

# Energy- and Performance-Aware Incremental Mapping for Networks on Chip With Multiple Voltage Levels

Chen-Ling Chou, *Student Member, IEEE*, Umit Y. Ogras, *Student Member, IEEE*, and Radu Marculescu, *Senior Member, IEEE*

**Abstract**—Achieving effective run-time mapping on multiprocessor systems-on-chip (MPSoCs) is a challenging task, particularly since the arrival order of the target applications is not known *a priori*. This paper targets real-time applications which are dynamically mapped onto embedded MPSoCs, where communication happens via the Network-on-Chip (NoC) approach, and resources connected to the NoC have multiple voltage levels. We address precisely the energy- and performance-aware incremental mapping problem for NoCs with multiple voltage levels and propose an efficient technique (consisting of region selection and node allocation) to solve it. Moreover, the proposed technique allows for new applications to be added to the system with minimal interprocessor communication overhead. Experimental results show that the proposed technique is very fast, and as much as 50% communication energy savings can be achieved compared to using an arbitrary allocation scheme.

**Index Terms**—Low-power design, multiprocessor interconnection, networks on chip (NoCs), optimization methods, real-time systems.

## I. INTRODUCTION

NETWORKS on chip (NoCs) emerged as a novel communication paradigm for systems on chip [2], [3]. The NoC solution brings a networking approach to on-chip communication and provides notable improvements in terms of performance, scalability, and flexibility over the traditional bus-based or more complex hierarchical bus structures (e.g., Advanced Microcontroller Bus Architecture and STBus) [1]. In general, the NoC architecture consists of multiple heterogeneous processors (in this paper, resources and processors are used interchangeably) and storage elements interconnected via a packet switched network. The “resources” can be an application-specific processing element (PE), digital signal processor, a general-purpose processor, *etc.* For this communi-

cation architecture, the issue of managing the system resources and optimizing the overall performance becomes crucial.

In this paper, we target real-time applications described as task graphs as opposed to general-purpose best-effort applications usually found in chip multiprocessors (CMPs). These applications are mapped onto embedded multiprocessor systems on chip (MPSoCs), where the basic architecture consists of homogenous PEs operating at multiple voltage levels. More precisely, we assume that only the PEs connected to the NoC have multiple voltage levels, whereas the network itself (including links, routers, *etc.*) is in its own voltage–frequency domain. A global manager (GM) is responsible for system resource management, which involves the mapping of the incoming applications to the available PEs and handling the interprocessor communication. Since the arrival order and execution times of the applications are not known at design time (i.e., applications arrive at arbitrary times and leave the system after being executed), performing effective *run-time mapping* is an important and challenging task. Toward this end, we propose a run-time mapping technique which allocates the appropriate resources to the incoming application tasks such that the communication energy is minimized, given some deadline constraints. At the same time, all the preexisting applications still run on the initial set of resources they have been allocated to.

To illustrate the proposed methodology, we assume a system architecture with two voltage levels, as shown in Fig. 1. The gray squares represent the PEs operating at higher voltage levels, while the black dots show the tasks belonging to a preexisting application which cannot be reallocated. Applications App 1 and App 2 shown in Fig. 1(a) and (b), respectively, need to be mapped sequentially to the initial system configuration in Fig. 1(c). Suppose that nodes 4 and 6 are the critical nodes for App 1 while node 2 is the critical node for App 2, this means that they must be allocated to the PEs operating at the highest voltage level in order to meet the application deadlines.

After the arrival of each new application App  $i$ , a greedy approach would map App  $i$  to the NoC resources such that the interprocessor communication cost of App  $i$  is minimized for the *current* configuration (i.e., ignoring any future arrivals). In this simple example, the total system communication cost is the sum of the communication cost of all applications, i.e.,

$$\text{System communication cost} = \sum_{\forall \text{App}} \sum_{\forall (i,j)} \text{MD}(\nu_i, \nu_j)$$

Manuscript received November 30, 2007; revised April 21, 2008. Current version published September 19, 2008. This work was supported by the Gigascale Systems Research Focus Center, which is one of the five research centers funded under the Focus Center Research Program, which is a Semiconductor Research Corporation program. Parts of this paper appeared as “Incremental Run-Time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels” in the Proceedings of the Hardware/Software Code-sign and System Synthesis (CODES+ISSS), Salzburg, Austria, October 2007, pp. 161–166. This paper was recommended by Associate Editor L. Benini.

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

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCAD.2008.2003301

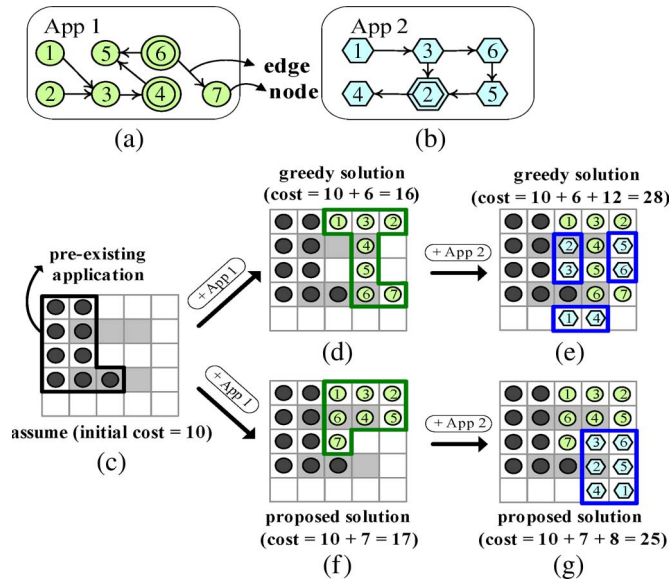


Fig. 1. Example of NoC incremental application mapping comparing the greedy and our proposed solutions. The greedy approach, which does not consider additional mappings, incurs higher communication overhead for App 2, and the system communication cost as well, compared to our proposed solution.

where  $MD(v_i, v_j)$  represents the Manhattan distance (MD) between any two application nodes  $v_i$  and  $v_j$  connected to each other. As shown in Fig. 1(d), even though the greedy approach minimizes the communication cost for the current configuration, the newly generated region consisting of the remaining (available) PEs is quite scattered. Consequently, mapping any additional application onto this configuration would be ineffective, as it can be seen for the noncontiguous region of App 2 in Fig. 1(e).

