Energy- and Performance-Aware Mapping for Regular NoC Architectures

Jingcao Hu, *Student Member, IEEE,* and Radu Marculescu, *Member, IEEE*

***Abstract*—In this paper, we present an algorithm which automatically maps a given set of intellectual property onto a generic regular network-on-chip (NoC) architecture and constructs a deadlock-free deterministic routing function such that the total communication energy is minimized. At the same time, the performance of the resulting communication system is guaranteed to satisfy the specified design constraints through bandwidth reservation. As the main theoretical contribution, we first formulate the problem of energy- and performance-aware mapping in a topological sense, and show how the routing flexibility can be exploited to expand the solution space and improve the solution quality. An efficient branch-and-bound algorithm is then proposed to solve this problem. Experimental results show that the proposed algorithm is very fast, and significant communication energy savings can be achieved. For instance, for a complex video/audio application, 51.7% communication energy savings have been observed, on average, compared to an ad hoc implementation.**

***Index Terms*—Energy, low power, networks-on-chip (NOCs), optimization, performance.**

I. Introduction

With the advance of the semiconductor technology, the huge number of transistors available on a single chip allows designers to integrate tens of intellectual property (IP) blocks together with large amounts of embedded memory. These IPs can be CPU or DSP cores, video stream processors, high-bandwidth input/outputs (I/O) devices, etc. The richness of the computational resources places tremendous demands on the communication resources as well. Additionally, the shrinking feature size in the deep submicron (DSM) era pushes interconnection delay and power consumption as the dominant factors in the optimization of modern systems. Another consequence of the DSM effects is the difficulty in optimizing interconnections because of the ensued worsening effects such as crosstalk, electromagnetic interference (EMI), etc.

Regular tile-based network-on-chip (NoC) architecture was recently proposed to mitigate these complex on-chip communication problems [1], [4], [5]. As shown in the left side of Fig. 1, such a chip consists of a grid of regular tiles where each tile can be a general-purpose processor, a DSP, a memory subsystem, etc. A router is embedded within each tile with the objective of connecting it to its neighboring tiles. Thus, instead of routing design-specific global on-chip wires, the intertile communication can be achieved by routing packets.

Three key concepts come together to make this tile-based architecture very promising: 1) structured network wiring; 2) modularity; and 3) standard interfaces. More precisely, since the network wires are structured and wired beforehand, their electrical parameters can be very well controlled and optimized. In turn, these controlled electrical parameters make it possible to use aggressively signaling circuits that help reduce the power dissipation and propagation delay significantly. Modularity and standard network interfaces facilitate reusability and interoperability of the modules. Moreover, since the network platform can be designed in advance and later reused directly with many applications, it makes sense to highly optimize this platform as its development cost can be easily amortized across many applications.

On the other hand, the regular tile-based architecture may lead to significant area overhead if applied to applications whose IPs' sizes vary significantly. In order to achieve the best performance/cost tradeoff, the designer needs to select the right NoC platform (e.g., the platform with the right size of tiles, routing strategies, buffer sizes, etc.) and further customize it according to the characteristics of the application under design. For most applications, the area cost overhead is fully compensated by the design time savings and performance gains because of the regular NoC architecture. The advantages of using the regular NoC approach can be further increased if the IPs in the library are developed with regularity (in terms of size) taken into consideration as well. Moreover, partitioning the application with regularity in mind can also help in reducing the cost overhead. Finally, the region-based design [5] can be used to further reduce the area overhead by embedding irregular regions inside the NoC, which can be insulated from the network.

From the design perspective, given a target application described as a set of concurrent tasks which have been assigned and scheduled, to exploit the architecture in Fig. 1, two fundamental questions need to be answered: 1) to which tile each IP should be mapped and 2) which routing algorithm is suitable for directing the information among tiles, such that the metrics of interest are optimized. More precisely, in order to get the best energy/performance tradeoff, the designer needs to determine the topological placement of these IPs onto different tiles. Referring to Fig. 1, this means to determine, for instance, onto which tile (e.g., t_{13}, t_7, etc.) each IP (e.g., DSP2, DSP3, etc.) should be placed. Since there may exist multiple minimal

Manuscript received June 25, 2003; revised December 25, 2003 and March 26, 2004. This work was supported in part by the National Science Foundation under CAREER Award CCR-0093104, in part by DARPA/Marco Gigascale Research Center (GSRC), and in part by the Semiconductor Research Corporation under Award 2001-HJ-898. Parts of this paper appeared as "Energy-Aware Mapping for Til-Cabes NoC Architectures Under Performance Constraints," in the *Proceedings of the ASP-DAC*, Kitakyushu, Japan, 2003, pp. 233–239, and as "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," in the *Proceedings of the Design, Automation, and Test in Europe Conference*, Munich, Germany, 2003, pp. 688–693. This paper was recommended by Associate Editor M. Pedram.

The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890 USA (e-mail: jingcao@ece.cmu.edu; radum@ece.cmu.edu).

Digital Object Identifier 10.1109/TCAD.2005.844106

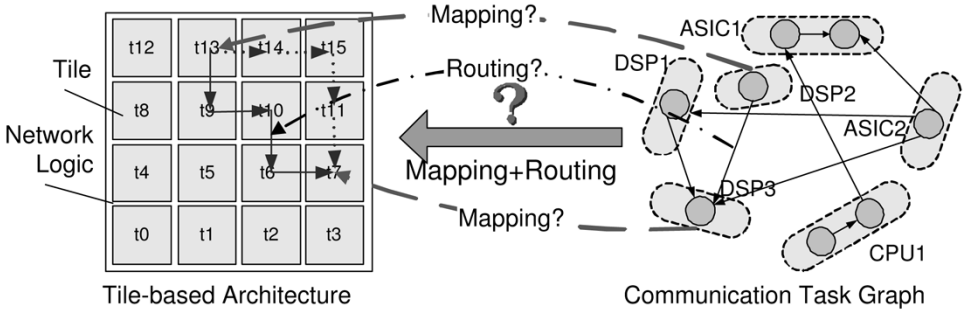


Fig. 1. Tile-based architecture and the illustration of the mapping/routing problems.

routing paths, one also needs to select *one* qualified path for each communicating pair of tiles. For example, one has to determine which path (e.g., $t_{13} \rightarrow t_9 \rightarrow t_{10} \rightarrow t_6 \rightarrow t_7$ or $t_{13} \rightarrow t_{14} \rightarrow t_{15} \rightarrow t_{11} \rightarrow t_7$, etc.) should the packets follow in order to send data from DSP2 to DSP3, if these two IPs are meant to be placed to tiles t_{13} and t_7, respectively.

While task assignment and scheduling problems have been addressed before [2], the *mapping* and *routing* problems described above represent a new challenge, especially in the context of the regular tile-based NoC architecture, as this significantly impacts the energy and performance metrics of the system. In this paper, we address this very issue and propose an efficient algorithm to solve it. To this end, we first propose a suitable routing scheme (Section III) and a new energy model (Section IV) for NoCs. The problem of mapping and routing path allocation are formulated in Section V. Next, an efficient branch-and-bound algorithm is proposed to solve this problem under performance constraints in Section VI. Experimental results in Section VII show that significant communication energy savings can be achieved, while guaranteeing the specified system performance. For instance, for a complex video/audio application, on average, 51.7% communication energy savings have been observed compared to a randomly generated implementation.

II. Related Work

In [1], Dally *et al.* suggest using the on-chip interconnection networks instead of ad hoc global wiring to structure the top-level wires on a chip and facilitate truly modular design. Along the same lines, Hemani *et al.* [4] present a honeycomb structure in which each processing core (resource) is located on a regular hexagonal node connected to three switches. In [5], Kumar *et al.* describe a NoC architecture implemented by a two-dimensional (2-D) mesh of switches and resources.

