A Power-Aware Mapping Approach to Map IP Cores onto NoCs under Bandwidth and Latency Constraints

XIAOHANG WANG
Zhejiang University
MEI YANG and YINGTAO JIANG
University of Nevada, Las Vegas
and
PENG LIU
Zhejiang University

In this article, we investigate the Intellectual Property (IP) mapping problem that maps a given set of IP cores onto the tiles of a mesh-based Network-on-Chip (NoC) architecture such that the power consumption due to intercore communications is minimized. This IP mapping problem is considered under both bandwidth and latency constraints as imposed by the applications and the on-chip network infrastructure. By examining various applications' communication characteristics extracted from their respective communication trace graphs, two distinguishable connectivity templates are realized: the graphs with tightly coupled vertices and those with distributed vertices. These two templates are formally defined in this article, and different mapping heuristics are subsequently developed to map them. In general, tightly coupled vertices are mapped onto tiles that are physically close to each other while the distributed vertices are mapped following a graph partition scheme. Experimental results on both random and multimedia benchmarks have confirmed that the proposed template-based mapping algorithm achieves an average of 15% power savings as compared with MOCA, a fast greedy-based mapping algorithm. Compared with a branch-and-bound–based mapping algorithm, which produces near optimal results but incurs an extremely high computation cost, the proposed algorithm, due to its polynomial runtime complexity, can generate the results of almost the same quality with much less CPU time. As the on-chip network size increases, the superiority of the proposed algorithm becomes more evident.

This work has been supported by NSF under grant no. ECCS-0702168 and Chinese NSF under grant no. 60873112.
Authors' addresses: X. Wang and P. Liu, Department of Information Science and Electronic Engineering, Zhejiang University, No. 2 Yugu Rd., Hangzhou, Zhejiang 310027, P.R. China; e-mail: baikeina@yahoo.com.cn, liupeng@isee.zju.edu.cn. M. Yang and Y. Jiang, Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, 4505 Maryland Parkway, Box 454026, Las Vegas, NV 89154; e-mail: {meiyang, yingtao}@egr.unlv.edu.

DOI 10.1145/1736065.1736066 http://doi.acm.org/10.1145/1736065.1736066

Categories and Subject Descriptors: C.5.4 [**Computer System Implementation**]: VLSI Systems; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms: Algorithms, Performance, Verification

Additional Key Words and Phrases: Low power, network-on-chip (NoC), IP mapping, bandwidth and latency constraints

ACM Reference Format:

Wang, X., Yang, M., Jiang, Y., and Liu, P. 2010. A power-aware mapping approach to map IP cores onto NoCs under bandwidth and latency constraints. ACM Trans. Architec. Code Optim. 7, 1, Article 1 (April 2010), 30 pages.
DOI = 10.1145/1736065.1736066 http://doi.acm.org/10.1145/ 1736065.1736066

1. INTRODUCTION

With the continuous scaling of CMOS technologies, interconnects dominate both performance and power dissipation in future System-on-Chip (SoC) designs. In the 45-nm technology node, for instance, the delay of chip-edge long global wires can exceed 120 to 130 clock cycles [Ho et al. 2001; Meindl 2003]. Even after aggressive repeater insertion, the delay of global wires may still be longer than one clock cycle. Besides increased interconnect delay, according to the International Technology Roadmap for Semiconductor (ITRS) [2007], without changes in interconnect design philosophy, in 2012, up to 80% of microprocessor power will be consumed by interconnect itself.

As an architectural remedy to the increased wire delay, which makes it practically impossible to design a global synchronous system on a chip [Kodi et al. 2008], multiprocessor system-on-chip (MPSoC) designs [Bertozzi et al. 2005], which integrate multithreaded processors and multiprocessor cores running at their own clock domains into a single chip have emerged. MPSoC is shown to deliver high performance yet reasonably low power consumption, making it an architecture of choice suitable for many embedded system applications. As the number of processor cores on an MPSoC chip is continuously increasing, traditional bus-based architecture tends to create a communication bottleneck. As a result, network-on-chip (NoC) has been considered as a viable communication infrastructure in MPSoC designs due to its distinct advantages of structure, performance, and modularity [Dally and Towles 2001; Jantsch and Tenhunen 2003].

For a target application to be running on an NoC-based MPSoC architecture, a three-step design flow similar to the one suggested in Hu and Marculescu [2005] is generally followed.

—Step 1: A target application is partitioned into multiple concurrent tasks.
—Step 2: The tasks are scheduled and allocated to selected IP cores.
—Step 3: IP cores selected from Step 2 are optimally mapped onto the NoC-based MPSoC architecture in terms of power, or performance, or a combination of these figure of merits under the constraints of latency and/or bandwidth [Ogras et al. 2005]. After mapping, all the routing paths between any pair of communicating IP cores are optimally identified.

The partitioning and scheduling problems in Steps 1 and 2 can be approached following the lessons learned from partitioning and scheduling in parallel computers but with interprocessor communication considered [Chang and Pedram 2000]. In this article, our study is limited to mapping the IP cores onto a regular tile-based NoC architecture (Step 3 in the design flow) and the mapping is subjected to the latency and bandwidth constraints as imposed by many real-time multimedia applications. Regular mesh remains the dominant NoC architecture of choice due to its distinct features: structured network wiring, modularity, and standard interfaces [Hu and Marculescu 2005].

In the literature, a number of algorithms have been proposed to solve this IP mapping problem, and they fall into four general categories: (i) branch-and-bound algorithms [Hu and Marculescu 2005; Lin et al. 2008], (ii) framework-based approach using simulated annealing (SA) [Harmanani and Farah 2008; Lu et al. 2008]/genetics algorithm (GA) [Ascia et al. 2004; Zhou et al. 2006]/ tabu search (TS) [Marcon et al. 2007], (iii) linear-programming-based schemes [Murali and De Micheli 2004], and (iv) greedy-based heuristics [Hansson et al. 2005; Mehran et al. 2007; Srinivasan and Chatha 2005]. Generally, branch-and-bound–based algorithms can generate very-high-quality results, very close to the optimized solutions, provided that the size of work queue is unbounded. With a large queue size, however, branch-and-bound algorithms often require huge memory depth and CPU time. The algorithms that fall into the second and third categories may produce reasonably high-quality solutions under certain conditions. The greedy-based algorithms, on the other hand, may generate lower-quality mapping results with significantly low CPU time required. In Section 2, a more detailed review of these existing approaches is provided.

In this article, we propose a template-based efficient mapping (TEM) algorithm that generates high-quality mapping results with low runtime. This algorithm is designed based on two distinguishable connectivity templates extracted from various applications' communication trace graphs: the graphs with tightly coupled vertices and those with distributed vertices. Correspondingly, different mapping strategies are proposed for these two templates. Simulation results show that the proposed TEM algorithm achieves significant improvement over MOCA [Srinivasan and Chatha 2005], the known best greedy algorithm in terms of the runtime and the quality of the solutions. Compared with a branch-and-bound algorithm with a large queue size, which produces near optimal results but incurs an extremely high computation cost, the proposed algorithm, due to its polynomial time complexity, can generate the results of almost the same quality but with much less CPU time.

The rest of the article is organized as follows. Section 2 reviews the approaches that have been proposed to solve the IP mapping problem, which is formally defined in Section 3. Section 4 introduces the two templates derived from various applications' communication trace graphs. The mapping algorithm based on the two templates is presented in Section 5 along with a few illustrative examples. Section 6 reports and discusses the simulation. Finally, Section 7 concludes the article.

2. RELATED WORK

As shown in Hu and Marculescu [2005], IP mapping is an NP-hard problem, as it is an instance of the constrained quadratic assignment problem [Garey and Johnson 1979]. Various heuristic algorithms have been proposed, and they generally can be categorized into following four categories.

Branch-and-bound algorithms. Hu and Marculescu [2005] proposed a branch-and-bound algorithm that attempts to search for the optimal solution through alternating branch and bound steps. This branch-and-bound algorithm can generate nearly optimized solutions due to its large search space, assuming the size of its work queue is set to infinity. With large queue size, however, this algorithm demands high memory depth and suffers from long CPU time. For mapping N tasks to N tiles, the timing complexity is $O(N!)$. For 6×6 mesh, the runtime (on a PC with one Intel Core2 P8600 2.4GHz processor and 2GB RAM) of the branch-and-bound algorithm can be a few hours if the queue is unlimited. The memory used is $36^*4^*36!(10^{44})$ byte assuming the queue elements are composed by 36 integers that store the tile info. As such, this approach is only feasible for mapping IP cores onto an NoC with a fairly small size, unless a trade-off between the runtime and the quality of solutions is made. In the literature, techniques have been suggested [Lin et al. 2008; Murali and De Micheli 2004] to reduce both the memory requirement and runtime by largely limiting the size of work queue, which adversely impacts the search space and thus the quality of the final solution. Actually, a branch-and-bound scheme boils down to a greedy algorithm if the size of the work queue is chosen to be very small; in this case, most of the solutions are trimmed off, resulting in a much reduced search space (i.e., local search dominates).

Greedy-based heuristic algorithms. Various heuristic algorithms [Hansson et al. 2005; Murali and De Micheli 2004] have been proposed based on different observations on the properties of both NoC topologies and the communication patterns among IP cores. Generally, greedy-based IP mapping algorithms have very low runtime but often at a sacrifice of the quality of solutions. However, as we will show in this article, if the algorithm is properly designed, such degradation of solution quality can be minimized. In the following, we review a few noteworthy greedy algorithms in greater detail.

To reduce the overall power consumption, one observation is that IP cores with demanding communication requests (i.e., communication between two IP cores requesting tight latency or high bandwidth) should be mapped first. The mapping with minimum-path routing algorithm [Murali and De Micheli 2004], the flow traversal algorithm [Hansson, et al. 2005], and the LCF algorithm [Marcon et al. 2007] are all based on this observation. These algorithms may fail to produce high-quality solutions when there is one IP core (referred as current IP core) needs to communicate with a large number of cores (referred as neighbor IP cores). In this case, the current IP core may be mapped onto a tile with its number of neighbor tiles far less than the number of neighbor IP cores. As a result, in the subsequent mapping steps, many of the current IP core's neighbor IP cores shall have to be mapped onto the tiles that are farther away from the current IP core. The communications between the current IP core and many of its neighbor IP cores thus may physically go through unnecessarily long distance with extra power consumption.

The previously mentioned mapping problem can be alleviated by taking the degree of each IP into mapping consideration. In Spiral [Mehran et al. 2007], an IP core with a large number of neighbors (degree of the IP core) is prioritized to be mapped onto a tile that connects to a large number of neighbor tiles (e. g., the center of a mesh-based NoC), thus resulting in lower communication delay. This algorithm, however, may fail to generate high-quality solutions either when there are IP cores with large degrees but the communications among these cores and their neighbors do not impose high bandwidth/tight latency requirements, or when the degree of each IP is small and all the IPs have nearly identical degree values.

