu

***Abstract*—With the emergence of many-core multiprocessor system-on-chips (MPSoCs), the on-chip networks are facing serious challenges in providing fast communication for various tasks and cores. One promising solution shown in recent studies is to add express channels to the network as shortcuts to bypass intermediate routers, thereby reducing packet latency. However, this approach also greatly changes the packet delay estimation and traffic behaviors of the network, both of which have not yet been exploited in existing mapping algorithms. In this paper, we explore the opportunities in optimizing application mapping for express channel-based on-chip networks. Specifically, we derive a new delay model for this type of networks, identify their unique characteristics, and propose an efficient heuristic mapping algorithm that increases the bypassing opportunities by reducing unnecessary turns that would otherwise impose the entire router pipeline delay to packets. Simulation results show that the proposed algorithm can achieve a 2~4X reduction in the number of turns and 10~26% reduction in the average packet delay.**

Keywords—network-on-chip; application mapping; express channels

I. INTRODUCTION

With the integration of tens to possibly a hundred of cores on a chip [8][18], multiprocessor system-on-chips (MPSoCs) have been provided with tremendous opportunities for parallel execution. A key challenge of the parallel paradigm is the design of high performance on-chip network (a.k.a. OCN or NoC) that can connect various IP blocks or tasks running on different cores. However, as the network sizes continue to grow, traditional NoC topologies such as mesh or concentrated mesh [1] have been facing serious performance issues due to their inherent nature of hop-by-hop packet forwarding.

A more scalable approach that has been paid increasing attention is to add express channels [7][12][14] to the tile-based NoCs. These express channels act as shortcuts between non-neighboring tiles to bypass all intermediate routers, thereby accelerating packet transfer. Nevertheless, the addition of express channels significantly changes the traffic patterns and requires different delay calculation models between tiles. For example, packets on express channels cannot make turns; so packets need to get off the express channels and go through the entire router pipeline stages in order to make a turn, which slows down the packet transport. These and other new characteristics exhibited in express channel-based networks are not captured and exploited in existing application mapping algorithms that are responsible for mapping tasks to physical tiles.

In this paper, we investigate the opportunity of optimizing application mapping for express channel-based networks. Specially, we identify the critical differences between traditional networks and express channel-based networks, derive a new delay model reflecting express channels, mathematically formulated the corresponding application mapping problem, and proposed an efficient heuristic mapping algorithm based on the key observations of the problem characteristics. The proposed algorithm, *Turn Reduction Algorithm for Mapping (TRAM)*, is able to not only effectively map tasks with large communication rate closer to each other as what have been achieved in previous algorithms, but also maximize the alignment of heavily communicating tasks in both rows and columns, thus reducing unnecessary turns that would otherwise impose the long delay of router pipeline to packets.

The rest of the paper is organized as follows. Section II provides more background on express channel-based on-chip networks and motivates the need for new mapping algorithms. Section III formulates the problem, and Section IV explains the details of the proposed TRAM algorithm. Section V and VI describe evaluation methodology and present simulation results. Finally, Section VII concludes the paper.