While these papers discuss the overall advantages and challenges of the regular NoC architecture, to the best of our knowledge, our work is the first to address the *mapping* and *routing path allocation* problems for tile-based architectures and provide an efficient way to solve them. Although routing (especially wormhole-based routing [3]) has been a hot research topic in the area of direct networks for parallel and distributed systems [6]–[8], the specifics of NoC design force us to rethink the standard network techniques and adapt them to the context of NoC architectures. In what follows, we address this issue by presenting a suitable routing technique for NoCs together with an algorithm for automatic generation of the routing function.

III. Proposed Routing Scheme for NoCs

In the following, we describe the major differences between macronetworks and NoCs and implications of adapting macronetwork routing techniques to the context of NoCs.

A. Buffering Space

To minimize the implementation cost, the on-chip network has to be implemented with little area overhead. This is especially critical for architectures consisting of tiles with fine-level granularity. Thus, instead of using huge memories (e.g., SRAM or DRAM) as buffering space for the routers/switches in the macronetwork, it is more reasonable to use small registers for on-chip routers. Another advantage of using registers over large memories is that the address decoding/encoding latency and the access latency can be significantly reduced. This is critical for those latency sensitive applications which are typical for many SoCs.

B. Wormhole Routing

Because of the limited buffering resources available and the stringent latency requirements for typical NoC applications, we believe that wormhole-based routing [3] is the most appropriate routing technique for NoCs. In wormhole routing, a packet is divided into *flow control digits* (*flits*). The flits are then routed through the network in a pipelined fashion, which leads to dramatically reduced communication latencies. The header flit in a packet contains all the routing information and leads the packet through the network. When the header flit is blocked due to congestion in the network, all of the trailing flits need to wait at their current nodes. Thus, large packet buffers at each intermediate node are obviated and only small buffers are required.

C. Deterministic Routing

There are quite a few wormhole-routing techniques published to date. In general, they can be classified into two categories: 1) deterministic and 2) adaptive [6]. We believe that for NoCs, deterministic routing is more suitable for the following reasons.

- *Resource limitation and stringent latency requirements.* Compared to deterministic routers, implementing adaptive routers requires by far more resources. Moreover, since in adaptive routing the

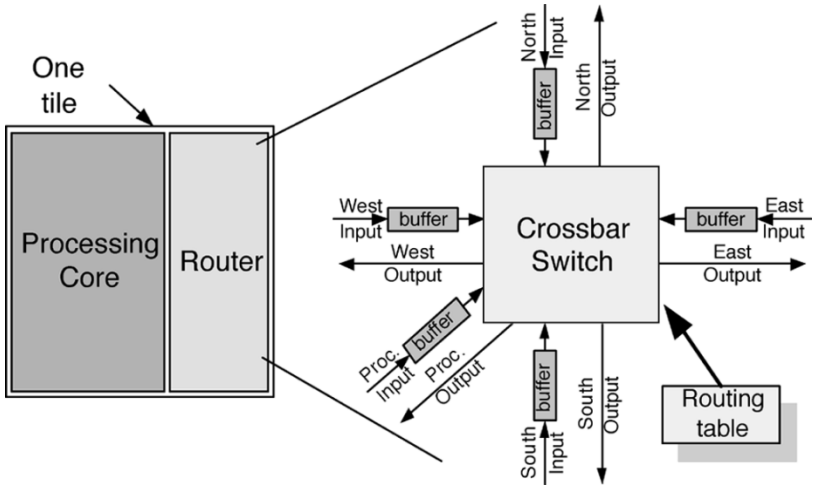


Fig. 2. Typical structure of a tile and the on-chip router.

packets may arrive out of order, huge buffering space is needed to reorder them. Together with the protocol overhead, this may lead to prohibitive costs, extra delay, and jitter.

- *Traffic predictability.* In contrast to typical macronetworks which represent general platforms for a large spectrum of applications, most NoCs are developed for one small set of applications. Consequently, the designer has a good understanding of the traffic characteristics and can avoid congestion by wisely mapping the IPs and allocating the routing paths.

D. Freedom of Deadlock and Livelock

A desirable feature of a routing algorithm is its freedom from *deadlock* and *livelock* [7]. All deterministic routing algorithms are livelock-free. Freedom from deadlock is especially critical for NoCs. Indeed, implementing a mechanism which automatically detects and recovers from deadlock may not be affordable in terms of silicon resources; it also may lead to unpredictable delays.

E. Programmability

Since the traffic characteristics vary significantly across different applications, it is necessary to reallocate the routing paths when the NoC platform is used for different applications. Since reallocation only involves reprogramming the routing table for each router, the cost of the programmability is almost negligible.

In summary, we argue that the appropriate routing technique for NoCs should be deterministic, deadlock-free, and minimal,[1] *wormhole-based*. Moreover, traffic characteristics should be considered when allocating the routing paths.

IV. Platform Description

In this section, we describe the regular tile-based architecture and the energy model for its communication network.

A. Architecture

The system under consideration is composed of $n \times n$ tiles interconnected by a 2-D mesh network (see Fig. 2). Each tile in Fig. 2 is composed of a *processing core* and a *router*. The router is connected to the four neighboring tiles and its local processing core via channels [each consisting of two one-directional point-to-point links].

Due to limited resources, the buffers are implemented using registers, typically in the size of one or two flits each. A 5×5 crossbar switch is used as the switching fabric in the router.

Each router has a routing table. Based on the source/destination address, the routing table decides which output link the packet should be delivered to.

B. Energy Model

Ye *et al.* [9] proposed a model for energy consumption of network routers. The *bit energy* (E_{bit}) metric is defined as the energy consumed when one bit of data is transported through the router

$$E_{\text{bit}} = E_{S_{\text{bit}}} + E_{B_{\text{bit}}} + E_{W_{\text{bit}}} \tag{1}$$

where $E_{S_{\text{bit}}}$, $E_{B_{\text{bit}}}$, and $E_{W_{\text{bit}}}$ represent the energy consumed by the switch, buffering and interconnection wires inside the switching fabric, respectively. Since $E_{W_{\text{bit}}}$ is the energy consumed on the wires inside the switch fabric, the energy consumed *on the links* between tiles $(E_{L_{\text{bit}}})$ should also be included. Thus, the average energy consumed in sending one bit of data from a tile to a neighboring tile can be calculated as

$$E_{\text{bit}} = E_{S_{\text{bit}}} + E_{B_{\text{bit}}} + E_{W_{\text{bit}}} + E_{L_{\text{bit}}}. \tag{2}$$

Since the length of a link is typically in the order of millimeters, the energy consumed by buffering $(E_{B_{\text{bit}}})$ and internal wires $(E_{W_{\text{bit}}})$ is negligible[2] compared to $E_{L_{\text{bit}}}$; (2) reduces to

$$E_{\text{bit}} = E_{S_{\text{bit}}} + E_{L_{\text{bit}}}. \tag{3}$$

[1]Minimal refers to routing packets along the shortest paths.

[2]We evaluated the energy consumption using Spice simulations for a 0.35-μm technology. The results show that $E_{B_{\text{bit}}} = 0.073$ pJ, which is indeed negligible compared to $E_{L_{\text{bit}}}$ (typically in the order of a few pJ).

This is a very nice approximation since it eliminates the $E_{B_{\text{bit}}}$ which is a parameter tightly coupled with network congestion for which an accurate value can only be measured by time-consuming simulations. Consequently, in the new model, the average energy consumption for sending one bit of data from tile t_i to tile t_j is

$$E_{\text{bit}}^{t_i,t_j} = n_{\text{hops}} \times E_{S_{\text{bit}}} + (n_{\text{hops}} - 1) \times E_{L_{\text{bit}}} \quad (4)$$

where n_{hops} is the number of routers the bit traverses from tile t_i to tile t_j.