The algorithm MOCA [Srinivasan and Chatha 2005] achieves a sound balance between the runtime and the quality of solutions. It uses a graph partition algorithm [Hendrickson and Leland 1995] to recursively partition both the NoC and the communication trace graph into two halves and in each iteration, one part of the communication trace graph is mapped to one region of the NoC. One problem of this algorithm is that pairs of communicating IP cores that require high bandwidths but can tolerate long latency may be mapped to tiles with high hop counts, resulting in higher power consumption. Also, according to our experimental results, the power performance of MOCA deteriorates drastically when latency constraints are applied.

Simulated annealing (SA)/genetics algorithm (GA)/tabu search (TS) framework. The genetic algorithms proposed in Ascia et al. [2004] and Zhou et al. [2006] for IP mapping target multiple objectives (e.g., low power, high throughput, or a combination of those). Harmanani and Farah [2008] proposed a simulated annealing (SA)-based algorithm, and Lu et al. [2008] proposed a cluster-based SA algorithm such that the IP mapping can be done in a divide-and-conquer–like manner. In these algorithms, design of the convergence condition is the key to finding a high-quality solution. Although these algorithms are capable to avoid local minima, finding high-quality solutions is not guaranteed. Note that these algorithms typically require considerably longer time than a greedy-based mapping algorithm.

Linear programming-based algorithms. Linear programming (LP) is used in Murali and De Micheli [2004] to solve the mapping problem. Here, forming the right LP problem is the key to obtaining high-quality solutions. Although there several software tools exist to solve LP problems, computation is still time-consuming and there is no guarantee that high-quality solutions can always be found.

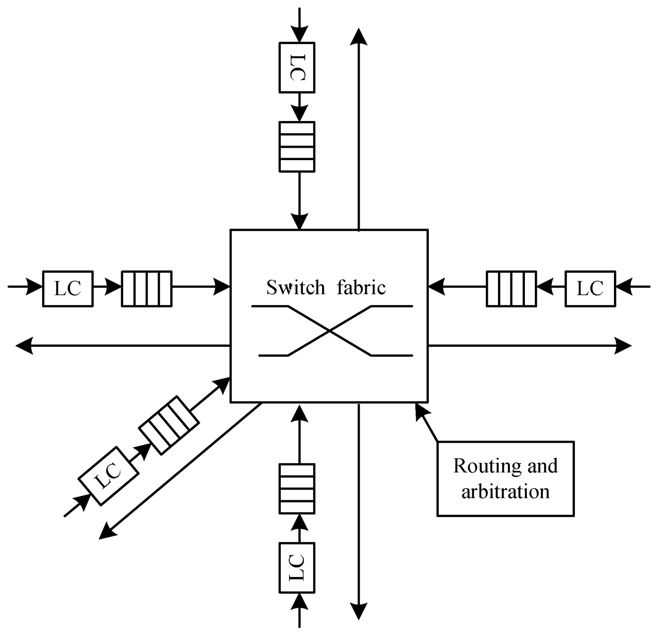

Fig. 1. The on-chip router structure.

3. PROBLEM FORMULATION

3.1 Architecture Description and Power Model

The NoC system under consideration is composed of $N \times N$ tiles interconnected by a 2D mesh network. Each tile, indexed by its coordinate (x, y), where $0 \leq x \leq N - 1$ and $0 \leq y \leq N - 1$, has one processing core and one router. Each router (shown in Figure 1) connects to its local processing core and four neighbor tiles through bidirectional channels. Buffers are included at each router input port, and a 5×5 crossbar switch is used as the switching fabric of the router. The link controllers (LCs) control the flow of traffic across various connections. The routing and arbitration component decides the routing paths, selects the output link for an incoming message, and accordingly guides the switch to switch the traffic.

The power model used in Hu and Marculescu [2005] is followed in this study. The bit power (E_{bit}) is defined as the power consumed when 1 bit of data is transported through a router, and it can be calculated as

$$E_{bit} = E_{Sbit} + E_{Bbit} + E_{Wbit} \tag{1}$$

where E_{Sbit}, E_{Bbit}, and E_{Wbit} represent the power consumed by the switch, the buffer, and the interconnection wires inside the switching fabric, respectively. As explained in Hu and Marculescu [2005], E_{Bbit} and E_{Wbit} are negligibly small. Hence, the average power consumption for sending 1 bit of data from tile t_i to tile t_j can be represented as

$$E_{bit}^{t_i,t_j} = \eta_{hops} \times E_{Sbit} + (\eta_{hops} - 1) \times E_{Lbit}, \tag{2}$$

where η_{hops} is the number of routers traversed from tile t_i to tile t_j, E_{Sbit} is the power consumed by the switch, and E_{Lbit} is the power consumed on the links between tiles t_i and t_j.

3.2 Problem Description

We assume that before IP mapping is performed, a given application described by a set of concurrent tasks is already bounded and scheduled onto a list of selected IP cores [Ogras et al. 2005]. The communication patterns between any pair of IP cores are modeled by the target application's communication trace graph, whereas the NoC architecture that the application will be mapped onto is described in terms of an architecture characterization graph.

Definition 1. A *communication trace graph* (CTG) $G = (P, E)$ is an undirected graph, where a vertex/node $p_k \in P$ represents an IP core (a processor, an ASIC device or a memory unit, etc.), and an edge $e_i = (p_k, p_j) \in E$ represents the communication trace between vertices p_k and p_j. For edge e_i,

—$\omega(e_i)$ defines the communication bandwidth request between vertices p_k and p_j given in bits per second (bps). $\omega(e_i)$ sets the minimum bandwidth that should be allocated by the network in order to meet the performance constraints.

—$\sigma(e_i)$ represents the latency constraint, which is given in number of hop count instead of an absolute number in cycles [Srinivasan and Chatha 2005].

—$W(e_i)$ represents the weight of edge $e_{i,}$ and it is defined in the same way as that in Srinivasan and Chatha [2005]. Of all the traces in the graph, let e_i be the trace with the highest bandwidth requirement, and e_j be the trace with the tightest (lowest) latency constraint. An integer K is defined as the minimum value required to ensure that among all the traces in the graph, $\frac{\omega(e_i)}{\sigma(e_i)^K} \le \frac{\omega(e_j)}{\sigma(e_j)^K}$. Once K is determined, $W(e_i) = \frac{\omega(e)}{\sigma(e)^K}$.

A CTG can be transformed from its corresponding application characterization graph (APCG) [Hu and Marculescu 2003] as follows. The vertices in the CTG are the same as those in the APCG. For each edge e_i, $\omega(e_i)$ is obtained by combining the communication volumes/bandwidth request of the two possible edges on different directions between two vertices connected by e_i in the APCG. The latency constraint of e_i is the tighter one of the two possible edges connected by e_i in the APCG.

Definition 2. An *architecture characterization graph* (ACG) $\breve{G} = (T, L)$ is an undirected graph, where each vertex $t_i \in T$ represents a tile and each edge $l_i \in L = (t_k, t_j)$ represents the link between adjacent tiles t_k and t_j. For link l_i:

- $bw(l_i)$ defines the bandwidth provided on link l_i between adjacent tiles t_k and t_j;
- $c(l_i)$ defines the link cost of l_i, that is, power consumption for transmitting 1 bit data from t_k and t_j.

In this article, we focus on regular NoC architectures which have $bw(l_i) = B$, $c(l_i) = C$ for each $l_i \in L$, where B and C are constants. $h_{k,j}$ is the set of links forming one of the shortest paths from tile t_k to tile $t_j (h_{k,j} \subseteq L)$. $dist(h_{k,j})$ determines the number of elements in $h_{k,j}$ (i.e., it is the hop count of the shortest path between tile t_k and tile t_j).

Definition 3. A mapping algorithm M: $P \rightarrow T$ maps each vertex in CTG onto an available tile in ACG. $M(p_i)$ represents the mapped tile in ACG, where $p_i \in P$ and $M(p_i) \in T$.

Definition 4. A routing algorithm R: $E \rightarrow H$, finds one of the shortest routing path between $M(p_k)$ and $M(p_j)$ for each edge $e_i = (p_k, p_j) \in E$. The links of forming this path belongs to set $h_{M(p_k),M(p_j)}$.

The IP mapping problem is formulated as follows:

Given a CTG(P, E) representing the communication pattern of an application and an ACG(T, L) representing the target NoC architecture, where $|P| \leq |T|$, find a mapping $M : P \rightarrow T$ which maps all the vertices in CTG onto available tiles in ACG and generates a deadlock-free and minimal routing paths for all edges in CTG such that the total power consumption is minimized, that is,

$$\text{Min}\left\{\sum_{\substack{i=0\\ e_i=(p_k,p_j)\in E}}^{|E|} \omega(e_i) \times \sum_{\substack{m=0\\ l_m\in h_{M(p_k),M(p_j)}}}^{|h_{M(p_k),M(p_j)}|} C\right\} \tag{3}$$

satisfying

$$\forall p_i \in P, M(p_i) \in T \tag{4}$$

$$\forall p_i, p_j \in P \text{ and } p_i \neq p_j, M(p_i) \neq M(p_j) \tag{5}$$

$$\forall l_i,\ B \geq \sum_{e_i=(p_k,p_j)} \omega(e_i) \times f\left(l_i, h_{M(p_k),M(p_j)}\right) \tag{6}$$

$$\forall e_i = (p_k,\ p_j),\ \sigma(e_i) \geq dist\left(h_{M(p_k),M(p_j)}\right) \tag{7}$$

where $f(l_i, h_{M(p_k),M(p_j)}) = \begin{cases} 1\ if\ l_i \in h_{M(p_k),M(p_j)} \\ 0\ if\ l_i \notin h_{M(p_k),M(p_j)} \end{cases}$.

Similar to the definitions adopted in Hu and Marculescu [2005], the conditions given by Equations (4) and (5) ensure that each IP core should be mapped exactly to one tile and no tile can host more than one IP core. The inequities given in Equation (6) specify the bandwidth constraint for every link, and the inequities given in Equation (7) ensure that the latency constraint (in terms of number of hop count) between two communicating IPs is satisfied after the mapping.

4. DERIVATION OF THE MAPPINGTEMPLATES FROM CTGS

4.1 A Motivation Example

As alluded in Section 2, MOCA [Srinivasan and Chatha 2005] does not perform well when the latency constraints are considered. Derivation of the mapping templates can be better illustrated from the following example.