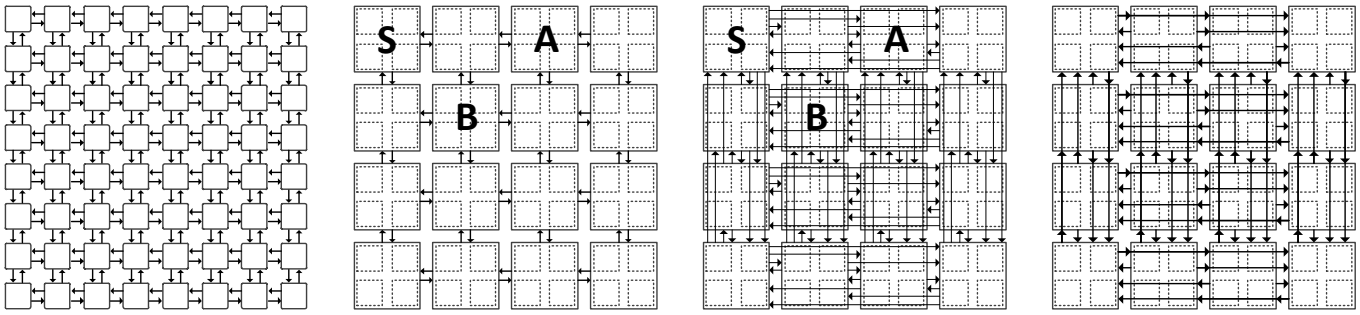
II. BACKGROUND AND MOTIVATION

A. Express channel-based On-chip Networks

While mesh topology has traditionally been used for tile-based NoCs, packets in mesh networks must be forwarded hop-by-hop, which exposes the router delay (e.g., 3~4 cycles) and link delay (e.g., 1 cycle) at *every* hop to the packet latency. To mitigate the latency problem of mesh, particularly for large networks, concentration [1] (Figure 1b CMesh) has been proposed in which multiple IP blocks or tasks are placed on the same tile to form a task cluster. All tasks in a task cluster occupy one tile and share one router. With a concentration degree of 4, the network diameter can be reduced by half. However, due to the layout constraints and the increased router complexity, it is difficult to employ high concentration degrees, thus limiting the latency reduction through this technique.

As more research being conducted to improve NoC performance, recent studies show promise of adding express channels on top of concentration to accelerate packet transfer [7][12][14]. Figure 1(c) shows an example of the popular flattened butterfly (FB) topology [12] that adds separate links to connect two non-neighbor tiles directly (e.g., from top-left tile to top-right tile). To better utilize the link resources, a network with multi-drop express channels (Figure 1d MECS) [7] is proposed to combine separate links to a unified link but with multiple "drops", so that no additional input or output ports are needed. Packets are routed on the express channels as much as possible and use

This work is supported in part by the Software and Hardware Foundations program of the NSF's Directorate for Computer & Information Science & Engineering.



(a) Mesh (b) CMesh (c) Flattened Butterfly (d) MECS

Figure 1. On-chip networks without express channels: (a) and (b), and with express channels: (c) and (d).

non-express channels only if contention occurs. In this way, intermediate routers on the same row or column can be by-passed, resulting in only the link latency.

However, in order to change dimension, packets need to get off the express channels and enter the normal router/switch pipeline to make the turns. Also, dimension-order routing is typically used in FB and MECS instead of adaptive routing [7]. This is because adaptive routing may generate a large number of turns, causing most packets to go through normal routers, which defeats the purpose of adding express channels.

B. Related Work

Application mapping is an important component in the design of multiprocessor systems. MPSoC applications such as video encoder/decoder typically consist of many tasks that are working collaboratively to perform certain functions. By mapping frequently or heavily communicating tasks to physically close tiles, the average packet delay and power consumption can be greatly reduced. Due to the importance of application mapping, a number of mapping algorithms have been proposed. For example, Hu *et al.* in [9] use graphs to model the characteristic of applications and propose a branch-and-bound algorithm to minimize communication energy of mapping. A two-step genetic algorithm is proposed in [15] to map applications on mesh-based NoCs to optimize task graph execution. Murali *et al.* focus on minimizing communication delay under bandwidth constraints in [16]. Chen *et al.* present mechanisms for joint optimization by task scheduling, application mapping, data mapping and routing on NoC-based CMPs [2]. Faruque *et al.* use a distributed approach based on agents for application mapping and greatly lowered the monitoring traffic and computational effort compared to centralized schemes [5]. In [10], Jang *et al.* form the mapping of heterogeneous cores on irregular mesh-based MPSoCs to a mixed-integer programming problem and proposed two effective heuristic algorithms.

While the above works are very effective in achieving their corresponding objectives, these algorithms are not able to distinguish the differences in tile communication latency between the two types of networks. For instance, in mesh networks, as long as two tiles (e.g., *A* and *B* in Figure 1b) have the same Manhattan distances from a source tile (e.g., *S* in Figure 1b), the latencies are the same; whereas in express channel-based networks, the tile with less turns has shorter latency (e.g., 7 cycles from *S* to *A* in Figure 1c) than the tile with more turns (e.g., 11 cycles from *S* to *B* in Figure 1c). Therefore, applying existing mapping algorithms to express channel-based NoCs may result in suboptimal or inefficient mapping solutions.

III. Problem Statement

A. Network, Application, and Average Packet Delay

Several important definitions are given below.

Definition 1 Network Topology:

1) A $n \times n$ CMesh network has a network size of $N = n^2$ tiles.
2) Concentration degree c is the number of processing elements (PEs) that can be placed on one tile.