It is interesting to note that, with (4), the communication energy consumption can now be analytically calculated independently of the underlying traffic model (e.g., Markovian, long-range dependence, etc. [12]), provided that the communication volume between any communicating IP pair is known. For 2-D mesh networks with *minimal* routing, (4) shows that the average energy consumption of sending one bit of data from t_i to t_j is determined by the *Manhattan* distance between them.

V. Problem of Energy- and Performance-Aware Mapping and Routing Path Allocation

A. Problem Formulation

Simply stated, for a given application, our objective is to decide on which tile should each IP be mapped to and how should the packets be routed, such that the total communication energy consumption is minimized under given performance constraints. To formulate this problem, we need the following definitions.

Definition 1: An application characterization graph (APCG) $\mathcal{G} = G(C, A)$ is a *directed* graph, where each vertex c_i represents one selected IP, and each directed arc $a_{i,j}$ characterizes the communication from c_i to c_j.[3] Each $a_{i,j}$ has the following properties:

- $v(a_{i,j})$ is the arc volume from vertex c_i to c_j, which stands for the communication volume (bits) from c_i to c_j.
- $b(a_{i,j})$ is the arc bandwidth requirement from vertex c_i to c_j, which stands for the minimum bandwidth (bits per second) that should be allocated by the network in order to meet the performance constraints.

Definition 2: An architecture characterization graph (ARCG) $\mathcal{G}' = G(T, R)$ is a *directed* graph, where each vertex t_i represents one tile in the architecture, and each directed arc $r_{i,j}$ represents the routing parameter from t_i to t_j. Each $r_{i,j}$ has the following properties:

- $P_{i,j}$ is the a set of candidate *minimal* paths from tile t_i to tile t_j. $\forall p_{i,j} \in P_{i,j}$, $L(p_{i,j})$ gives the set of links used by $p_{i,j}$.
- $e(r_{i,j})$ is arc cost. This represents the average energy consumption (Joules) of sending one bit of data from t_i to t_j, i.e., $E_{\text{bit}}^{t_i,t_j}$.

[3]The communication from c_j to c_i is represented as $a_{j,i}$. There can be at most two directional arcs between any vertex pair, with one arc dedicated to each direction.

Definition 3: For an ARCG $\mathcal{G}' = G(T, R)$, a deterministic routing function $\mathcal{R} : R \rightarrow P$ maps $r_{i,j}$ to *one* routing path $p_{i,j}$, where $p_{i,j} \in P_{i,j}$.

Using these definitions, the problem of energy/performance aware mapping and routing path allocation under performance constraints can be formulated as follows.

Given an APCG and an ARCG that satisfy

$$\text{size(APCG)} \leq \text{size(ARCG)} \quad (5)$$

find a mapping function map() from APCG to ARCG and a *deterministic*, *deadlock-free*, *minimal* routing function $\mathcal{R}()$ which:

$$\textbf{min}\left\{\text{Energy} = \sum_{\forall a_{i,j}} v(a_{i,j}) \times e\left(r_{\text{map}(c_i),\text{map}(c_j)}\right)\right\} \quad (6)$$

such that:

$$\forall c_i \in C, \qquad \text{map}(c_i) \in T \quad (7)$$

$$\forall c_i \neq c_j \in C, \qquad \text{map}(c_i) \neq \text{map}(c_j) \quad (8)$$

$$\forall l_k, B(l_k) \geq \sum_{\forall a_{i,j}} b(a_{i,j}) \times f\left(l_k, \mathcal{R}\left(r_{\text{map}(c_i),\text{map}(c_j)}\right)\right) \quad (9)$$

where $B(l_k)$ is the bandwidth of link l_k and

$$f(l_k, p_{m,n}) = \begin{cases} 0, & l_k \notin L(p_{m,n}) \\ 1, & l_k \in L(p_{m,n}) \end{cases}.$$

To give a little bit of intuition, conditions (7) and (8) mean that each IP should be mapped to exactly one tile and no tile can host more than one IP. Equation (9) specifies the communication performance constraints for the problem in terms of the aggregated bandwidth requirements for each link. More precisely, the resulting network has to guarantee that the communication traffic (workload) of any link does not exceed the available bandwidth, such that the bandwidth requirements between each communicating IP pair can be satisfied.

We need to note that although latency is another very important performance metric, it is also a very difficult metric to evaluate, especially since the characterization of the traffic itself is difficult for most applications. Moreover, the accurate latency estimation may depend on many other factors, such as packet/flit size, router's arbitration scheme, etc. Thus, similar to other work in the literature (e.g., [13]), we use the bandwidth requirement as a performance constraint. Another advantage is that the bandwidth requirement is actually indirectly related to the packet latency. For instance, the designer can calculate the bound of packet latency with the specified bandwidth using the techniques presented in [14].

Having this problem formulation, the network to be synthesized guarantees that the packets from the same source IP to the same destination IP will always arrive in order, as the resulting network is indeed deterministic. As shown in Definition 3, a deterministic routing function maps $r_{i,j}$ to *one* routing path $p_{i,j}$, where $p_{i,j} \in P_{i,j}$. This means that given a source IP c_i and a destination IP c_j, the algorithm decides *only one* routing path for *all* packets sent from c_i to c_j. Thus, the packets that belong

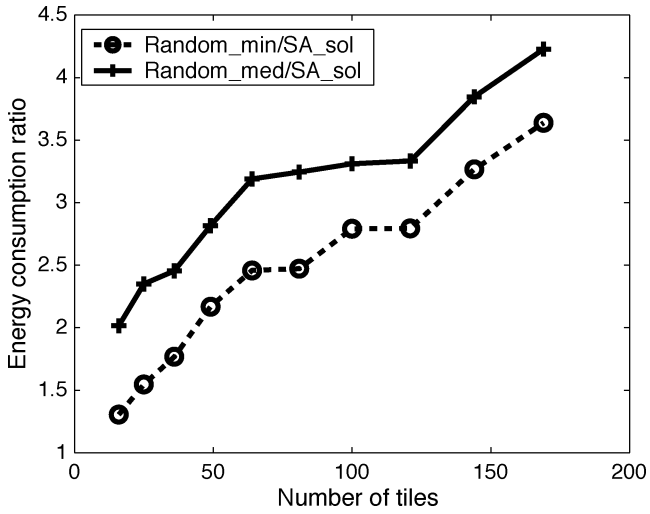


Fig. 3. Impact of mapping on energy consumption.

to the same message will *never* arrive out of order since they have the same source and destination.

B. Significance of the Problem

To prove that the choice of mapping heavily affects the communication energy consumption, we consider the following experiment. A series of task graphs are generated using the TGFF package [10]. Then the output graph is randomly assigned to a given number of IPs, with the computational times and communication volumes randomly generated according to a specified distribution. Our tool is then used to preprocess and annotate these task graphs and build the communication task graphs (CTGs), which characterize the application partitioning, task assignment, scheduling, communication patterns, and task execution time. Also, the bandwidth requirements between any communicating IP pairs are calculated.

The number of IPs used in the experiment ranges from 3×3 to 13×13. For each benchmark, we generate 3000 random mapping configurations and the corresponding energy consumption values are calculated. At the same time, an optimizer based on simulated annealing (SA) was also developed and used with the goal of finding the legal mapping which consumes the least amount of communication energy. The resulting energy ratios are plotted in Fig. 3.

The dashed line in Fig. 3 shows the energy consumption ratio between the best solution among the 3000 random mappings (*Random_min*) and the solution found by the SA (*SA_sol*). The solid line shows the ratio between the median solution among the 3000 random mappings (*Random_med*) and *SA_sol*.