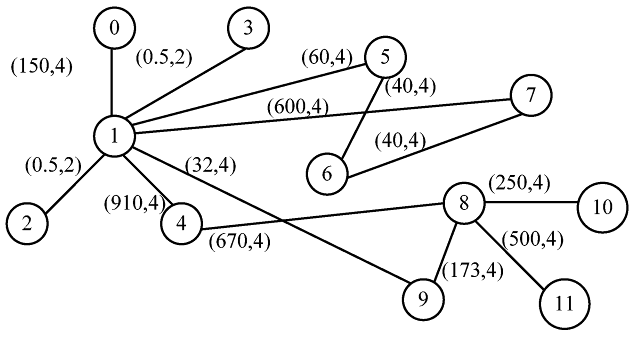

Fig. 2. The CTG of an MPEG4 decoder with 11 IP cores [Srinivasan and Chatha 2005]. The bandwidth request (Mbps) and latency constraint (number of hop count) of each edge is labeled on the edge.

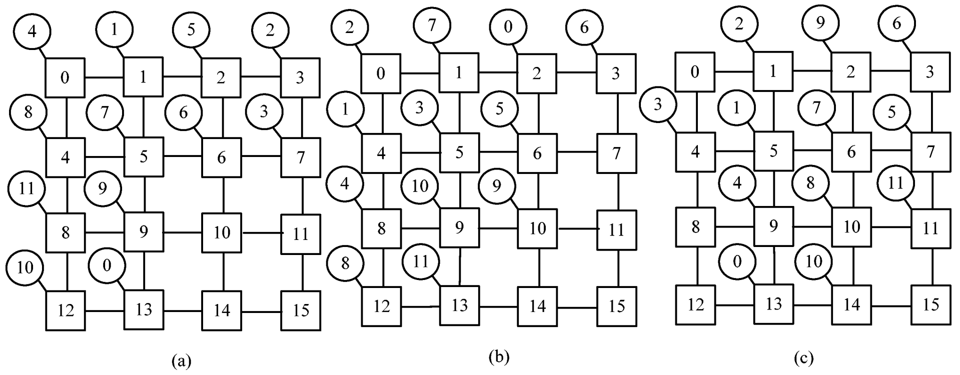

Fig. 3. (a) The mapping result of MOCA on the CTG of MPEG4 decoder without latency constraints applied in Figure 2. (b) The mapping result of MOCA on the CTG of MPEG4 decoder with latency constraints applied. (c) A better mapping solution on the CTG of MPEG4 decoder with latency constraint. The square boxes are tiles and the circles are IP cores.

The mapping solution without and with latency constraints obtained by MOCA is shown in Figures 3(a) and 3(b), respectively.

A communication path with higher bandwidth request should be mapped to links with lower hop count to reduce power consumption as Equation (3) indicates. In the CTG shown in Figure 2, one can see that of the seven neighbors of vertex 1, vertices 3, 2, 4, and 7 have the highest bandwidth/tightest latency requirements. Such neighbors are referred as four most significant neighbors (formally defined in Section 5). Consequently, it will be beneficial to map these four vertices onto tiles close to vertex 1 to reduce the hop count and subsequently help reduce the power consumption. In the same token, the four most significant neighbors of vertex 8 shall be mapped in a similar fashion; i.e., vertices 4, 11, 10, 9 should be mapped onto tiles with low hop count to vertex 8. When latency constraints are not considered, MOCA maps the pairs of communicating IP cores with high bandwidth requirements to tiles with minimum hop count. As shown in Figure 3(a), edge (1, 4) requests very high bandwidths and thus vertex pairs 1 and 4 are mapped to tiles that are one-hop-count away from each other. Edges (1, 7) and (8, 10) are mapped similarly. As such, the mapping result generated by MOCA is in very high quality. However, when the latency constraint is applied, the result of MOCA deteriorates. For example, in Figure 3(b), the hop counts of edges (1, 7), (8, 9), (8, 10) are greater than 1, which results in higher power consumption.

A better mapping solution can be obtained and shown in Figure 3(c). In this solution, edges in CTG with higher weights force the corresponding vertices to be mapped onto the tiles with lower hop counts than those edges with smaller weights. In Figure 2, vertices 1 and 8 have many edges with high bandwidths or tight latency. In Figure 3(c), these two vertices are mapped onto the two tiles with the largest degree or maximal number of neighbor tiles (i.e., 4). Therefore, the neighbors of the two vertices with which the two vertices form communications requesting higher bandwidth or tighter latency are mapped with lower hop counts to the two vertices such that the power consumption is reduced.

4.2 Classification of Applications

From the previous example, one can see that different mapping strategies shall be adopted for CTGs with different features. Here, we present two distinct templates derived from various CTGs: CTG that includes vertices with many significant neighbors defined later in the text, and CTG that does not have such vertices.

Given a CTG(P, E), and a sorted list of edges in decreasing order of edge weight denoted as $\hat{E}$:

Definition 5. A vertex $p_i \in P$ is a *hot node* if (a) p_i has a degree greater than or equal to α, and (b) of the first β edges in the edge list $\hat{E}$, there are at least γ edges that are connected to p_i.

Values of α, β, γ are topology- and application-dependent, and they can be set accordingly. From our experiment, we have determined that $\alpha = 4$ is appropriate for a mesh-based structure, and $\beta = 50\%$, $\gamma = 1$ are experimentally determined.

Template 1: An application falls into Template 1 (tightly coupled) if there is at least one hot node in its CTG.

Template 2: An application falls into Template 2 (distributed) if there is no hot node in its CTG.

5. ALGORITHM DESCRIPTION

Of the two distinct application templates described in Section 4, different mapping strategies shall be adopted. As such, the proposed mapping algorithm is named as TEM.

The overall structure of the TEM algorithm is shown as follows: Before the template-based mapping takes place, the edges have to be sorted in a nonincreasing order in terms of the edge weight. After all the vertices are mapped, a routing allocation routine is needed to help find routing paths for communicating vertices.

```
TEM ($G(P,E)$, $\breve{G}(T,L),\hat{E},MAP$)
Input: (1) $G$: CTG of an application.
       (2) $\breve{G}$: ACG of an NoC architecture
       (3) $\hat{E}$: A sorted edge list of CTG
Output: (1) $MAP1$: The table storing the mapping result for each IP core
Function: Map the application (CTG) onto the NoC architecture (ACG) and allocate routing paths
Procedure body:
{
    var:
    $MAP[1...|P|]$, $MAP1[1...|P|]$ ; // table element storing the tiles allocated for each IP core $p \in P$.
    // The values of tables are initialized to null.
    $\hat{E} = \{\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_{|E|}\}$; // a sorted edge list, s.t. $W(\hat{e}_1) \geq W(\hat{e}_2) \geq \ldots W(\hat{e}_{|E|})$ and for $1 \leq i \leq |E|$, $\hat{e}_i \in E$.
    $\hat{H} = \{ \varnothing \}$; // The set of hot nodes, initially empty
    // Find the hot nodes if any and determine which template the application belongs to
    for each vertex $p_i$ { // Definition 5
        if (vertex $p_i$ has degree $\geq \alpha$ and $p_i$ has $\gamma$ edges in the first $\beta$ edges in $\hat{E}$){
            mark $p_i$ as a hot node and insert into $\hat{H}$ ;
        }
    }
    // If $G$ has hot nodes, the application belongs to Template 1, otherwise Template 2
    // Template specific algorithm
    case 1: Template 1 { // Tightly coupled
        $MAP1$ = TEM_Template1($G$, $\breve{G}$, $\hat{E}$, $\hat{H}$); // To be introduced in Section 5.1
        break;
    }
    case 2: Template 2 { // Distributed
        $MAP1$ = TEM_Template2($G$, $\breve{G}$); // To be introduced in Section 5.2
        break;
    }
    Route_Alloc($G$, $\breve{G}$); // To be introduced in Section 5.3
    return $MAP1$;
}
```

5.1 Mapping Algorithm for Template 1: TEM_Template1()

5.1.1 *Algorithm Description.* An application of Template 1 (CTG has at least one hot node) is mapped based on the following observations.

(i) Hot nodes should be given a higher mapping priority; that is, they shall be mapped before any other nodes are mapped. All the hot nodes in a CTG will be first mapped along with their α most significant neighbor vertices. A hot node is better mapped onto a tile in an NoC that has the maximum number of neighbor tiles.

(ii) Once all the hot nodes are mapped, the mapping sequence of remaining unmapped non-hot nodes will be performed based on the decreasing order of weight of edges connecting them.

As such, this procedure consists of two major steps: (i) map hot nodes in CTG, (ii) map other vertices.

```
TEM_Template1 ($G(P, E)$, $\breve{G}(T, L)$, $\hat{E}$, $\hat{H}$)
Input: (1) $G$: CTG of an application
       (2) $\breve{G}$: ACG of an NoC architecture
       (3) $\hat{E}$: A sorted edge list of CTG
       (4) $\hat{H}$: The set of hot nodes
Output: (1) $MAP2$: The table storing the mapping result for each IP core
Function: Map the application of Template 1 onto the NoC architecture
Procedure body:
{
    var: $MAP[1...|P|]$, $MAP1[1...|P|]$, $MAP2[1...|P|]$; // table element storing the tiles allocated for each IP core $p \in P$.
    // Step 1: find and map the hot nodes with their four most significant neighbors
    Set the counter of each vertex to zero;
    for each edge $\hat{e}_i$=($p_k$, $p_j$) $\in \hat{E}$ {
        increase the counter of $p_k$, $p_j$ by 1 if they are hot nodes;
        if one of the vertex's counter equals $\alpha${ // suppose $p_k$'s counter is $\alpha$
            if ($p_k$ is the first hot node and each of $p_k$'s neighbor is also a hot node) {
                $MAP[p_k]$ = tile <$\lfloor X/2 \rfloor$, $\lfloor Y/2 \rfloor$> // $X$ and $Y$ are the row and column numbers of mesh
            }
        }
            $MAP1$ = Map_Hot_Node($p_k$, $\hat{E}$, $T$, $MAP$);
        }
    }

    // Step 2: map the remaining unmapped vertices
    update the hop counts of edges whose terminal vertices are already mapped;
    // rescan the edge list
    for each edge $\hat{e}_i$ = ($p_k$, $p_j$) $\in \hat{E}$ {
        $MAP2$ = Map_Edge($\hat{e}_i$, $\hat{E}$, $T$, $MAP1$);    // $T$ is the available tile set
    }
    return $MAP2$;
}
```

Before we discuss the algorithm in detail, the following definitions are introduced:

Definition 6. Vertex p_n is a significant neighbor of vertex p_i if there exists an edge $(p_i, p_n) \in E$ and such an edge (p_i, p_n) is within the first β edges in the sorted edge list $\hat{E}$.

Definition 7. Since in a mesh structure, the maximal degree of a tile is four, the four most significant neighbors $\{p_{n1}, \ldots, p_{n4}\}$ of a hot node p_i are the four neighbors of p_i that edges $(p_{n1}, p_i), \ldots, (p_{n4}, p_i)$ have the highest bandwidth/tightest latency requirements among all the edges formed by p_i and its other neighbors.