Therefore, a $n \times n$ CMesh-based MPSoC with a concentration degree of c can hold at most $n^2 c$ PEs.

Definition 2 Application:

3) An application contains a set of tasks $\{t_i\}$, each executed on one PE. Tasks communicate with each other during execution to exchange data, maintain coherency, etc.
4) A task cluster tc_i is a set of tasks that are grouped together to be placed on one tile of a CMesh network. Concentration degree c indicates a task cluster tc_i contains at most c tasks.

Since the partitioning of tasks into task clusters greatly depends on the specific functionalities and restrictions of each task in a particular application, in this paper, we assume the task clusters $\{tc_i\}$ are given for an application, and focus on the main problem of mapping task clusters to tiles on the NoC.

Definition 3 An application mapping solution is a permutation $\pi(i) = j,\ i,j \in \{1,2,\dots,N\}$, so that task cluster tc_i is mapped to tile l_j.

In order to give a formal definition of average packet delay, we define the communication graph of an application and the tile delay graph of a given NoC topology as follows.

Definition 4 A communication graph $G_C(V_C, E_C)$ is a directed graph, in which each vertex vc_i represents a task cluster tc_i and each edge $ec_{ij} = (vc_i, vc_j)$ denotes the communication from tc_i to tc_j. The weight associated with edge ec_{ij} denotes the communication rate r_{ij}, i.e., the average number of flits sent from tc_i to tc_j per unit time.

Definition 5 A tile delay graph $G_D(V_D, E_D)$ is a complete directed graph, in which each vertex vd_i represents a tile l_i. There is an edge $ed_{ij} = (vd_i, vd_j)$ between any two vertices (tiles). The weight associated with edge ed_{ij} represents the delay d_{ij} from tile l_i to tile l_j when following the routing path (e.g., XY routing path) from l_i to l_j.

Given that task cluster tc_i is mapped to tile $l_{\pi(i)}$, the average packet delay of an application can be defined as follows.

Definition 6 The average packet delay (APD) of an application can be calculated by

$$APD = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij} d_{\pi(i)\pi(j)}}{\sum_{i=1}^{N}\sum_{j=1}^{N} r_{i,j}} \quad (1)$$

Note that this equation is applicable to both CMesh networks as well as networks with express channels. The key difference is the tile delay model d_{ij} used in task delay graph in Definition 5, which is discussed next.

B. Delay Models

1) Tile delay model for CMesh networks

Definition 7 Unit-length link delay T_L is the number of cycles (typically 1) between neighboring tiles. Delays for long express channels are proportional to the length. Router delay T_R is the number of cycles a packet takes to go through a router, i.e., the number of router pipeline stages.

In CMesh networks without express channels, each packet has to go through the entire router pipeline for each hop it travels. Therefore the tile delay on CMesh network without express channels can be calculated by:

$$\tilde{d}_{ij} = (M(i,j)+1)(T_R + t_c) + M(i,j)T_L \quad (2)$$

where $M(i,j)$ is the Manhattan distance between tile l_i and l_j, and t_c is the per router contention latency which depends on traffic load. In contemporary NoCs, because of the large link-width (e.g., 256-bit) and low load of real applications, the value of t_c is usually between 0.5 to 1 cycles per router. Also note that this delay model has already included the injection router and the ejection router to account for end-to-end tile delay.

2) Tile delay model for express channel-based networks

To derive the tile delay model for express channel-based networks, we first define an auxiliary turn function as below:

Definition 8 A turn function $\delta(i,j)$ is used to identify whether packets sent from tile l_i to tile l_j need to make a turn assuming XY routing:

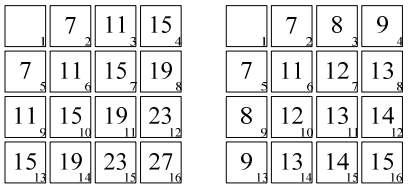
$$\delta(i,j) = \begin{cases} 0, & i-(i \bmod n) = j-(j \bmod n) \\ 0, & i \bmod n = j \bmod n \\ 1, & otherwise \end{cases} \quad (3)$$

The turn function is crucial in determining the packet delay on express-channel networks. If l_i and l_j are on the same row or column, the router of l_i will directly send packets to the express channel from l_i to l_j, so that packets only go through two router pipelines (the injection router and ejection router) before reaching the destination tile. Otherwise, packets are sent to the router of the turning point tile first, which is in the same column with the destination tile. Packets go through three routers in total in this case.