As we can see, although the SA optimizer does not necessarily find the optimal solution, it still saves around 50% energy compared to the median solution for the system consisting of 3×3 tiles. Moreover, the savings increase as the system size scales up. For instance, for the system with 13×13 tiles, the savings can be as high as 75%. Another observation is that the best solution among the 3000 random mappings is far from satisfactory, even for a system as small as 3×3 tiles.

From the routing perspective, it is not unusual to apply XY routing to this kind of systems since minimal, deterministic, and deadlock-free routing is needed. Although it eliminates the complexity in routing path allocation, the restriction to XY routing shrinks the solution space when performance constraints are specified, as the flexibility in choosing suitable routing paths is obviously sacrificed. This will be demonstrated by our experimental results as shown in Section VII.

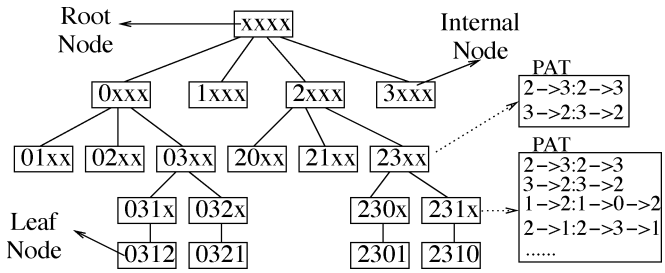


Fig. 4. Example search tree.

Therefore, mapping and routing path allocation are critical to designing a system which consumes the least amount of energy while satisfying the specified performance constraints. Unfortunately, neither of these problems is simple. The mapping problem by itself is an instance of constrained *quadratic assignment problem* which is *NP-hard* [11]. The search space of the problem increases *factorially* with the system size. Even for a system with 4×4 tiles, there can be 16! mappings which are already impossible to enumerate, not to mention systems with 10×10 tiles that are anticipated in five years or so [5]. Moreover, the routing path allocation is also computationally unaffordable if we have to enumerate all the combinations of routing path allocations while guaranteeing freedom from deadlock.

In the following section, we propose an efficient heuristic which is able to find nearly optimal solutions in reasonable run times.

VI. Energy- and Performance-Aware Mapping and Routing Path Allocation

A. Data Structure

Our approach is based on a *branch-and-bound* algorithm which efficiently walks through the *searching tree* that represents the solution space. Fig. 4 shows a searching tree example for mapping a four-IP application onto a 2×2-tile architecture.

Each node in the tree is either a *root* node, an *internal* node, or a *leaf* node. The root node (labeled "*xxxx*") corresponds to the state where *no* IP has been mapped and *no* routing path has been allocated. Each internal node represents a *partial* mapping. For example, the node labeled "*23xx*" represents a partial mapping where IP_0 and IP_1 are mapped to tile t_2 and tile t_3, respectively, while IP_2 and IP_3 are not mapped yet. Each leaf node represents a *complete* mapping of the IPs to the tiles. Each node also has a path allocation table (PAT) which stores the routing paths for the traffic among its *occupied* tiles. For instance, the PAT of node "*231x*" in Fig. 4 shows that the traffic from t_1 to t_2 takes the path $t_1 \rightarrow t_0 \rightarrow t_2$, etc. Similarly, the PAT of any other node is automatically generated by the algorithm. When a child node is generated, the PAT of its parent node is automatically inherited. Next, the routing paths for the traffic involving

the newly occupied tile are allocated and added to its PAT. Caution must be taken to ensure freedom of deadlock.

To further explain how our algorithm works, the following definitions are needed.

Definition 4: The *cost* of a node is the total energy consumed by the communication among all IPs that have already been mapped.

Definition 5: Let $\mathcal{M}$ be the set of vertices in the APCG that have already have been mapped. A node is called a *legal* node if and only if it satisfies the following two conditions.

- The routing paths specified by the PAT are deadlock free.
- $B(l_k) \geq \sum_{\forall a_{i,j}, c_i c_j \in \mathcal{M}} b(a_{i,j}) \times f(l_k, p_{\mathrm{map}(c_i),\mathrm{map}(c_j)})$, $\forall l_k$, where $p_{\mathrm{map}(c_i),\mathrm{map}(c_j)}$ is the routing path from tile $\mathrm{map}(c_i)$ to $\mathrm{map}(c_j)$ specified by the PAT.

Definition 6: The upper bound cost (UBC) of a node is defined as a value that is no less than the *minimum* cost of its *legal*, descendant *leaf* nodes.

Obviously, based on this definition, if a node has a UBC cost of x, then it has at least one *legal* descendant leaf node whose cost is no larger than x.

Definition 7: The lower bound cost (LBC) of a node is defined as the *lowest* cost that its descendant leaf nodes can *possibly* achieve.

Differently stated, this means that if a node has the LBC equal to x, then each of its descendant leaf nodes has at least a cost of x.

B. Branch-and-Bound Algorithm

Given the above definitions, finding the optimal solution of (6) is equivalent to finding the *legal leaf* node which has the *minimal* cost. To achieve this, our algorithm searches the optimal solution by alternating the following two steps.

Branch: In this step, an unexpanded node is selected and its next *unmapped* IP is enumeratively assigned to the remaining *unoccupied* tiles to generate the corresponding new child nodes. The PAT of each child node is also generated by first copying its parent node's PAT and then allocating the routing paths for the traffic between the newly occupied tile and the other occupied tiles. The routing paths specified by the PAT have to be deadlock-free.

Bound: Each of the newly generated child nodes is inspected to see if it is possible to generate the best leaf nodes later. A node can be trimmed away without further expansion if either its cost or its LBC is higher than the lowest UBC that has been found so far, since it is guaranteed that other nodes will eventually lead to a better solution.

How the algorithm allocates the routing paths and computes UBC/LBC are critical to its performance. Better routing path allocation helps balancing the traffic which leads to better solutions, but needs more time to compute. Tight UBC and LBC help in trimming away more nonpromising nodes early in the search, but also demand more computational time.

Next, we describe our routing path allocation heuristic which can find a good routing path allocation within reasonably short computational times. We also show our method for computing UBC and LBC, which offers a satisfactory tradeoff between the average computation time for processing one node and the number of nodes that need to be processed.

Step 1: Routing Path Allocation: Routing path allocation has two objectives. First, all the routing paths have to be *minimal* and the resulting network has to be *deadlock-free*. Second, it needs to balance the traffic across all the available links so that no link will be overloaded.

To be deadlock free, the routing algorithm needs to prohibit *at least one turn* in each of the possible routing cycles. In addition, it should *not* prohibit more turns than necessary to preserve the adaptiveness. Based on this, several deadlock-free adaptive routing algorithms have been proposed [7], including west-first, north-last, and negative-first. In [8], Chiu proposed the *odd-even* turn model which restricts the locations where some types of turns can take place such that the algorithm remains deadlock-free.

In our algorithm, we convert the adaptiveness offered by the above algorithms into flexibility for the routing path allocation by constructing the legal turn set (LTS). 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. The advantage of using LTS is twofold. First, since the turns are restricted to LTS, deadlock-free routing is guaranteed. Second, since only minimum turns are prohibited, the routing path allocation is still highly flexible.

Theoretically, the turns allowed in *any* of such deadlock-free adaptive routing algorithms can be used to construct an LTS. In this paper, we chose the west-first and odd-even routing algorithm to build our LTSs.[4] In short, west-first routing prohibits all the north $\rightarrow$ west and south $\rightarrow$ east turns, all the remaining turns being legal. While the odd-even routing prohibits the east $\rightarrow$ north and east $\rightarrow$ south turns at any tiles located in an even column, it also prohibits the north $\rightarrow$ west and south $\rightarrow$ west turns at any tiles located in an odd column.

Both of these two algorithms prohibit only 1/4 of the total possible turns. However, the degree of the adaptiveness provided by odd-even routing is distributed more evenly than that provided by west-first [8].