Definition 8. A *center tile* is an unmapped tile that its four neighbor tiles are also unmapped. A center tile in a 2D topology such as a mesh can be determined by the following algorithm. This algorithm searches along the tiles and stops if an unmapped tile is found whose four neighbors are unmapped. All tiles in the NoC architecture are described by a table *tbl* = $\{<X_0, Y_0>, MN_0), \ldots, (<X_{|T|-1}, Y_{|T|-1}>, MN_{|T|-1})\}$, where for tile t_i, $<X_i, Y_i>$ represents its coordinate and a flag, MN_i, indicates whether t_i is mapped or not.

Figure 4 shows an example of the center tiles found in a 5 × 5 mesh. In a simple term, a center tile refers to a tile with the largest degree in the network (e.g., a tile connected to four neighbor tiles in a mesh-based structure).

```
Find_Center_Tile (tbl)
Input: (1) tbl: A table describing all tiles with each element containing a tile's coordinates and a flag MN,
        tbl = {(<X_0, Y_0>, MN_0), …, (<X_{|T|-1}, Y_{|T|-1}>, MN_{|T|-1})}
Output: (1) t: A free center tile
Function: Find a free center tile
Procedure body:
{
    var: t;// representing a tile
    for each element of tbl (<X_i, Y_i>,MN_i){
        if (MN_i is 0 and tile <X_i, Y_i> has a degree of 4) { // tile with coordinate <X_i, Y_i> is not mapped
            if (all the four neighbor tiles <X_{i+1}, Y_i>, <X_{i-1}, Y_i>, <X_i, Y_{i+1}> and <X_i, Y_{i-1}> are not mapped) {
            // all the neighbors are free
                set the four neighbor tiles' MN values to 1;
                t = tile <X_i, Y_i>; // <X_i, Y_i>is a free center tile
                return t;
            }
        }
    }
}
```

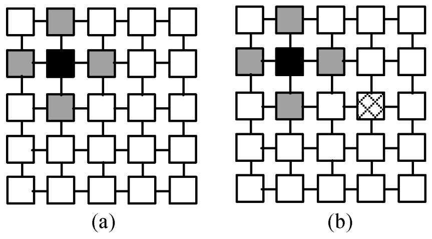

Fig. 4. (a) A center tile found (denoted as black dots) with four neighbor tiles (denoted as gray dots). (b) A second center tile found (denoted as cross dots).

Definition 9. Two vertices p_k, p_j are "close" if (i) edge (p_k, p_j) is among the first β edges in the sorted edge list or (ii) a neighbor vertex of p_k (or p_j) is connected with p_j (or p_k), and this edge is among the first β edges in the edge list.

Step 1: Map Hot Node. Procedure Map_Hot_Node maps each hot node and its four most significant neighbor vertices. Each hot node p_i is associated with a counter ctr_i that counts the number of times p_i appears as a terminal vertex of the edges in the sorted edge list. The procedure works as follows.

—*Case* 1. The current hot node p_i is not mapped yet. Check whether this hot node is *close* (Definition 9) to any of those mapped hot nodes. If yes, this hot node is mapped to a tile with a minimum hop count to the already mapped hot node. Otherwise, a center tile is selected and it is allocated to this hot node. Then, p_i's four most significant neighbors are mapped to the neighbor tiles if they have not been mapped. Two criteria should be observed in mapping the neighbors of p_i: (i) if a neighbor has a high degree value, it should be mapped onto a tile with more available neighbor tiles; (ii) for a neighbor p_n, if the weight of edge (p_n, p_i) is larger than that of edge (p_m, p_i) (p_m represents any other neighbor), p_n should be placed on a tile having lower hop count to the tile of p_i. Tile_Min_Hop_Count(p_i,T) returns a tile in tile set T with minimum hop count to the tile allocated for p_i.

—*Case* 2. The current hot node has already been mapped because it belongs to the first α neighbors of a previously mapped hot node. In this case, only the four most significant neighbor vertices need to be mapped. The procedure Improve_Edge can be called for optimization.

```
Map_Hot_Node($p_{hi}$, $\hat{E}$, $T$, $MAP$)
Input: (1) $p_{hi}$: A hot node as defined in Section 4.2
       (2) $\hat{E}$: The sorted edge list
       (3) $T$: The available tile set in NoC architecture
       (4) $MAP$: The table storing the mapping result for each IP core
Output: (1) $MAP1$: The table storing the mapping result for each IP core after hot nodes are mapped
Function: Map hot node $p_{hi}$ and its neighbors
Procedure body:
{
   var: $MAP[1...|P|]$, $MAP1[1...|P|]$ ; // the table element storing the tiles allocated for each IP core $p \in P$.
   // Case 1
   if ($p_{hi}$ is unmapped){
      // 1) Map the hot node $p_{hi}$
      check whether other mapped hot nodes are close to $p_{hi}$;   // Definition 9
      if (there is a mapped hot node $p_{hk}$ close to hot node $p_{hi}$){
         $MAP[p_{hi}]$ = Tile_Min_Hop_Count($p_{hk}$, $T$);
      }
      else{
         $MAP[p_{hi}]$= Find_Center_Tile($tbl$); // let $tbl$ be the table describing each tile $t_i$ with element (<$X_i$, $Y_i$>, $MN_i$)
      }
      // 2) Map the four most significant neighbors
      for each of $p_{hi}$'s four most significant neighbors { // suppose $p_n$ is one of such neighbors
         if ($p_n$ is unmapped){
            // Two criteria should be observed in mapping neighbors as described in text
            $MAP[p_n]$ = Tile_Min_Hop_Count($p_{hi}$, $T$);
         }
      }
      $MAP1 = MAP$;
   }
   // Case 2
   else if ($p_{hi}$ is mapped) {
      for each of $p_{hi}$'s four most significant neighbors {// Map the four most significant neighbors
         // suppose $p_n$ is one of such neighbors
         if ($p_n$ is mapped) {
            $MAP1$ = Improve_Edge(($p_{hi}$, $p_n$), $\hat{E}$, $T$, $MAP$);   // Optional. ($p_{hi}$, $p_n$) is an edge
            $MAP = MAP1$;
         }
         else{
            $MAP[p_n]$ = Tile_Min_Hop_Count($p_{hi}$, $T$);
            $MAP1 = MAP$;
         }
      }
   }
   return $MAP1$;
}
```

Step 2: Mapping other vertices. After Step 1, all the hot nodes have been mapped. In this step, procedure Map_Edge maps all the remaining unmapped vertex/vertices. There are three cases to consider.

—*Case* 1. Neither of the two terminal vertices p_k, p_j of edge $\hat{e}_i = (p_k, p_j)$ is mapped. Search the edges from$\hat{e}_{i+1}$ to $\hat{e}_{|E|}$ (i.e., edges whose weight is less than $e_{\pi(i)}$). If one of the two terminal's neighbor vertices is found mapped, the vertex with a mapped neighbor is mapped onto a tile that has the minimum hop count to its mapped neighbor. For example, if p_j has a mapped neighbor p_s, then p_j is mapped to a tile with minimum hop count to p_s. Then, the other unmapped terminal vertex p_k in this example is mapped onto a tile with a minimum hop count to p_j. On the other hand, if none of the two terminal vertices' neighbors is mapped, an available tile is selected and immediately allocated to one of the vertices, after which the other vertex is mapped onto a tile with a minimum hop count to the just mapped tile.

—*Case* 2. One of the two terminal vertices is mapped, but the other one is not. In this case, only the unmapped vertex needs to be mapped onto a tile with minimum hop count to the mapped tile.

After two vertices have been mapped, the Improve_Edge procedure is called for optimization.

```
Map_Edge($\hat{e}_i$, $\hat{E}$, $T$, $MAP$)
Input: (1) $\hat{e}_i$: An edge to be mapped
       (2) $\hat{E}$: The sorted edge list of the CTG or a block (Section 5.2)
       (3) $T$: The available tile set in the ACG or a region (Section 5.2)
       (4) $MAP$: The table storing the mapping result for each IP core
Output: (1) $MAP1$: The table storing the mapping result for each IP core after en edge is mapped
Function: Map the terminal vertices of edge $e_{\pi(i)}$ to available tiles
Procedure body:
{
   var: $MAP[1 \dots |P|]$, $MAP1[1 \dots |P|]$, $MAP2[1 \dots |P|]$;
   // table element storing the tiles allocated for each IP core $p \in P$.
   // $\hat{e}_i=(p_k, p_j)$
   // Case 1
   if ($p_k$ and $p_j$ are unmapped) {
      scan the edges list from $\hat{e}_{i+1}$ to $\hat{e}_{|E|}$;
      // suppose one of $p_k$'s neighbor is found, i.e. $\hat{e}_m=(p_s, p_k)$, $m \in [i+1..|\hat{E}|]$
      if ($p_k$'s neighbor $p_s$ is found and it is already mapped)
         $MAP[p_k]$ = Tile_Min_Hop_Count($p_s$, $T$);
      }
      if (none of the neighbors of $p_k$, $p_j$ is mapped) {
         // find a tile with more available neighbor tiles if $p_k$'s degree is 2 or 3
         $MAP[p_k]$ = an available tile in $T$;
      }
   }
   // Case 2
   if (one of the two vertices is not mapped) {
      // suppose that $p_k$ is mapped and $p_j$ is unmapped
      $MAP[p_j]$ = Tile_Min_Hop_Count($p_k$, $T$) ;
   }

   if (both of the two vertices are mapped) {
      $MAP1$ = Improve_Edge($\hat{e}_i$, $\hat{E}$, $T$, $MAP$); // optional
   }
   $MAP1 = MAP;$
   return $MAP1$;
}
```

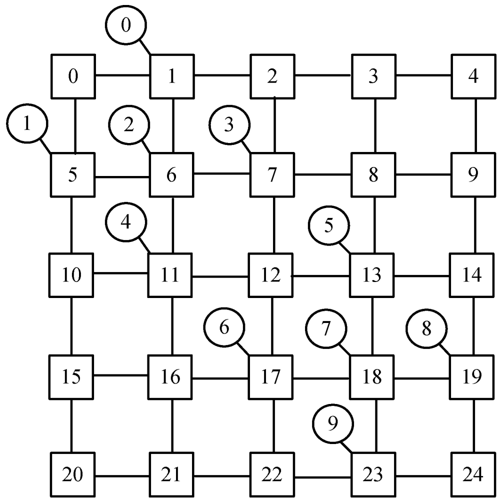

Fig. 5. An example which needs the procedure Improve_Edge.