With the above turn function $\delta(i,j)$, the tile delay model from tile l_i to tile l_j can be expressed by:

$$d_{ij} = \big(2+\delta(i,j)\big)(T_R + t_c) + M(i,j)T_L \quad (4)$$

Figure 2 exemplifies the base packet latency from tile l_1 to all other tiles in a CMesh-based NoC and express channel-based networks, assuming $T_L = 1$ and $T_R = 3$ (the 3-cycle router follows a canonical pipeline design consisting of virtual channel allocation, switch allocation and switch traversal, with the optimization of look-ahead routing to hide routing computation). Figure 2 highlights why algorithms proposed for CMesh-based NoCs are less effective when applied to express channel-based NoCs directly. In the CMesh delay model, tile l_4, l_7, l_{10} are l_{13} are considered to have the same packet delay to l_1; whereas in the new delay model with express channels, l_7 and l_{10} have 33% larger delays compared to the other two.



(a) Tile delay on CMesh (b) Tile delay on MECS

Figure 2. Tile delay of packets with source at tile l_1.

C. Problem Formulation

With the above definitions and delay models, we can formulate the application mapping problem as follows:

Given:

1) An express channel-based network, containing n^2 tiles;
2) The application communication graph $G_C(V_C, E_C)$, with communication rate r_{ij} as the edge weight; and
3) The tile delay graph $G_D(V_D, E_D)$, with delay d_{ij} as the edge weight;

Find: Mapping of task clusters to tiles:

$$\pi(i) = j, where\ i,j \in \{1,2,\dots,N\}$$

Minimize the average packet delay:

$$APD = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij} d_{\pi(i)\pi(j)}}{\sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij}} \quad (5)$$

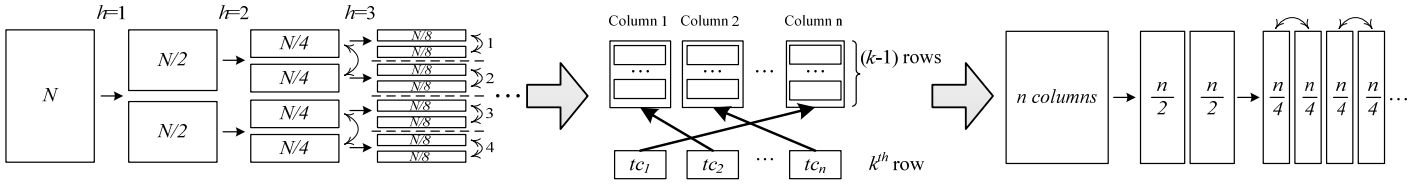
The above formulated problem has the form of a *Quadratic Assignment Problem* (QAP). A general QAP is *NP-hard* [6]. Enumerating all $(n^2)!$ possible solutions is costly even for a simple 4×4 NoC, not to mention larger networks. However, the special characteristics of the tile delay model of express-channel networks may give us some insights for designing effective heuristic algorithms.

IV. Proposed Algorithm

In this section, we propose an efficient heuristic algorithm that runs in polynomial time for application mapping in express channel-based networks. The proposed algorithm, *Turn Reduction Algorithm for Mapping (TRAM)*, utilizes the following two observations. First, as tiles on the same row or column have smaller packet delay, aligning task clusters with large communication rate in the same row or column can effectively reduce both delay and turns. Second, similar to mapping methods on CMesh networks, as the link delay linearly depends on the Manhattan distance between source and destination tiles according to Equation (4), it is still beneficial to put task clusters as close to each other as possible. TRAM contains three main steps to realize these objectives.

Step 1 Partition n^2 task clusters into n sets and place each set on one row of the express-channel network.

The partitioning is based on Kernighan–Lin (KL) algorithm [11], an efficient heuristic algorithm for solving graph partitioning problems. It attempts to partition a graph into two sets with equal sizes, such that the sum of edge weights between vertices in the two sets are minimized (min-cut).



(a) Step 1: Row Placement (b) Step 2: Column arangement (c) Step 3: Column Adjustment

Figure 3. Three steps of TRAM.