Given an LTS, the following heuristic is used to allocate routing paths for a list of communication loads (LCL):

In Fig. 5, the communication load (CL) is first sorted by paths flexibility. The flexibility of a CL can be 1 (if there exists more than one *legal* path[5]) or 0, otherwise. CLs with lower flexibility are given higher priorities for path allocation. If two CLs are tied in flexibility, the one with the higher bandwidth requirement is given priority. Function *choose_link* returns the *least* loaded link allowed by LTS.

Step 2: UBC Calculation: From definition 6, the cost of any *legal* descendant leaf node can be used as the UBC of that node. Since a tight UBC cost is preferred, we choose the descendant leaf node by greedily mapping the remaining unmapped IPs.

[4]North-last and negative-first routing are similar to west-first routing.

[5]A *legal* path has to be *minimal* and employ only those turns in the LTS.

```
Sort LCL by the flexibility of each CL
for each CL in LCL {
   cur_tile = CL.source; dst_tile=CL.destination
   while(cur_tile ≠ dst_tile) {
     link = choose_link(cur_tile, dst_tile);
     cur_tile = link.next_tile();
     link.add_load(CL.bandwidth);
     }
}
```

Fig. 5. Pseudocode of the routing path allocation algorithm.

In each step, the next unmapped IP c_k with the highest communication demand is selected and its *ideal* topological location (x, y) on the chip is calculated as:

$$x = \frac{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right) \times c_i^x}{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right)} \tag{10}$$

$$y = \frac{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right) \times c_i^y}{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right)} \tag{11}$$

where c_i^x and c_i^y represent the row id and column id of the tile that c_i is mapped onto, respectively, and $\mathcal{M}$ is the set of mapped IPs. c_k is then mapped to an unoccupied tile whose topological location has the *smallest Manhattan* distance to (x, y).

This step is repeated until all IPs have been mapped, which leads to a leaf node. The aforementioned heuristic is then used to allocate routing paths for the unallocated traffic. If this leaf node is illegal, then the UBC of the node under inspection is set to be infinitely large; otherwise, it is set to be the cost of that leaf node.

Step 3: LBC Calculation: The LBC cost of a node n can be decomposed into *three* components

$$\text{LBC} = C_{m,m} + C_{u,u} + C_{m,u}. \tag{12}$$

$C_{m,m}$ is the cost of the intercommunication among mapped IPs and can be calculated exactly. $C_{u,u}$ is the cost of the intercommunication among the unmapped IPs and can be calculated using (13) ($\bar{\mathcal{M}}$ and $\bar{\mathcal{O}}$ are the sets of unmapped IPs and unoccupied tiles, respectively)

$$C_{u,u} = \frac{1}{2} \times \sum_{\forall c_i \in \bar{\mathcal{M}}} \sum_{\forall c_j \in \bar{\mathcal{M}}} v(a_{i,j}) \times \min_{\forall t_m t_n \in \bar{\mathcal{O}}} e(r_{m,n}) \tag{13}$$

Lastly, $C_{m,u}$ represents the cost of the intercommunication between the mapped IPs and the unmapped IPs. $C_{m,u}$ can be derived by:

$$C_{m,u} = \sum_{\forall c_i \in \mathcal{M}} \sum_{\forall c_j \in \bar{\mathcal{M}}} \left(v(a_{i,j}) + v(a_{j,i})\right) \times \min_{\forall t_k \in \bar{\mathcal{O}}} e\left(r_{\text{map}(c_i),k}\right). \tag{14}$$

C. Pseudocode of the Algorithm

Fig. 6 gives the pseudocode of our algorithm. Two speedup techniques are proposed to trim away more nonpromising nodes early in the search process.

- *IP ordering*. IPs are sorted by their communication demands ($\sum_{\forall j \neq i} \{v(a_{i,j}) + v(a_{j,i})\}$ for IP c_i) so that the IPs with higher demand will be mapped earlier. Since the positions of the IPs with higher demands have larger impact on the overall communication energy consumption, fixing their positions earlier helps exposing those nonpromising nodes earlier in the searching process; this reduces the number of nodes to be expanded. As most applications have nonuniform traffic patterns, this heuristic is quite useful in practice.
- *Priority queue (PQ)*. The PQ sorts the nodes to be branched based on their cost. The lower the cost of the node, the higher the priority it is assigned for branching. Intuitively, expanding a node with lower cost will likely decrease the minimum UBC so that more nonpromising nodes can be detected.

```
Sort the IPs by communication demand
root_node = new node(NULL)
MUBC = +∞, best_mapping_cost = +∞
PQ.Insert(root_node)
while(!PQ.Empty()) {
   cur_node = PQ.Next()
   for each unoccupied tile t_i {
     generate child node n_new
     allocate routing paths
     if(n_new's mirror node exists in the PQ)
       continue
     if(n_new.LBC>MUBC)
       continue
     if(n_new.isLeafNode) {
       if(n_new.cost < best_mapping_cost) {
         best_mapping_cost = n_new.cost
         best_mapping = n_new
       }
     }
     else {
       if(n_new.UBC<MUBC)
         MUBC = n_new.UBC
       PQ.insert(n_new)
     }
   }
}
```

Fig. 6. Pseudocode of the mapping algorithm.

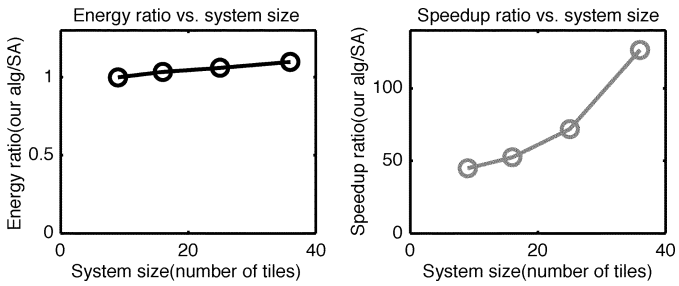


Fig. 7. Comparison between SA and EPAM-XY for a random set of benchmarks.

Obviously, as the system size scales up, the runtime of the algorithm will also increase. Fortunately, we can tradeoff the solution quality with runtime by limiting the maximum length of PQ. When PQ's length reaches a threshold value, strict criteria are applied to select the child nodes for insertion into PQ.

VII. Experimental Results

A. Evaluation on Random Applications

We first compare the runtime and the solution quality of our algorithm against an SA optimizer. To make the comparison fair, SA was optimized by carefully selecting parameters such as the number of moves per temperature, cooling schedule, etc. Since SA optimization is restricted to XY routing, for the first set of

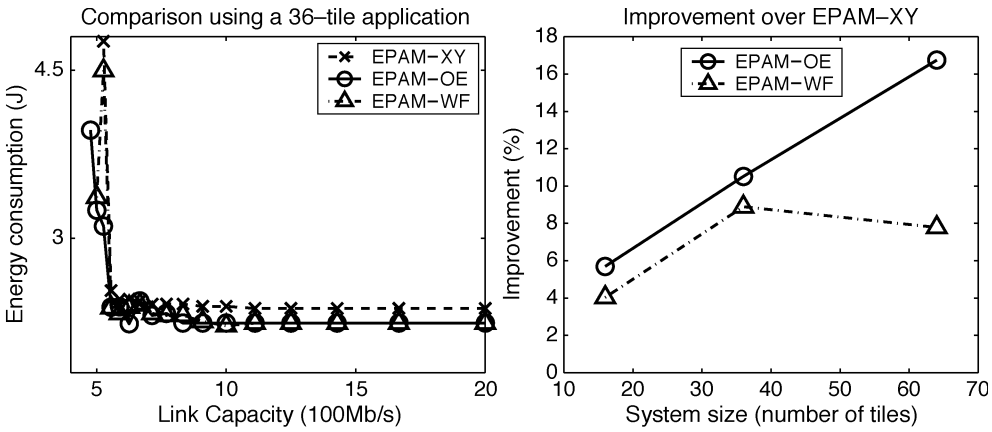


Fig. 8. Effectiveness of exploiting the routing flexibility.

experiments, we also configure our algorithm to run without exploiting the routing flexibility by fixing it to do just XY routing. In the following, we call this version EPAM-XY which stands for energy- and performance-aware mapping with XY routing.