After completing the mapping process described earlier, it may be necessary to optimize the mapping outcome. This can be seen from an example illustrated in Figure 5, where vertices 2 and 7 are assumed to be the hot nodes. In the first step, these two hot nodes are mapped along with their respective four most significant neighbors. In the second step, when mapping the remaining edges, edge (3, 8) is the edge with the greatest weight. In Figure 5, edge (3, 8) has a delay of 4 hops. For a better mapping, vertices 3 and 8 should be mapped onto tiles with lower hop count apart. This optimization can be achieved by running the Improve_Edge procedure after the mapping.

The Improve_Edge procedure can be made flexible in implementation with different runtime implications. For example, for an edge (p_k, p_j), the following options can be chosen: (i) A free tile is allocated for p_j to check whether the hop count between mapped tiles for p_k and p_j can be reduced. (ii) In a more aggressive way, the tile of a mapped vertex p_m is reallocated to p_j and another tile is found for p_m. Then, the hop count for the related edges needs to be updated. This optimization can be performed on edges with larger weight only.

For edge $\hat{e}_i = (p_k, p_j)$, if the hop count between the mapped tiles of p_k and p_j is greater than 1, the optimization is performed as follows. A parameter *OPT_HOP* is used to control the runtime and the quality of solution. If an available free tile with *OPT_HOP* hop count to p_k's tile is found, it is reallocated to p_j if the following two conditions are satisfied: (i) The hop count between the mapped tiles of p_k and p_j is reduced and (ii) the hop counts between the mapped tiles of the two vertices of the edges from 0 to $i - 1$ in the sorted edge list do not increase. Here, we set *OPT_HOP* to 1.

5.1.2 *Complexity Analysis.* The sorting procedure, Sort_Edge, has a time complexity of $O(|E|\log|E|)$. Thus, the complexity of the TEM_Template1 algorithm is determined by procedures Map_Hot_Node and Map_Edges. Both procedures use Improve_Edge. If an edge has a hop count greater than 1, one free tile with minimum hop count to the tile p_k' mapped to is found. The hop counts of edges $e_{\pi(i)}$ and $e_{\pi(1)}$, $e_{\pi(2)}$, ..., $e_{\pi(i)-1}$ are checked. Hence, the search time of Improve_Edge is bounded to $O(|T|+|E|)$.

Map_Hot_Node: Assume there are $|\hat{H}|$ hot nodes to be mapped. To map each hot node, we need (i) find a center tile which takes $O(|T|)$ time, or (ii) find a tile with minimum hop count to a mapped hot node, which also takes $O(|T|)$ time. After a free tile is found and the hot node is mapped, its four most significant neighbors should be mapped onto the uncommitted tiles with minimum hop counts to the hot node, which has the time complexity of max($|T|$, time(Improve_edges). Hence, the complexity of Map_Hot_Node is $O(|\hat{H}|^*(|T|+\max(|T|,\ |T|+|E|)))$. The average runtime is much less as finding a free tile that is two-hop-count away from a given tile requires a search of no more than 8 tiles.

ii) Map_Edges: There are no more than $|E|$ edges to be considered. For each edge, there are three cases.

Case 1: If the two vertices are unmapped, the edge list needs to be scanned to find an edge such that one of its terminal vertices happens to be the two unmapped vertices. The total search time is $O(|E|)$.

Case 2: One of the vertices is mapped, and the other vertex is unmapped. The search time is $O(|T|)$.

Case 3: If both of the vertices are mapped, Improve_Edge. The time is $O(|T|+|E|)$.

Hence, the time complexity of Map_Edges is $O(|E|^*(|E|+|T|+\ |T|+|E|))$.

Combining the complexity of Map_Hot_Node and Map_Edges, the complexity of TEM_Template1 is shown to be $O(|\hat{H}|^*(|E|+|T|)+|E|^*(|E|+|T|))$, which can be further simplified as $O((|E|+|T|)^2)$, since $|\hat{H}|\leq|P|\leq|T|$.

5.1.3 *An Example.* Here, we use an example to illustrate the TEM_Template1 algorithm. Assume a MPEG4 decoder (its CTG is given in Figure 2) needs to be mapped onto a 4 × 4 mesh network (Figure 6(a)) under latency constraints. The sorted edge list according to the weights of edges in decreasing order is given by:{(1, 2), (1, 3), (1, 4), (8, 4), (1, 7), (8, 11), (8, 10), (8, 9), (1, 0), (1, 5), (5, 6), (6, 7), (1, 9)}. The hot nodes are vertices 1 and 8.

The edge list is scanned. After the fifth edge (1, 7) is reached, the counter of vertex 1 has already reached four. Then vertex 1 is mapped to a center tile in NoC, tile 5. Among the four neighbors, vertices 4 and 7 have the highest degree value. Hence, they are mapped to tiles 9 and 6, respectively, which have more neighbor tiles than tiles 2 and 3. Figure 6(a) shows the result after vertex 1 and its four most significant neighbors are mapped.

Of the first nine edges in the edge list (i.e., up to edge (8, 9)), vertex 8 has appeared four times. Vertex 8 is close to vertex 1 as they are both connected to vertex 4, then it is mapped to tile 10, a tile with the minimum hop count to tiles 5 and 9. Now, tile 10 has only two free neighbor tiles while vertex 8 has three unmapped neighbors. Hence, its neighbors with which vertex 8 forms edges with larger weights among all its neighbors (i.e., vertices 10, 11) are mapped to the neighbor tiles of tile 10. After vertex 9 is mapped to tile 2, which is two-hop away from tile 10, the four most significant neighbors of vertex 8 are

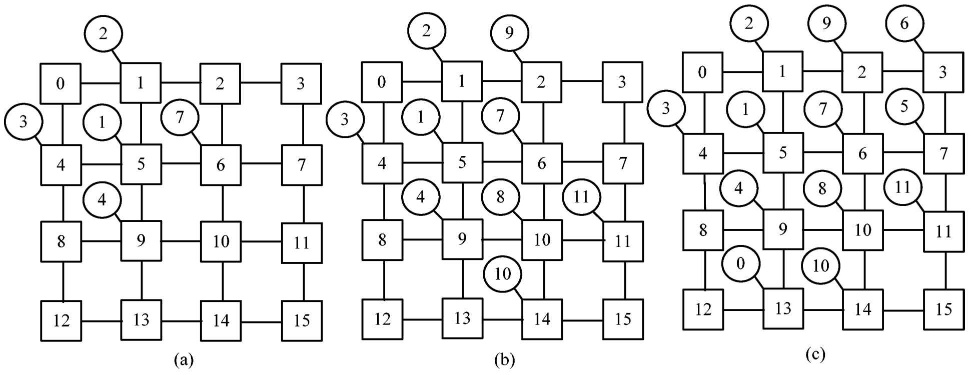

Fig. 6. (a) Result after mapping vertex 1 and its four neighbors. (b) Result after mapping vertex 8 and its four neighbors. (c) Final mapping result of the MPEG4 decoder.

mapped. Figure 6(b) shows the result after mapping vertex 8 and its four most significant neighbors.

After vertices 1 and 8 along with their respective neighbors are mapped, the edge list is rescanned to map the remaining edges, that is, {(1, 0), (1, 5), (5, 6), (6, 7), (1, 9)}. For edge (1, 0), vertex 1 is mapped, but vertex 0 is not. Then, vertex 0 is mapped onto tile 13, a tile with two-hop away from vertex 1 and then. then edge (1, 5) needs to be mapped. As vertex 1 is mapped and vertex 5 is unmapped, tile 7 is selected for mapping vertex 5. The remaining edges are mapped similarly. The final mapping result is shown in Figure 6(c).

5.2 Mapping Algorithm for Template 2: TEM_Template2()

Mapping applications of Template 2: (CTG does not contain hot nodes) is based on the following observation:

If a CTG does not have such hot nodes, instead of recursively partitioning the CTG and ACG as in MOCA [Srinivasan and Chatha 2005], the CTG and the ACG can be both partitioned into only four parts and mapped in a divide-and-conquer–like manner. For each part of the CTG/ACG, the mapping is performed with larger weight first criterion.

5.2.1 *Algorithm Description*

Definition 10. A *block trace graph* (BTG) is denoted as $G' = (B, BE)$, where each vertex $b_i \in B$ is a partitioned block and $b_i \subset P$ (P is the vertex set of CTG(P, E)), and an edge $be_i = (b_k, b_j) \in BE$ exits if $p_{kl} \in b_k$ and $p_{jm} \in b_j$, $(p_{kl}, p_{jm}) \in E$. In Figure 7(a), the circles are blocks. For an edge $be_i = (b_k, b_j) \in BE$,

—$\omega(be_i)$ represents the total bandwidth request of the CTG edges between the vertices in b_k and b_j.

—$\sigma(be_i)$ represents the tightest latency constraint among all edges (p_{kl}, p_{jm}) for $\forall p_{kl} \in b_k$ and $\forall p_{jm} \in b_j$. $\sigma(be_i)$ is represented in the number of hops among the CTG edges between vertices in b_k and b_j.

—$W(be_i)$ represents the weight of be_i. The calculation is the same as the weight of edges in CTG.

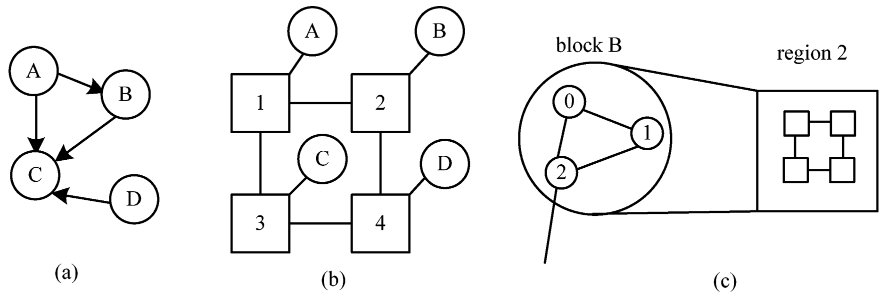

Fig. 7. (a) An example BTG. (b) Mapping result of the BTG to regions in a VACG. (c) Mapping of vertices inside each block to the tiles in the corresponding region in VACG.

—An CTG edge $(p_i, p_j) \in E$ belongs to a block b_k if $p_i \in b_k$ or $p_j \in b_k$. E_i represents the set of edges belonging to b_i.

— $|b_k|$ represents the number of CTG vertices in b_k (the size of b_k).

The vertices in each block b_i are categorized into two types, and they are mapped differently.

(i) *Internal vertices*. An internal vertex p_{intern} has all its neighbors in the same block of p_{intern}. In Figure 7(c), vertices 0 and 1 are internal vertices. The internal vertices set of b_i is represented as IN_i.

(ii) *External vertices*. An external vertex p_{extern} has at least one of its neighbors not in the same block of p_{extern}. In Figure 7(c), vertex 2 is an external vertex. The external vertices set of b_i is represented as EX_i.