We call KL algorithm in a hierarchical fashion until we get n sets each with n task clusters, as shown in **Error! Reference source not found.**. After each two-way partitioning, we use a heuristic to determine the placement of the two sets. Take the $N/8$ partitioning stage in **Error! Reference source not found.**(a) as an example. We name each two sets a KL section (i.e., KL sections are labeled 1 to 4). The order among these four KL sections is decided at the previous stage, and KL has finished the partitioning in the current four KL sections. The orders of the pair of sets within each KL section need to be determined. Consider the KL section 2, which contains the third and fourth sets. Let m_3^{high} denote the total communication rate between the third set and all the sets above KL section 2 (i.e. section 1), and m_3^{low} denote the total communication rate between the third set and all the sets below section 2 (i.e. section 3 and 4). Similarly we define m_4^{high}, m_4^{low} for the fourth set. We calculate and compare the differences between high/low communication rate, i.e. $g_3 = m_3^{high} - m_3^{low}$ and $g_4 = m_4^{high} - m_4^{low}$, and then place the set with higher g in the third row and the other in the fourth row, so that the heavier communication is put closer to the outside of the KL section. The orders in other sections are determined similarly. The complete pseudo code for step 1 is shown below:

---

for h from 1 to $\log_2 n$
  // current number of sections is 2^{h-1}
  // in this iteration we get 2^h sets
  for i from 1 to 2^{h-1}
    in current section S_i, call KL to get the new $(2i-1)$-th and $(2i)$-th sets
    $m_{2i}^{high} = \sum r_{kl}$ ($k \in (2i)$-th set, $l \in S_j, j < i$)
    $m_{2i-1}^{high} = \sum r_{kl}$($k \in (2i-1)$-th set, $l \in S_j, j < i$)
    $m_{2i}^{low} = \sum r_{kl}$ ($k \in (2i)$-th set, $l \in S_j, j > i$)
    $m_{2i-1}^{low} = \sum r_{kl}$ ($k \in (2i-1)$-th set, $l \in S_j, j > i$)
    $g_{2i} = m_{2i}^{high} - m_{2i}^{low}$
    $g_{2i-1} = m_{2i-1}^{high} - m_{2i-1}^{low}$
    if $g_{2i} > g_{2i-1}$ place $(2i)$-th set at $(2i-1)$-th row
                place $(2i-1)$-th set at $(2i)$-th row
    else place $(2i)$-th set at $(2i)$-th row
                place $(2i-1)$-th set at $(2i-1)$-th row

The time complexity of KL algorithm is $O(n^6)$ since the graph has n^2 vertices. Calculating m^{high} and m^{low} takes $O(n^4)$ operations. Therefore the time complexity of Step 1 is $O(n^6) = O(N^3)$ according to the master theorem [3].

Step 2 Distribute task clusters in each set to the columns of the network.

The first step fixes the positions of rows whereas the order of task clusters within each row remains unsolved. In Step 2, we iteratively distributes of task clusters within each row to the columns. The order of task clusters in the first row is randomly assigned, of which the possible performance loss can be restored in Step 3. At the k^{th} iteration, with the task clusters in the first $(k-1)$ rows already placed, the placement of the task clusters of the k^{th} set is determined to minimize the average packet delay considering the communication rate between the current row and the first $(k-1)$ rows, as shown in **Error! Reference source not found.**(b). The above problem at each iteration is an assignment problem: In the cost matrix $\{c_{ij}\}$, c_{ij} denotes the APD contributed by tc_i placed at the j-th column. It is solved by Hungarian algorithm [13] optimally. The pseudo code for Step 2 is shown below:

Randomly assign tasks clusters in the first row to each column;
for k from 2 to n (the k-th row)
  Calculate the $n \times n$ cost matrix $\{c_{ij}\}$;
  Call Hungarian with the cost matrix $\{c_{ij}\}$ as input;
  Assign task clusters in the k-th row to each column according to the Hungarian assignment results;

Hungarian algorithm can achieve a time complexity of $O(n^3)$. Calculating the cost matrix $\{c_{ij}\}$ has a time complexity of $O(n^4)$. Therefore the time complexity of Step 2 is $O(n^5)$.

Step 3 Rearrange the columns to minimize the link delay of communication traffic on horizontal links.