Four categories (I, II, III, and IV) of random benchmarks were generated, each containing ten applications with 9, 16, 25, and 36 IPs, respectively. EPAM-XY and SA were then applied to map these applications onto architectures with the same number of tiles. The results are shown in Fig. 7.

As shown in Fig. 7, for applications with nine tiles, EPAM-XY runs 45 times faster than SA, on average. As the system size scales up, the speedup also increases dramatically. For applications with 36 tiles, the average speedup of EPAM-XY over SA increases to 127. Meanwhile, the solutions produced by our algorithm remain very competitive. Indeed, the energy consumption of the solutions generated by EPAM-XY is only 3%, 6% and 10% larger compared to the SA solutions for category II, III, and IV, respectively. For category I, EPAM-XY even finds better solutions because it can walk through the entire search tree due to the small problem size.

To evaluate the benefits of exploiting the routing flexibility, we also applied our algorithm on benchmark applications using different LTS configurations. The comparison between the algorithms with LTSs based on XY routing (EPAM-XY), odd-even routing (EPAM-OE), and west-first (EPAM-WF) is shown in Fig. 8.

The left part of Fig. 8 shows how EPAM-XY, EPAM-OE, and EPAM-WF perform for a typical 36-tile application as the link bandwidth constraints change. The results demonstrate two advantages of routing flexibility exploitation.

1) It helps to find solutions for architectures with a lower link bandwidth (which implies a lower implementation cost). For example, for this application, without exploiting the routing flexibility, the link bandwidth has to be 526 Mb/s (or higher) in order to find a solution which meets the performance constraints. By exploiting the routing flexibility, the link bandwidth requirement decreases to 476 Mb/s (EPAM-OE) and 500 Mb/s (EPAM-WF), respectively.
2) Given the same architecture, exploiting routing flexibility leads to solutions with less energy consumption. This benefit becomes more significant as the links' loads reach their capacity. For instance, using the architecture where each link can provide 526 Mb/s bandwidth, the energy consumption of the solution generated by EPAM-XY is 4.80 J. The energy consumptions of the solutions generated by EPAM-OE and EPAM-WF are only 4.50 and 3.12 J, respectively, so significant energy savings are achieved.

The right part of Fig. 8 shows how the effectiveness of routing flexibility exploitation changes as the problem size scales up. Let $BW^{\min}$ be the minimum link bandwidth needed by the corresponding algorithm to be able to find a solution which meets the performance constraints. The improvement (plotted on the y axis) is defined as $(\mathrm{BW}_{XY}^{\min} - \mathrm{BW}^{\min})/\mathrm{BW}^{\min}$, which is a measure of how much the link bandwidth can be relaxed by exploiting routing flexibility compared to that using EPAM-XY. As we can see, both EPAM-OE and EPAM-WF perform better than EPAM-XY. For applications mapped onto 16 tiles, EPAM-OE provides 5.69% improvement over EPAM-XY, on average. As the problem size scales up to 36 and 64 tiles, the average improvement increases to 10.51% and 16.74%, respectively.

Overall, EPAM-OE performs much better than EPAM-WF, due to the fact that the odd-even routing provides more even adaptiveness than the west-first routing [8].

B. Complex Video/Audio Application

To evaluate the potential of our algorithm for real applications, we applied it to a generic MultiMedia System (MMS). MMS is an integrated video/audio system which includes an H263 video encoder, an H263 video decoder, an MP3 audio encoder, and an MP3 audio decoder. We first partition MMS into 40 concurrent tasks and then assign/schedule these tasks onto 16 selected IPs available from industry [16]. These IPs range from DSPs, generic processors, and embedded DRAMs to customized application specific integrated circuits (ASICs). We then use real video and audio clips as inputs to derive the communication patterns among these IPs. Fig. 9 shows the CTG of this system based on the simulation results. Every link connecting two tasks on different IPs represents data dependency, and is tagged with a number showing the volume of data (in

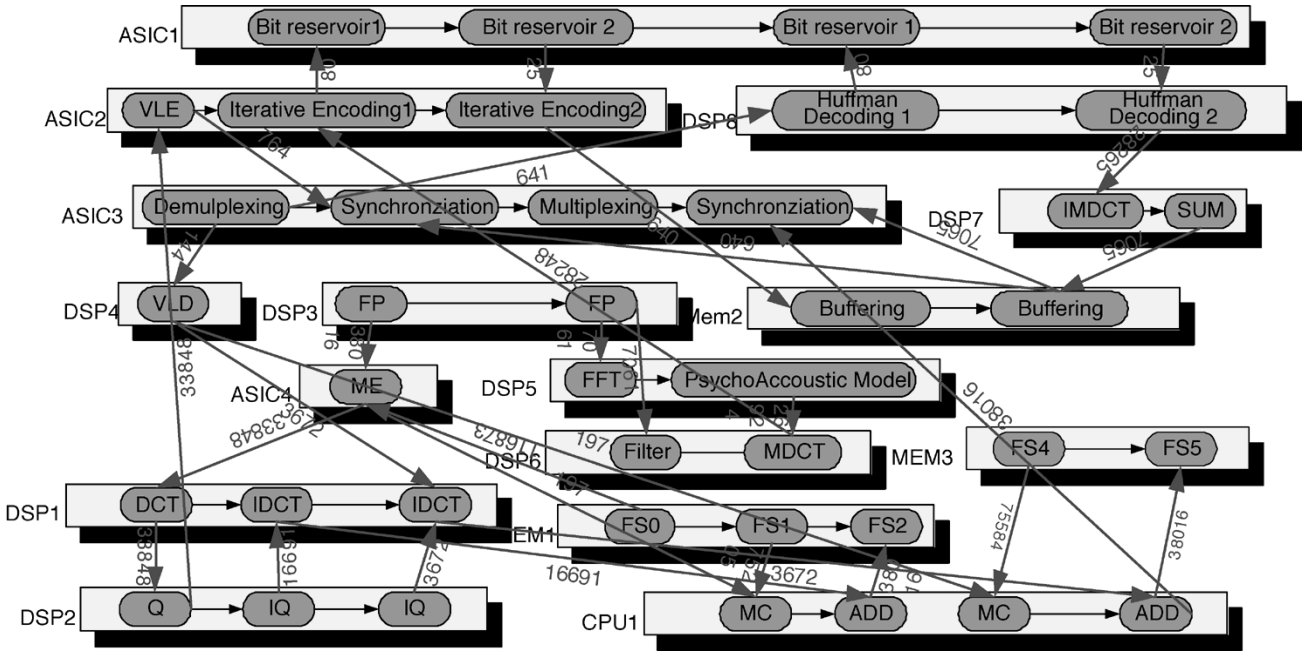


Fig. 9. Communication task graph for a complex multimedia application.

TABLE I
COMPARISON OF AD HOC MAPPING VERSUS EPAM-OE

Movie clips	Ad-hoc(mJ)	EPAM-OE(mJ)	Savings
box/hand	1369	662.4	51.6%
akiyo/cup	1810	838.4	53.7%
man/phone	2133	1070	49.8%

TABLE II
COMPARISON BETWEEN SA AND EPAM-OE

	SA	EPAM-OE	Improvement
Run Time (sec)	25.55	0.31	82.419X
Energy (mJ)	954.9	841.0	11.9%

bytes) to be transferred between these two tasks. Each oval in Fig. 9 represents a task.[6]

Applying EPAM-OE to MMS, the solution to the mapping problem is found in less than 0.5 s CPU time. A randomly generated implementation was also developed to serve as reference. The results are shown in Table I.