Definition 11. A *Virtual ACG* (VACG) is denoted as $\breve{G}' = (R, CH)$, where a vertex $r_i \in R$ represents a partitioned region and $r_i \subseteq T$ (T is the tile set of ACG(T, L)). Each edge $ch_i = (r_k, r_j) \in CH$ represents the existence of direct links between tiles in r_k and r_j. In Figure 7(b), each square box is region to be mapped. For each edge $ch_i \in CH$,

—$bw(ch_i)$ represents the total bandwidth provided by the ACG links between all the tiles in r_k and r_j.

—$c(ch_i)$ represents the cost of edge ch_i, which is calculated as the average power per bit of the ACG links between all the tiles in r_k and r_j.

The Template 2 TEM algorithm has two major steps. In the first step, the BTG is formed using a graph partition algorithm [Hendrickson and Leland 1995] to partition the CTG into four blocks. The VACG is formed by partitioning the ACG into four default regions (e.g., square regions in Figure 7(b)). The blocks in the BTG are mapped to the regions in the VACG. Then, the actual regions of the VACG are partitioned according to the size of the block in the BTG. Figures 7(a) and 7(b) illustrate this process. Blocks A, B, C, D are mapped to regions 1, 2, 3, 4, respectively.

In the second step, the vertices in each block are mapped to the tiles within its region. The external vertices are first mapped to the border of each region, after which the internal vertices are mapped with larger weight edge first criterion. Figure 7(c) illustrates this process. The edges in a block are sorted in a nonincreasing order using Sort_Edge. For b_i, let the sorted list be $\hat{E}_i = \hat{e}_1, \hat{e}_2, \ldots, \hat{e}_{|Ei|}$, i.e., $W(\hat{e}_1) \geq W(\hat{e}_2) \geq \cdots \geq W(\hat{e}_{|Ei|})$ for $\hat{e}_i \in E_i$, $1 \leq i \leq |E_i|$.

The following text lists the TEM_Template2 algorithm, which calls the Map_Edge and Improve_Edge procedures described in Section 5.1. Based on the mapping result of BTG onto VACG, the VACG is partitioned into regions in a top-down manner. The size of each partitioned region needs to be set equal to the size of the corresponding block in the BTG.

```
TEM_Template2($G(P,E)$, $\breve{G}(T,L)$)
Input: (1) $G$: The CTG of an application.
       (2) $\breve{G}$: The ACG of an NoC architecture
Output: (1) $MAP2$: The table storing the mapping result for each IP core
Function: Map the Template 2 application onto the NoC architecture
Procedure body:
{
    var: $MAP[1...|P|]$, $MAP1[1...|P|]$, $MAP2[1...|P|]$ ;// tables storing the tiles allocated for each IP core $p \in P$.
    // Step 1. Obtain the BTG and VACG by partitioning the CTG into blocks and ACG into default
    // regions, respectively. Map the blocks in BTG to the regions in VACG.
    obtain BTG ($G'$) using the graph partition algorithm to partition CTG into four blocks;
    obtain VACG ($\breve{G}'$) with default four square regions;
    map the four blocks $b_{r1}$, $b_{r2}$, $b_{r3}$, $b_{r4}$ in BTG to four default regions of VACG $r_1$, $r_2$, $r_3$, $r_4$ using Map_Edge;
    // partition VACG
    1. partition the tiles of ACG horizontally into two parts, $P_1$, $P_2$,
       where $P_1$'s size fits size($b_{r1}$)+size($b_{r2}$) and $P_2$'s size fits size($b_{r3}$)+size($b_{r4}$);
    2. partition tiles in $P_1$ into $P_{11}$ and $P_{12}$ vertically, where $P_{11}$'s size fits size($b_{r1}$) and $P_{12}$'s size fits size($b_{r2}$);
    3. partition tiles in $P_2$ into $P_{21}$ and $P_{22}$ vertically, where $P_{21}$'s size fits size($b_{r3}$) and $P_{22}$'s size fits size($b_{r4}$) ;
    4. corresponds the four parts with the four regions (i.e., $P_{11}$ to $r_1$, $P_{12}$ to $r_2$, $P_{21}$ to $r_3$, $P_{22}$ to $r_4$);

    // Step 2. Inside each block, map the vertices to the tiles of the corresponding region.
    for each block $b_i$ in BTG mapped to region $r_x$ in VACG {
        // 1) Map the external vertices. Let $PE_i$ be set of external vertices in $b_i$;
        sort external vertices in $PE_i$ in the non-increasing order of the total weight of the external vertices.
        Let $\hat{p}_1$, $\hat{p}_2$, ..., $\hat{p}_{|EX_i|}$ be the sorted list;

        for each external vertex $\hat{p}_i$ {

            $MAP1$ = Map_External_Vertex( $\hat{p}_i$, $G'$, $\breve{G}'$, $MAP$);

        }
        // 2) Map the internal vertices
        sort the edges in $b_i$ into $\hat{E}_i$ using Sort_Edge;
        for each edge $\hat{e}_i = (p_k, p_j) \in \hat{E}_i$ {
            $MAP2$ = Map_Edge ($\hat{e}_i$, $b_i$, $r_x$, $MAP1$); // described in Section 5.1
        }
    }
    return $MAP2$;
}
```

Figure 8 shows an example. Assume the size of b_{r1}, b_{r2}, b_{r3}, b_{r4} is 3, 1, 4, 4, respectively. Then, the first partition generates P_1 with tiles 0, 1, 2, 3, and P_2 with the rest tiles. Then, according to the sizes of the four blocks, P_1 and P_2 are partitioned vertically. The final partitioned VACG regions are shown in Figure 8.

The Map_External_Vertex_Block procedure maps extern vertices to the border of each block.

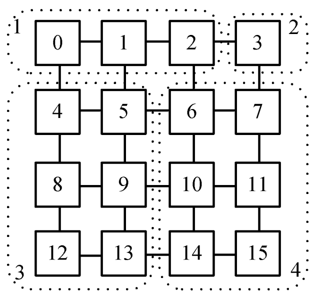

Fig. 8. An example of partitioning NoC into regions of VACG according to the blocks of CTG.

```
Map_External_Vertex($p_{extern}$, $G'(B, BE)$, $\hat{G}'(R, CH)$, $MAP$)
Input: (1) $p_{extern}$: An external vertex to be mapped which belongs to block $b_i$
       (2) $G'$: The BTG of an application (Definition 10)
       (3) $\hat{G}'$: The VACG of the NoC architecture (Definition 11)
       (4) $MAP$: The table storing the mapping result for each IP core
Output: (1) $MAP1$: The table storing the mapping result for each IP core after an external vertex is mapped
Function: Map the external vertex $p_{extern}$ to an available tile
Procedure body:
{
    var: $MAP[1...|P|]$, $MAP1[1...|P|]$;// tables storing the tiles allocated for each IP core $p \in P$.
    // suppose $p_{extern} \in b_i$
    if one of $p_{extern}$'s neighbor $p_n \in b_j$ is mapped {
        find an available tile $t_k \in r_x$ with minimum hop count to $p_n$'s tile;
        $MAP[p_{extern}] = t_k$;
    }
    else { //none of $p_{extern}$'s neighbors outside of $b_i$ is mapped
        if $p_{extern}$ has one neighbor outside of $b_i$ {
            $MAP[p_{extern}]$ =
                an available tile located at the border to the region that $p_{extern}$ neighbor's block is mapped to;
        }
        else {
            find the two neighbors with the highest weight;
            $MAP[p_{extern}]$ =
                an available tile located at the border to the regions that its two neighbor's blocks are mapped to ;
        }
    }
    $MAP1 = MAP$;
    return $MAP1$;
}
```

Figure 9 illustrates how an external vertex, p_{extern}, is mapped if none of its neighbors outside of its block has been mapped yet. Assume p_{extern}'s block is mapped to region 1. If p_{extern} only has one neighbor outside of its block and its neighbor's block is mapped to region 2, p_{extern} shall be mapped to a tile in the shaded area in Figure 9(a). If its neighbor's block is mapped to region 3, then p_{extern} shall be mapped to the shaded area in Figure 9(b). Similarly, if its neighbor's block is mapped to region 4, p_{extern} shall be mapped to a tile in the shaded area in Figure 9(c).

5.2.2 *Complexity Analysis.* The complexity of TEM_Template2 is determined by the step where the vertices in each block are mapped to the tiles within its region.

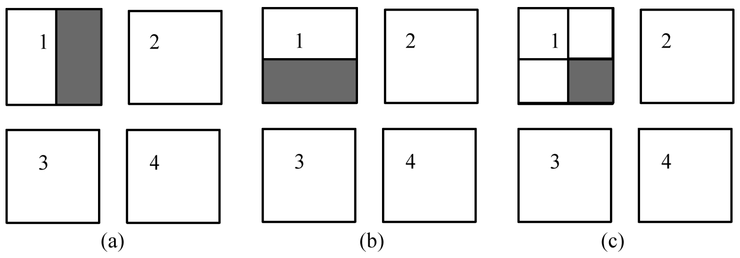

Fig. 9. Illustration of mapping an external vertex. The shaded area in region 1 is the border between (a) region 2, (b) region 3 and (c) region 4.

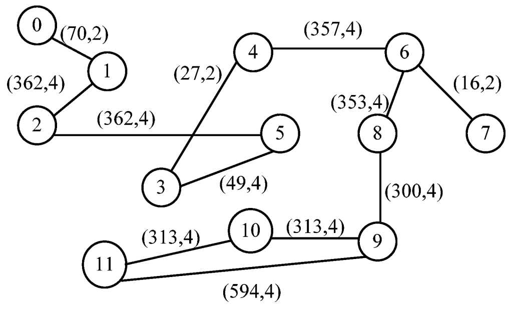

Fig. 10. CTG of VOPD.

For each block, the external vertices are found and mapped first. This requires scanning the edges in this block to find the external vertices, which requires $|E_i|$ time. For each external vertex, find a tile for it, which takes $O(|T_i|)$ time (T_i is the tile in the region which b_i is mapped to). Then, the internal vertices are mapped after sorting the edges that belong to this block. The complexity is bounded by the complexity of Map_Edge for the edges inside a block. The runtime is

$$|E_i|+|b_i|\cdot log(|b_i|)+|b_i|\cdot(|E_i|+|T_i|)+|E_i|\cdot\log(|E_i|)+|E_i|^2+2\cdot|E_i|\cdot|T_i|+O(|T|+|E|).$$

Hence, the overall complexity of TEM_Template2 is bounded to $O((|E|+|T|)^2)$ as $|b_i|<|P|\leq|T|$, $|E_i|<|E|$, $|T_i|<|T|$.

5.2.3 *An Example.* Here we use the VOPD with latency constraint [Srinivasan and Chatha 2005] shown in Figure 10 to illustrate how the TEM_Template2 algorithm works. The tile ordering is the same as shown in Figure 6(a).