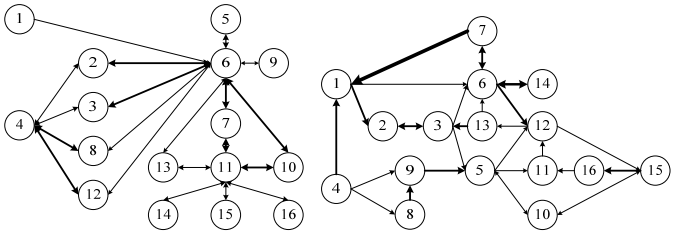
The process is similar to Step 1, except that each column is treated as a node in the input graph of KL algorithm. The time complexity of Step 3 is $O(n^3)$.

Taking into account all the three steps, the overall time complexity of the proposed algorithm is $O(N^3)$.

V. Evaluation Methodology

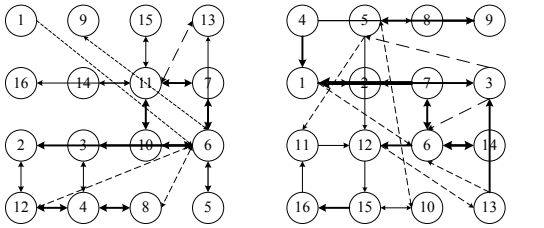
A. Schemes Under Comparison

As mesh network without concentration has much higher latency than other structures, in order to provide more fair comparison, we use CMesh as the baseline. The following six application mapping schemes on CMesh and MECS architectures are compared: 1) **MC_CMesh** (the baseline): Monte Carlo method on CMesh, which picks the mapping with the smallest latency among a large number of randomly generated mapping solutions based on CMesh structure; 2) **SA_CMesh**: simulated annealing algorithm on CMesh structure; 3) **MC_MECS**: Monte Carlo method on MECS structure; 4) **SA_MECS**: simulated annealing algorithm on MECS structure using the new tile delay model; 5) **SA_CMesh(MECS)**: the mapping solution is first generated by SA_CMesh, and then apply the solution on MECS structure; and 6) **TRAM**: our proposed approach.



(a) mpeg4 Communication Graph (b) toybox Communication Graph

Figure 4. Communication graph for mpeg4 and toybox.



(a) mpeg4 Communication Graph (b) toybox Communication Graph

Figure 5. Mapping results of mpeg4 and toybox.

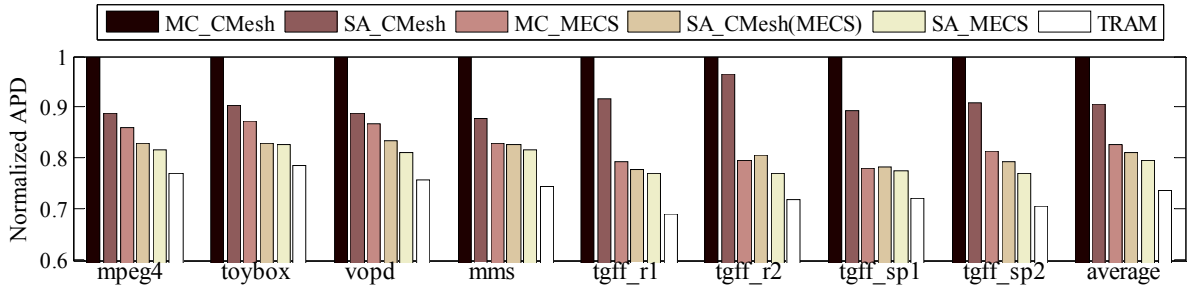


Figure 6. Normalized average packet delay for eight different applications.

Since Monte Carlo and simulated annealing are algorithms that have tradeoff between runtime and performance, for fair comparison, the runtime of both algorithms are configured to be roughly the same as the runtime of our proposed algorithm.

B. Simulation Setup

The proposed TRAM algorithm is evaluated quantitatively under both typical and stressed workloads. This includes the traces of four real applications, namely mpeg4, toybox, vopd, and mms, as well as four random task graphs generated by TGFF [4], referred to as tgff_r1, tgff_r2 tgff_sp1 and tgff_sp2. Figure 4 shows the communication rate graph of mpeg4 and toybox (vopd and mms are omitted here due to space limitation). Each node denotes a task cluster, and the edge width indicates the relative magnitude of the communication rate. The tgff_r1 and tgff_r2 are two random graphs while tgff_sp1 and tgff_sp2 are two series-parallel graphs formed recursively by joining two sub-graphs in series and parallel, mimicking the stressed behaviors of multithreaded applications. Collectively, these eight inputs comprise a representative set of MPSoC scenarios. A 64-task configuration with concentration degree 4 is simulated for majority of the evaluation. In addition, 256-task configuration is also evaluated for scalability discussion.