In Table I, each row represents the energy consumption of using two movie clips as simulation inputs, with one clip for the video/audio encoder and the other for the video/audio decoder. Compared to the ad hoc solution, we observe 51.7% communication energy savings, on average, which demonstrates the effectiveness of our algorithm.

To compare the performance between EPAM-OE and SA, both of them are applied to MMS for an architecture whose link bandwidth is fixed to 333 Mb/s. The results are shown in Table II.

As shown in Table II, our algorithm generates a better solution (about 12% less energy) with a significantly shorter runtime

[6]To make the figure more readable, the execution time of each task and the associated deadline of some tasks are not shown in Fig. 9.

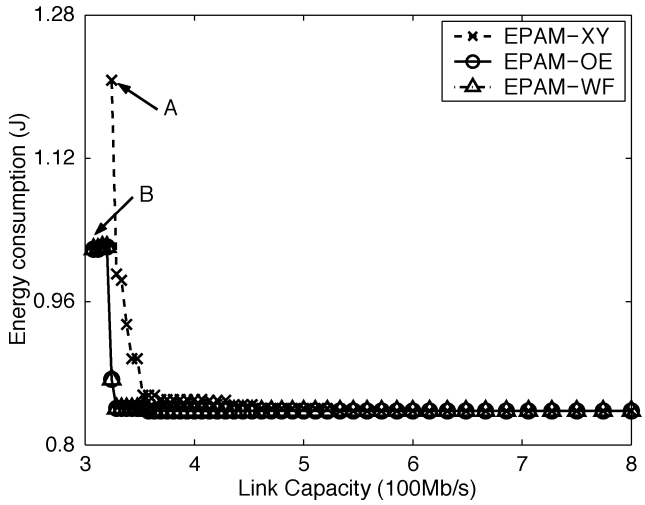


Fig. 10. Comparison among different routing schemes using a complex multimedia application.

(more than 82 times faster) compared to SA. We should point out that while the time needed by SA to complete is affordable for this system (because it has only 4×4 tiles), the runtime of SA increases dramatically as the system size scales up. For instance, for systems with 7×7 tiles, the average runtime of SA increases to 2.2 h. For systems with 10×10 tiles, our algorithm needs just a few minutes to complete while the runtime of SA becomes prohibitive (in our experiments, SA did not finish in 40 h of CPU time).

To show how much improvement our algorithm can achieve by exploiting the routing flexibility, we applied EPAM-XY, EPAM-OE and EPAM-WF to the MMS (Fig. 10). As shown in Fig. 10, EPAM-XY fails to find a solution when the link bandwidth decreases to 324 Mb/s (point A in Fig. 10). In contrast, both EPAM-OE and EPAM-WF can still find solution even the link bandwidth decreases to 307 Mb/s (point B in

Fig. 10), thus suggesting a 5.5% improvement. Given the same architecture whose link bandwidth is fixed to 324 Mb/s, the energy consumption of the solutions found by EPAM-OE and EPAM-WF are 873.6 mJ, while the energy consumption of the EPAM-XY solution is 1207 mJ; thus, 27.6% energy savings are achieved. Again, this shows the advantage of exploiting the routing flexibility.

VIII. Possible Extensions

Next, we describe two possible extensions of the current approach which are highly relevant to future NoCs.

A. Regions With Irregular Sizes

Regular NoC architecture may suffer from serious die-area waste when the comprising IPs of the application under design vary significantly in terms of individual sizes. In addition, the tile size has to fit the largest IP, thus increasing the link length between neighboring on chip routers. This may lead to performance degradation and energy inefficiency.

To solve this issue, region-based NoC allows the regular NoC designs to embed irregular regions [5]. More specifically, a *region* is an area inside the NoC which can be insulated from the network. Compared to other regular tiles, a region may have a different internal topology and communication mechanism. Fig. 11 gives an example of such architecture in which the shaded IP near the center occupies an area four times larger than the other regular IPs.

The proposed mapping algorithm can be adapted to handle applications with such irregular-sized regions. The basic idea to deal with regions which cover two or more tiles is to modify the corresponding APCG of the application by dividing the regions into several smaller IPs such that each dummy IP can fit into a single tile. Large weights need to be assigned to the arcs which connect the dummy IPs belonging to the same region. Due to these large weights, the mapping algorithm will then map these dummy IPs close to each other's proximity such that the region required shape can be maintained when merging these dummy IPs after the mapping process.

We want to point out, however, that the concrete implementation of such an extension is highly dependent on the structure of the region-based architecture. However, the region-based implementation is an active research topic where many questions remain to be answered. For instance, if a region covering 3×3 tiles is to be used in the design, one question would be how many routers should the region be connected to? Should there be only one router in the center of the region [Fig. 12(a)], or one router in each of the boundary dummy tiles [Fig. 12(b)], or maybe a router in every dummy tiles [Fig. 12(c)]?

Obviously, the answer to such a question will have a significant impact on the mapping process; more than that, it may even affect the routing algorithm itself. Take Fig. 12(a) for instance, the routing algorithm (e.g., XY, odd-even, etc.) has to be modified in order to be applicable to such region-based designs, which means that a change of our routing path allocation algorithm has to be made as well. To summarize, the answers to this kind of questions have to be found *before* a complete mapping algorithm can be developed to support region-based designs.

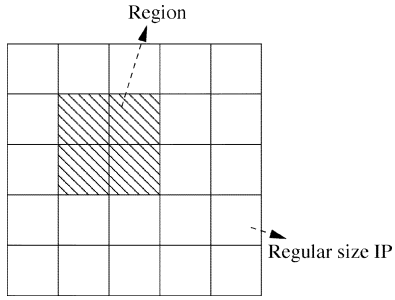


Fig. 11. Example of a regular NoC containing a region.

Nonetheless, as an example, we describe how the proposed mapping algorithm can be extended if the region structure as shown in Fig. 12(c) is used. Suppose the application has a total of 30 IPs, among them 28 IPs have a size of 1 unit long and 1 unit wide, while the remaining 2 IPs (c_4 and c_6) have a size of 2 unit long and 2 unit wide. The application is to be mapped to a 6×6 mesh. Fig. 13(a) shows part of its original APCG where c_4 and c_6 are highlighted.

In order to handle these irregular-sized regions, c_4 and c_6 are split into dummy IPs $c_4^0 - c_4^3$ and $c_6^0 - c_6^3$ [Fig. 13(b)], respectively.[7] Each dummy IP will be treated as other regular-sized IPs by the mapping algorithm (therefore it needs to be mapped onto one tile). The thick dummy arcs connecting dummy IPs (e.g., the arc connecting c_6^0 to c_6^1) are assigned with dummy arc volumes $(v(a_{i,j}))$, which are orders of magnitude larger than those of regular arcs. On the other hand, the bandwidth requirement $(b(a_{i,j}))$ of those arcs are assigned to be zero, since the dummy communication inside a region does not really consume any on-chip network bandwidth and is used only to glue the mapped dummy IPs together topologically.

After such modifications, the mapping algorithm can be applied to this modified APCG without any change; the dummy IPs belonging to the same region are guaranteed to be grouped in a square shape. To support this claim, Fig. 14 shows the mapping results of applying our algorithm on Fig. 13, with IP c_4 and c_6 mapped onto the left bottom part of the chip.

Obviously, the aforementioned technique can be extended to regions of nonsquare shapes by appropriately deriving the dummy arcs and assigning dummy volumes and bandwidth requirements to these arcs.

B. Preplaced IPs

Sometimes, it is preferable to preplace certain IPs at fixed locations on the chip. For instance, a designer may prefer to preplace I/O-related IPs to the peripheral region of a chip.