The sorted edge list is given by {(0, 1), (3, 4), (6, 7), (9, 11), (10, 11), (1, 2), (2, 5), (4, 6), (6, 8), (9, 10), (8, 9), (3, 5)}. The four blocks after the partition are: A{0, 1}, B{2, 5}, C{3, 4, 6, 7}, D{8, 9, 10, 11}. The mapping of the blocks onto four default regions is shown in Figure 11. The partitioned VACG regions are shown in Figure 12.

To save space, only the mapping inside block D is explained in detail. Figure 13(c) shows the mapping result before the vertices in block D are mapped. The edge list in block D is {(9, 11), (10, 11), (6, 8), (9, 10), (8, 9)}

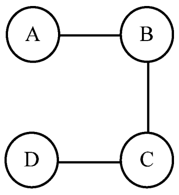

Fig. 11. Mapping blocks to the default regions in VACG.

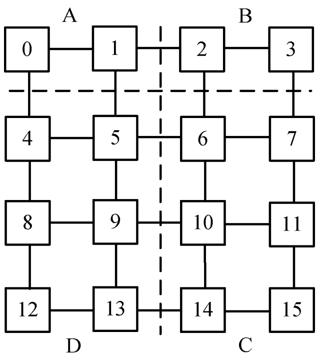

Fig. 12. Partition of a NoC into regions.

and the external vertex is 8, then vertex 8 is mapped to tile 9, a tile sitting at the border of regions C and D and adjacent to vertex 6. Now both of the two vertices of edge (9, 11) are unmapped. Then, the edge list is searched again and edge (8, 9) is found. Since vertex 8 is already mapped, vertex 9 is mapped to a tile adjacent to vertex 8 and vertex 11 is mapped to a tile adjacent to vertex 9. Next, for edge (10, 11), vertex 10 is mapped adjacent to vertex 11. Figure 13(d) shows the result after all the vertices in block D are mapped.

5.3 Routing Path Allocation

After the mapping step, the minimum hop count between any mapped tile pair is calculated, and this calculation and the bandwidth constraint shall guide the routing algorithm to find the routing paths. In an NoC design, the deterministic and static routing is preferred due to its simplicity and less resources needed [Hu and Marculescu 2005]. In TEM, a similar routing algorithm as in Hu and Marculescu [2005] to find the routing paths for each communication edge with two objectives: (i) all the routing paths have to be minimal and the resulting network has to be deadlock-free, and (ii) the bandwidth and latency constraints need to be satisfied.

To ensure deadlock-free routing, the turns are restricted to the legal turn set (LTS) [Hu and Marculescu 2005] allowed by the underline routing algorithm. An LTS is composed of and only of those turns allowed in the corresponding algorithm. Any path to be allocated can only employ turns from the LTS. A *legal path* is a minimal path that employs the turns in LTS. Different from Hu and Marculescu [2005], the flexibility of an edge is subject to the latency constraint (i.e., an edge with tighter latency shall be made less flexible). Here,

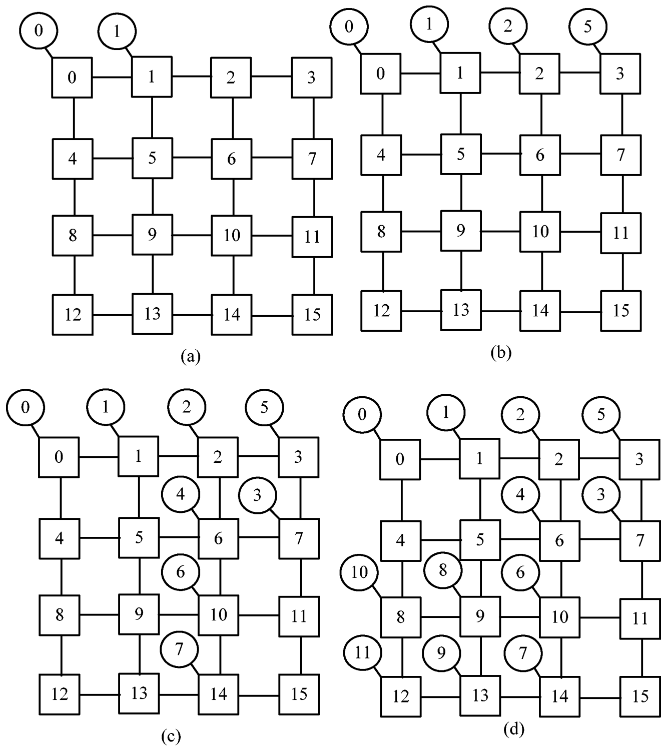

Fig. 13. Mapping results after block A is mapped (a), block B is mapped (b), block C is mapped (c), and block D is mapped (d).

the flexibility of an edge $e_{\theta(i)}$, flex($e_{\theta(i)}$), is defined as $flex(e_{\theta(i)}) = \delta(h_{M(p_k),M(p_j)} \times \sigma(e_{\theta(i)})^K$, and

$$\delta(H(M(p_k), M(p_j)) = \begin{cases} 2, & \textit{if there is more than one legal path} \\ 1, & \textit{if there is only one legal path} \end{cases}$$

[Duato et al. 2003], where H and σ, K are defined in Section 3. Edges with a lower degree of flexibility are given a higher priority during the path allocation.

6. PERFORMANCE EVALUATION

To evaluate the performance of the TEM algorithm, both TEM_Template1 and TEM_Template2 algorithms are implemented and simulated. The brand-and-bound (BNB) [Hu and Marculescu 2005] and MOCA [Srinivasan and Chatha 2005] algorithms are also implemented and the mapping results of all three algorithms are compared. The size of the work queue is properly selected for a practical branch-and-bound algorithm [Lin et al. 2008; Murali and De Micheli 2004]. If the length of work queue is not limited, the solution of BNB is nearly optimized. However, when the network size increases, both the runtime and the memory consumption of BNB increase dramatically. Controlling the length of work queue attempts to make a compromise between the quality of solution and the runtime of BNB. To demonstrate the effects of different routing

Table I. Numbers of Vertices of Random Benchmarks for Different Network Sizes

Network size	Numbers of vertices of benchmarks
4×4	13, 14, 15, 16
5×5	23, 24, 24, 25
6×6	33, 34, 35, 36

Table II. Comparison of Power Consumption (Normalized) of TEM and MOCA on Mesh-Based NoCs with Different Network Sizes

	Applications of Template 1			Applications of Template 2		
	TEM			TEM		
Network size	XY	Odd–even	MOCA	XY	Odd–even	MOCA
4×4	0.94	0.88	1	0.95	0.89	1
5×5	0.93	0.87	1	0.94	0.92	1
6×6	0.86	0.84	1	0.87	0.83	1

algorithms, the XY routing and odd–even routing (one adaptive routing algorithm) algorithms are selected. Two types of applications, random applications generated from task graph for free (TGFF) [Dick et al. 1998] and multimedia applications from Hu and Marculescu [2003] and Srinivasan and Chatha [2005], are adopted as benchmarks. The Noxim simulator [Noxim] is modified to obtain the total communication power after the mapping and the routing path allocation steps. The energy dissipation parameter for routers from Orion [Wang 2002] and the link power dissipation parameter from COSI [Pinto et al. 2009], both based 90 nm CMOS technology, are adopted in our simulations. The simulations are performed on a PC with one Intel Core2 P8600 2.4GHz processor and 2GB RAM.

6.1 Experiments on Random Applications

We first apply TEM and MOCA to map five randomly generated benchmark applications onto 4 × 4, 5 × 5, and 6 × 6 mesh-based NoCs. The numbers of vertices of random benchmarks over these three different sizes of networks are shown in Table I. The results are tabulated in Table II. For the ease of comparison, the power consumption of the mapping results obtained from MOCA and TEM for each benchmark is normalized. For each template on networks with different sizes 4 × 4, 5 × 5, 6 × 6, five random benchmark applications are generated. The number of vertices in each of the five applications is selected from Table I.

Table II shows that that TEM outperforms MOCA with $\geq$5% less power consumption when XY routing is used and with $>$10% less power consumption when adaptive routing is used. For applications of Template 1, the improvement of TEM over MOCA becomes more noticeable with the increase of the network size.

Table III reports the work queue length and the runtimes of the branch-and-bound algorithm and TEM. We set the mapping result of TEM as the seed and adjust the length of work queue in BNB until the mapping result of BNB outperforms that of TEM (the seed), that is, the power consumption of BNB is

Table III. The Work Queue Length and Runtime of BNB When BNB Outperforms TEM in Power Consumption (Normalized)

	BNB		
Network size	Length of work queue	Ratio of runtime of BNB/TEM	Ratio of power consumption of BNB/TEM
4×4	1,000	70.5	0.98
5×5	10,000	606.42	0.93
6×6	100,000	N/A	N/A

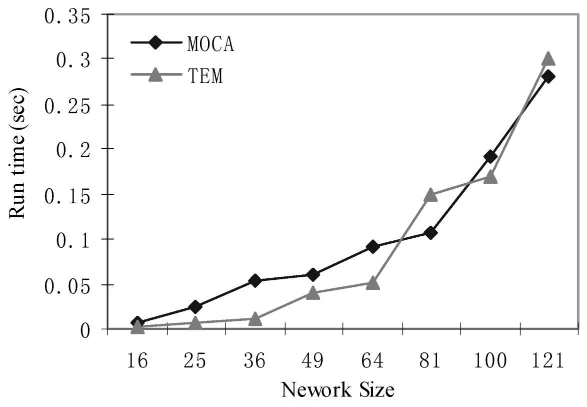

Fig. 14. Runtime vs. network size for TEM and MOCA.

lower than that of TEM. The ratio of BNB's runtime over TEM is defined by the runtime of BNB divided by that of TEM, with given length of work queue. The ratio of power consumption over TEM is defined as the power consumption of BNB divided by that of TEM. The ratio of power consumption over TEM of 6 × 6 networks is not available because BNB with a work queue size of 100,000 cannot generate better mapping results than TEM after running for 3 hours. As also shown in this column, with the increase of network size, in order to outperform TEM, BNB has to run a much longer time, so TEM is much more scalable than BNB with the increase of the network size.

Next, we compare the runtime of MOCA and TEM when network size grows. For each network size, five random applications are generated with the number of vertices equal to the number of tiles. As shown in Figure 14, the average runtimes of MOCA and TEM are comparable.