In the simulation results, the APDs are calculated according to our delay model. Runtime is based on a machine with an Intel Core i7-3770 processor. NoC power is calculated using the latest NoC power model *dsent* [17] under 45nm and 1V. The unit-length link delay T_L is set to 1 and T_R is set to 3. For each of the test case, the contention delay t_c is acquired by feeding the trace in a cycle-accurate NoC simulator.

VI. Results and Analysis

A. Impact on Performance

We first evaluate the effectiveness of TRAM to reduce turns. Table I compares the percentage of communication traffic that needs to make turns in express-channel networks for different algorithms. It can be seen that the proposed TRAM is able to achieve an average of 2~4X reduction in the percentage compared to other algorithms. Figure 5 presents the mapping results obtained by TRAM for mpeg4 and toybox. A dashed arrow means the packet from source to destination tile needs to take a turn. When TRAM is used, only 11.8% and 4.2% of the traffic needs to make turns for mpeg4 and toybox, respectively. It is worth noting that, while the proposed algorithm is optimizing for the number of turns, most of the heavily communicating tasks (as indicated by wider edges) are also mapped close to each other, as can be seen from Figure 5.

The reduced turns and closer physical distances result in considerable improvement of packet latency. Figure 6 plots the results of average packet delay for the eight different test cases. Compared to the baseline system, the proposed TRAM algorithm reduces the packet delay by 26.5% on average. Also, TRAM is 10% better than SA_CMesh(MECS). This indicates that the mapping solution generated from CMesh-based networks is not optimal when applied to express channel-based networks.

B. Impact on Power Consumption

Although the primary objective is to reduce packet delay, the proposed TRAM is also able to slightly reduce power consumption as a side effect, because the algorithm reduces the number of routers and links through which packets need to travel. Table II shows the dynamic power of different mapping algorithm solutions on various applications. It can be seen that, even though TRAM does not target for power optimization, it still achieves the lowest dynamic power consumption among all schemes.

C. Impact of Pipeline stages

So far we have assumed a 3-stage router pipeline, which is an optimized version on top of the canonical 4-stage router. Equation (5) indicates that the number of router pipeline stages may affect the latency of express-channel networks. To assess this impact, Figure 7 compares the mapping results of simulated annealing on CMesh networks, simulated annealing on MECS and the proposed TRAM on MECS while varying the numbers of pipeline stages (T_R) from 1 to 4. As can be seen,

TABLE I. PERCENTAGE OF TRAFFIC THAT NEEDS TO MAKE TURNS (IN %).

Systems	Percentage (%)								
	mpeg4	***toybox***	***vopd***	***mms***	***tgff_r1***	***tgff_r2***	***tgff_sp1***	***tgff_sp2***	***Average***
MC_MECS	40.82	31.26	34.16	37.62	56.73	50.07	33.27	48.30	41.53
SA_CMesh(MECS)	38.47	22.53	31.67	20.98	48.71	48.78	43.02	43.04	37.15
SA_MECS	25.03	15.00	19.41	11.96	40.97	39.98	36.92	27.69	27.12
TRAM	**11.80**	**4.22**	**4.37**	**0.12**	**20.93**	**19.52**	**21.23**	**12.10**	**11.79**

TABLE II. DYNAMIC POWER CONSUMPTION.

Systems	Dynamic Power (mW)							
	mpeg4	***toybox***	***vopd***	***mms***	***tgff_r1***	***tgff_r2***	***tgff_sp1***	***tgff_sp2***
MC_CMesh	95.91	130.05	167.25	42.30	107.28	121.83	119.86	100.31
SA_CMesh	91.38	125.46	156.93	40.77	105.24	110.67	108.31	92.31
MC_MECS	84.22	113.51	146.54	38.79	87.59	97.26	96.15	83.98
SA_CMesh(MECS)	82.46	111.74	145.07	38.72	85.99	96.98	95.67	83.13
SA_MECS	80.69	105.63	140.17	38.09	85.80	96.44	94.17	81.27
TRAM	77.07	104.55	129.78	37.91	82.66	91.62	88.80	76.12

the proposed TRAM is effective across different number of pipeline stages. This illustrates that TRAM can be useful in a wide range of networks built from more aggressive or more conservative router architectures.