Our algorithm can be easily extended to handle such a case by premapping those IPs. More precisely, the IP ordering step can be modified such that the preplaced IPs get mapped onto the required tiles before the other IPs. The algorithm can then make its mapping decision only for the remaining IPs and tiles without any other change.

[7]The number of dummy IPs for a region depends on the size of the region.

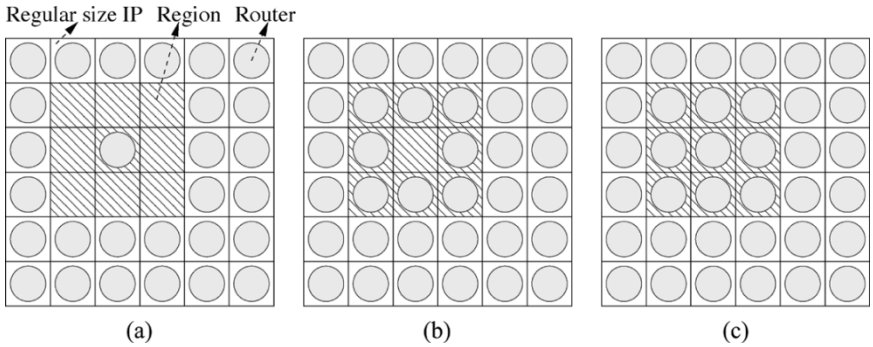


Fig. 12. Examples of possible region implementations.

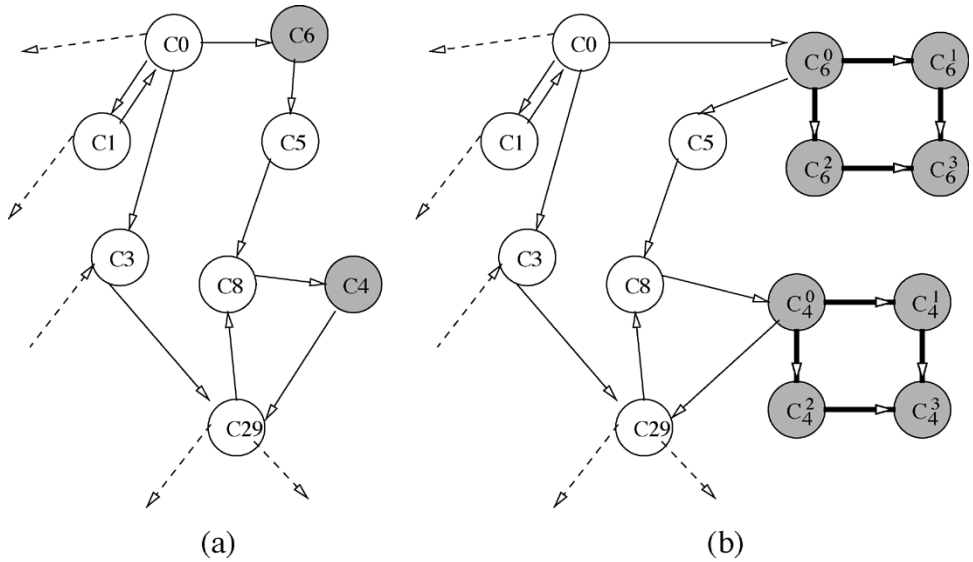


Fig. 13. Example APCG for an NoC containing two regions.

C_9	C_{13}	C_{28}	C_{17}	C_{21}	C_{22}
C_{14}	C_{20}	C_{12}	C_{29}	C_{10}	C_{26}
C_7	C_3	C_0	C_{18}	C_{16}	C_{19}
C_2	C_5	C_1	C_8	C_{11}	C_{15}
C_4		C_6		C_{24}	C_{25}
				C_{27}	C_{23}

Fig. 14. Mapping result obtained by the modified algorithm with the specified region shape maintained.

IX. Conclusion and Future Work

In this paper, we addressed the mapping and routing path allocation problems for regular tile-based NoC architectures. An efficient algorithm was proposed, which automatically maps the IPs to tiles and generates a suitable deadlock-free routing function such that the total communication energy consumption is minimized under specified performance constraints. As suggested, although we focus on the architectures interconnected by 2-D mesh networks, our algorithm can be adapted to other *regular* architectures with different network topologies. This remains to be done as future work.

The presented mapping algorithm takes APCG graph as the input, which assumes that the tasks and communication transactions have already been scheduled onto a set of selected IPs. The separation of the scheduling procedure and the mapping/routing procedure may lead to the suboptimality of the solution. Some of our initial work has been presented in [15] to address the communication and task scheduling for regular NoCs. However, more work needs to be done in order to efficiently merge the scheduling and mapping procedures.

Acknowledgment

The authors would like to thank the associate editor and anonymous reviewers for their suggestions that contributed to improving several drafts of this paper.

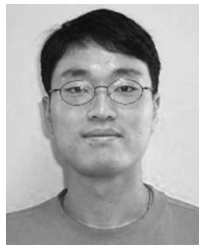
References

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. Design Automation Conf.*, Jun. 2001, pp. 684–689.

[2] J. Chang and M. Pedram, "Codex-dp: co-design of communicating systems using dynamic programming," *IEEE Trans. Coputer-Aided Design Integr. Circuits Syst.*, vol. 19, no. 7, pp. 732–744, Jul. 2002.

[3] W. J. Dally and C. L. Seitz, "The torus routing chip," *J. Distributed Comput.*, vol. 1, no. 3, pp. 187–196, 1986.

[4] A. Hemani *et al.*, "Network on a chip: An architecture for billion transistor era," in *Proc. IEEE NorChip Conf.*, Nov. 2000, pp. 166–173.
[5] S. Kumar *et al.*, "A network on chip architecture and design methodology," in *Proc. Symp. VLSI*, Apr. 2002, pp. 105–112.
[6] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.
[7] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, May 1992, pp. 278–287.
[8] G. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distributed Syst.*, vol. 11, no. 7, pp. 729–738, Jul. 2000.
[9] T. T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Proc. Design Automation Conf.*, Jun. 2002, pp. 524–529.
[10] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Int. Workshop Hardware/Software Codesign*, Mar. 1998, pp. 97–101.
[11] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *Freeman*, 1979.
[12] G. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for MPEG-2 video applications," *IEEE Trans. VLSI Syst.*, vol. 12, no. 1, pp. 108–119, Jan. 2004.
[13] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Proc. Design Automation Conf.*, Jun. 2002, pp. 783–788.
[14] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. New York: Springer–Verlag, 2001.
[15] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. Design, Automation, Test Eur.*, Feb. 2004, pp. 234–239.
[16] Mentor Graphics IP Core Catalog. *http://www.mentor.com/products/ip/product_index.cfm* [Online]



Jingcao Hu (S'01) received the B.S. and M.S. degrees in electronics engineering from Tsinghua University, Beijing, China, in 1998 and 2000, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering at Carnegie Mellon University, Pittsburgh, PA.

His research focuses on design methodologies for system on a chip with special interest on on-chip communication and low power.

Mr. Hu received the Best Paper Award from the Design, Automation, and Test in Europe (DATE) Conference in 2003.

Radu Marculescu (M'94) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1998.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA. His current research focuses on developing design methodologies and software tools for system-on-a-chip design, on-chip communication, and ambient intelligence.

Prof. Marculescu is a Member of the ACM. He received the National Science Foundation's CAREER Award in 2001 in the area of design automation of electronic systems, two Best Paper Awards from the Design, Automation, and Test in Europe Conference in 2001 and 2003, and a Best Paper Award from the Asia South Pacific Design Automation Conference in 2003. He was also awarded the Carnegie Institute of Technology's Ladd Research Award (2002–03) in recognition of his research, professional accomplishments, and potential.