6.2 Experiments on Multimedia Benchmarks

Multimedia benchmarks of both Template 1 and Template 2 are tested on a 4 × 4 mesh-based NoC. Table IV lists these benchmarks. For benchmarks of both templates, we compare the simulation results of TEM, BNB, and MOCA. Specifically, we compare the degradation of normalized power consumption of TEM compared to BNB (i.e., the increase in power consumption of mapping results from TEM compared to that from BNB) and the degradation of MOCA compared to BNB. Since 4 × 4 mesh is relatively small in size, the length of work queue in BNB is set to unlimited (i.e., no speed-up technique is used). The

Table IV. Multimedia Programs and Their Templates

Benchmark	Template	Description
MPEG4	1	MPEG4 decoder
263enc	1	H263encoder
263enc+MP3enc	1	H263encoder and MP3 encoder
263enc+MP3dec	1	H263encoder and MP3 decoder
263enc+263dec	1	H263encoder and H263 decoder
Multimedia Systems (MMS)	1	Multimedia Systems
VOPD	2	Video plane object decoder
MP3enc	2	MP3 encoder
MP3enc+MP3dec	2	MP3 encoder and MP3 decoder

Table V. Comparison of Power Consumption (Normalized) of TEM and MOCA over BNB on Benchmarks of Template 1

	With latency constraint				Without latency constraint			
	TEM				TEM			
Benchmarks	XY	Odd–even	MOCA	BNB	XY	Odd–even	MOCA	BNB
MPEG4	1.05	1.05	1.45	1	1.01	1.01	1.06	1
263 enc	1.07	1	1.11	1	1.4	1.03	1	1
263enc+MP3enc	1.14	1.11	1.27	1	1.01	1	1	1
263enc+MP3dec	1.06	1.01	1.28	1	1.04	1.01	1	1
263enc+263dec	1.06	1.01	1.9	1	1.06	1.02	1	1

runtime of BNB is in the order of a few minutes depending on the characteristics of the CTG.

Table V shows the result of benchmarks of Template 1 with and without latency constraints. When the latency constraints are not considered, the result of TEM is comparable with that of MOCA. When the latency constraints are considered, the TEM algorithm outperforms MOCA for most media programs listed in Table IV. The degradation of TEM as opposed to BNB is within 10% with odd–even routing. The reduction in power consumption of TEM compared to MOCA is over 15%. For MPEG4dec, the degradation of TEM is only 6% compared to that of MOCA 45%. This is due to that the weight of edges with tight latency constraint, but low bandwidth request is greatly increased and the difference in edge weights is increased. The recursive graph partition of MOCA may separate IP cores having communications with high bandwidth request but loose latency constraint. Thus, edges with higher bandwidth requests are mapped to paths with more hop counts, resulting in higher power consumption. Consequently, the mapping of tightly coupled subgraphs is not optimized.

Table V also shows that the power consumption of mapping result from TEM with odd–even routing is lower than that from TEM with XY routing for 263enc, 263enc + MP3enc, 263enc + MP3enc, 263enc + 263dec with and without latency constraints. This is due to the fact that applications of Template 1 have several hot nodes that request high bandwidth/tight latencies. Adaptive routing is more suitable for this type of applications with unevenly distributed communication patterns.

Table VI. Comparison of Power Consumption (Normalized) of TEM over MOCA on MMS (Template 1)

	With latency constraint			Without latency constraint		
	TEM			TEM		
Benchmarks	XY	Odd–even	MOCA	XY	Odd–even	MOCA
MMS	0.9	0.84	1	0.97	0.91	1

Table VII. Comparison of Power Consumption (Normalized) TEM and MOCA over BNB on Benchmarks of Template 2

	With latency constraint				Without latency constraint			
	TEM				TEM			
Benchmarks	XY	Odd–even	MOCA	BNB	XY	Odd–even	MOCA	BNB
VOPD	1.02	1.02	1.1	1	1.02	1.02	1.08	1
MP3enc	1.1	1.1	1.12	1	1	1	1	1
MP3enc+MP3dec	1.14	1.11	1.41	1	1.08	1.07	1.08	1

The multi-media system (MMS) benchmark [Hu and Marculescu 2005] is also simulated on a 5×5 mesh-based NoC. Table VI shows the reduction in power consumption (normalized) of TEM over MOCA.

Table VII shows the benchmark result (Template 2) with and without latency constraints. Without latency constraints, the result of TEM is comparable with that of MOCA. On average, the degradation of TEM over BNB is within 10% for both XY routing and odd–even routing. With latency constraints, the power consumption of mapping result from TEM is slightly lower than that from MOCA for VOPD and MP3enc. For MP3enc + MP3dec, TEM achieves significant reduction in power consumption compared to MOCA. This is due to that the TEM algorithm for applications of Template 2 uses a divide-and-conquer approach. Inside each partitioned block, TEM first maps external vertices to the border of the block and the remaining edges with larger weight first criterion. Thus, larger weight edges are mapped first inside each block. MOCA, on the other side, recursively partitions the CTG without considering the structure of the network. Thus, some edges with large bandwidth requests but satisfy low latency constraint may be separated and allocated to tiles with higher hop counts.

For benchmarks of Template 2, the power consumption of mapping result from TEM with odd–even routing is slightly lower than that from TEM with XY routing. Since the communication traffic is more evenly distributed in this type of applications, the superiority of odd–even routing over XY routing is not so significant under such traffic as compared to what is seen in Template 1.

7. CONCLUSION AND FUTURE WORK

This article presented a template-based greedy algorithm to address the IP mapping problem under the bandwidth and latency constraints. An application falls into Template 1, if there are one or more hot nodes in the application which have many communications demanding either higher bandwidth or tighter latency. In this case, the proposed algorithm TEM maps the hot nodes first along with their four most significant neighbors, after which the remaining IP cores are mapped in descending order based on the edge weights. On the other hand, an application is categorized as Template 2, if the communications are nearly evenly distributed among the vertices. In this case, TEM first divides the NoC into regions and CTG into blocks, then it maps the IP cores inside each block in a divide-and-conquer manner. The experiments on both random and multimedia benchmarks showed that TEM generates high quality mapping results with low runtimes. Future work includes extension of TEM to other NoC topologies, such as folded torus, fat tree, and a few others.

REFERENCES

Ascia, G., Catania, V., and Palesi, M. 2004. Multi-objective mapping for mesh-based NoC architectures. In *Proceedings of the 2nd International Conference on Hardware/Software Co-Design and System Synthesis*. IEEE, Los Alamitos, CA, 182–187.

Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and De Micheli, G. 2005. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Paral. Distrib. Syst. 16*, 2, 113–129.

Chang, J. M. and Pedram, M. 2000. Codex-dp: Co-design of communicating systems using dynamic programming. *IEEE Trans. Comput. Aid. Des. Integr. Circuits Syst. 19*, 7, 732-744.

Dally, W. J. and Towles, B. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (DAC).* ACM, New York, 684–689.

Dick, R. P., Rhodes, D. L., and Wolf, W. 1998. TGFF: Task graphs for free. In *Proceedings of the 6th International Workshop on Hardware/Software Co-Design*. ACM, New York, 97–101.

Duato, J., Yalamanchili, S., and Ni, L. M. 2003. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, San Francisco, CA.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*WH Freeman, San Francisco, CA.

Hansson, A., Goossens, K., and Radulescu, A. 2005. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proceedings of the 3rd International Conference on Hardware/Software Co-Design and System Synthesis.* IEEE, Los Alamitos, CA, 75–80.

Harmanani, H. M. and Farah, R. 2008. A method for efficient mapping and reliable routing for NoC architectures with minimum bandwidth and area. In *Proceedings of the Conference on Circuits and Systems and TAISA.* IEEE, Los Alamitos, CA, 29–32.

Hendrickson, B. and Leland, R. 1995. *The Chaco User's Guide: Version 2.0.* Sandia National Laboratories, Albuquerque, NM.

Ho, R., Mai, K. W., and Horowitz, M. A. 2001. The future of wires. *Proc. of IEEE. 89*, 4, 490–504.

Hu, J. and Marculescu, R. 2003. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proceedings of the Design Automation Conference*. ACM, New York, 233–239.

Hu, J. and Marculescu, R. 2005. Energy-and performance-aware mapping for regular NoC architectures. *IEEE Trans. Comput. Aid. Des. Integr. Circuits Syst. 24*, 4, 551–562.

ITRS. 2007. International technology roadmap for semiconductors. http://www.itrs.net/Links/2007ITRS/Home2007.htm.

Jantsch, A. and Tenhunen, H. 2003. *Networks on Chip*. Kluwer Academic Publishers, New York.

Kodi, A. K., Sarathy, A., and Louri, A. 2008. Adaptive channel buffers in on-chip interconnection networks: A power and performance analysis. *IEEE Trans. Comput. 57*, 9, 1169–1181.

Lin, H., Li, X., Tong, D. and Cheng, X. 2008. A low energy mapping and routing approach for network on chip with QoS guarantees. *J. Comput. Aid. Des. Comput. Graphics. 20*, 4, 425–431.

Lu, Z., Xia, L., and Jantsch, A. 2008. Cluster-based simulated annealing for mapping cores onto 2D mesh networks on chip. In *Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems.* IEEE, Los Alamitos, CA, 1–6.

Marcon, C. A. M., Moreno, E. I., Calazans, N. L. V. and Moraes, F. G. 2007. Evaluation of algorithms for low energy mapping onto NoCs. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* IEEE, Los Alamitos, CA, 389–392.

Mehran, A., Saeidi, S., Khademzadeh, A., and Afzali-Kusha, A. 2007. Spiral: A heuristic mapping algorithm for network on chip. *IEICE Electron. Express 4*, 15, 478–484.

Meindl, J. D. 2003. Interconnect opportunities for gigascale integration. *IEEE Micro. 23*, 3, 28–35.

Murali, S. and De Micheli, G. 2004. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition.* ACM, New York, 896–901.

Noxim. Network-on-chip simulator. http://sourceforge.net/projects/noxim

Ogras, U. Y., Hu, J., and Marculescu, R. 2005. Key research problems in NoC design: a holistic perspective. In *Proceedings of the 3rd International Conference on Hardware / Software Co-Design and System Synthesis.* IEEE, Los Alamitos, CA, 69–74.

Pinto, A., Carloni, L., and Vincentelli, A. 2009. A methodology for constraint-driven synthesis of on-chip communications. *IEEE Trans. Comput. Aid. Des.* 28, 3, 364–377.

Srinivasan, K. and Chatha, K. S. 2005. A technique for low energy mapping and routing in network-on-chip architectures. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, 387–392.

Wang, H., Zhu, X., Peh, L-S., and Malik, S. 2002. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual International Symposium on Microarchitecture.* IEEE, Los Alamitos, CA, 294–305.

Zhou, W., Zhang, Y., and Mao, Z. 2006. Pareto-based multi-objective mapping IP cores onto NoC architectures. In *Proceedings of the Asia Pacific Conference on Circuits and Systems*. IEEE, Los Alamitos, CA, 331–334.

Received March 2009; revised July 2009; accepted September 2009