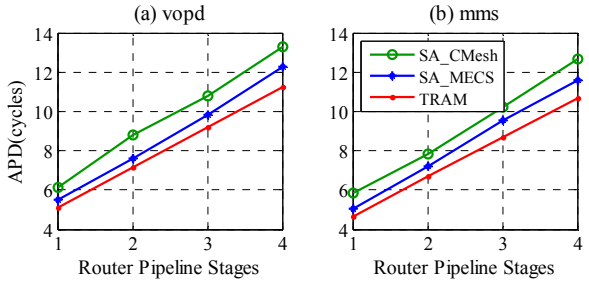


Figure 7. Average packet delay as a function of router pipeline stages.

D. Scalability

Previous evaluation uses 64-task configurations with concentration degree of 4. To further illustrate the scalability of the proposed algorithm, we generate four TGFF configurations of 256 tasks with the same concentration degree. Simulation results show that, compared with MC_CMesh and SA_MECS, TRAM is able to reduce the average packet delay by 55% and 23% under the same runtime, respectively. This demonstrates that the proposed TRAM can achieve higher improvement for larger networks, indicating its good scalability.

VII. CONCLUSION

Express channel-based networks have been proposed in recent studies as a promising approach to support fast on-chip communications for current and future many-core MPSoCs. However, the characteristics of these new topologies have not been exploited in existing application mapping algorithms. In this paper, we propose an efficient heuristic algorithm to explore the application mapping opportunities in express-channel networks. The proposed TRAM algorithm is able to effectively map tasks with large communication rate closer to each other, and aligns heavily communicating tasks to the same rows or columns to reduce unnecessary turns. Simulation results show significant reduction in the number of turns and considerable reduction in average packet delay in the generated mapping solutions.

REFERENCES

[1] Balfour, J., & Dally, W. J. (2006). Design tradeoffs for tiled CMP on-chip networks. In *ACM International Conference on Supercomputing*.

[2] Chen, G., Li, F., Son, S. W., & Kandemir, M. (2008). Application mapping for chip multiprocessors. In *Design Automation Conference*.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. MIT press.

[4] Dick, R. P., Rhodes, D. L., & Wolf, W. (1998). TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign* (pp. 97-101). IEEE Computer Society.

[5] Faruque, A., Abdullah, M., Krist, R., & Henkel, J. (2008, June). ADAM: run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of the 45th annual Design Automation Conference* (pp. 760-765). ACM.

[6] Garey, M. R., & Johnson, D. S. (1979). Computers and intractability A Guide to the Theory of NP-Completeness.

[7] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu (2009). Express cube topologies for on-chip interconnects. In *International Symposium on High Performance Computer Architecture* (pp. 163-174).

[8] J. Howard, S. Dighe, Y. Hoskote, et al. (2010). A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *IEEE International Solid-State Circuits Conference* (pp. 108-109)

[9] Hu, J., & Marculescu, R. (2003). Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proceedings of the ASP-DAC*.

[10] Jang, W., & Pan, D. Z. (2012). A3MAP: Architecture-aware analytic mapping for networks-on-chip. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *17*(3), 26.

[11] Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49.

[12] J. Kim, J. Balfour, and W. J. Dally (2007) . Flattened butterfly topology for on-chip networks. In *IEEE/ACM International Symposium on Microarchitecture* (pp. 172-182).

[13] Kuhn, H. W. (2005), The Hungarian method for the assignment problem. *Naval Research Logistics*.

[14] Kumar, A., Peh, L.-S., Kundu, P. & Jha, Niraj K. (2007). Express virtual channels: Towards the ideal interconnection fabric. In *IEEE International Symposium on Computer Architecture*.

[15] Lei, T., & Kumar, S. (2003, September). A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Digital System Design, 2003. Proceedings. Euromicro Symposium on* (pp. 180-187). IEEE.

[16] Murali, S., & De Micheli, G. (2004). Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings of the conference on Design, automation and test in Europe*.

[17] Sun, C., Chen, C., Kurian, G., et al. (2012). DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *International Symposium on Networks-on-Chip*.

[18] Tilera Corporation. http://www.tilera.com/products/processors.