As opposed to this, our newly proposed methodology *does* consider applications that may arrive to the system at future times, and consequently, it offers a more effective mapping in the presence of dynamically incoming applications. Indeed, as shown in Fig. 1(g), when App 2 is mapped after App 1, the system communication cost becomes smaller than the cost obtained using the greedy approach. Intuitively, since the pre-existing applications cannot be reallocated, the performance of the greedy solution becomes much worse compared to our proposed solution.

Note that the task migration approach is complementary to our incremental mapping; indeed, task migration is an effective strategy to achieve load balancing and high resource utilization. For distributed systems without shared memory support, the task migration policy must be implemented by passing messages among PEs; the implicit migration cost is large due to the need of moving the process context [18]. However, for embedded MPSoCs with shared memory, we have two contexts to worry about from a migration perspective: The user context (called the remote) and the system context (called the deputy or home node). Only the user context (i.e., stacks, memory maps, and registers of the process) needs to be migrated, while the system context is kept either on the home node or in the shared memory. Therefore, the migration process can be implemented with middleware support on top of the operating system (OS).

In this paper, we do not focus on the task migration process. Instead, we target an incremental mapping process which does not need to change the current system configuration.

In summary, the novel contribution of this paper consists of a new approach for dynamic application mapping such that the total communication energy consumption in the system is minimized. At the same time, additional applications can be easily added to the resulting system with minimal communication cost overhead.

The rest of this paper is organized as follows. First, we review the prior work (Section II) and present the platform description and the run-time mapping methodology (Section III). In Section IV, we formulate the problem of run-time incremental mapping. Then, we propose a two-step algorithm to solve this problem; more precisely, the near convex region selection problem is discussed in Section V, while the node allocation problem is addressed in Section VI. The experimental results appear in Section VII, while Section VIII summarizes our main contribution.

## II. RELATED WORK

Resource allocation is a fundamental problem encountered in a variety of areas, including processor allocation for supercomputers and task assignment in massively parallel processing systems. While dealing with the resource management process, Karp *et al.* in [21] study the problem of finding the shape of a region assigned to tasks for minimizing the pairwise distance of all points within that region; they observe that there is no closed-form solution for getting the optimal region. Bender *et al.* in [20] express the solution as a differential equation to solve the resource allocation problem and provide a theoretical proof for getting the optimal solution. Bender *et al.* in [17] present an approximate algorithm for selecting processors such that to minimize the average number of communication hops in supercomputers.

In terms of the offline resource allocation problem for NoC, several approaches have been proposed. Hu and Marculescu [4] propose a branch and bound algorithm to map IP cores onto a tile-based NoC architecture, while satisfying the bandwidth constraints and minimizing the total communication energy consumption. The work in [5] considers the mapping problem for minimizing the communication delay with split routing.

While dealing with the resource management problem for “multiple applications” in the system, Pop *et al.* present an approach to incremental design of distributed systems for hard real-time applications over a bus [14]. More recently, Murali *et al.* proposed a methodology for mapping multiple use cases onto NoCs, where each use case has different communication requirements and traffic patterns [7].

In terms of the online resource management for NoCs, the techniques proposed so far rely on a resource manager operating under OS control [13]. This OS-controlled mechanism allows the system to operate effectively in a dynamic manner. Smit *et al.* [6] propose a run-time task assignment algorithm on heterogeneous processors. However, the task graphs are restricted to have either a small number of tasks or a task degree of no more than two. More recently, Carvalho *et al.* propose

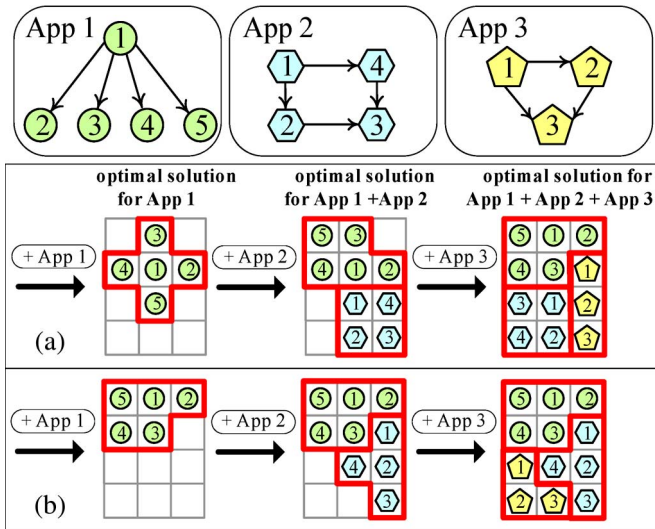


Fig. 2. Motivational example for incremental mapping process. (a) Optimal solution. (b) Near convex region solution.

dynamic task mapping scheme in NoC-based heterogeneous MPSoCs, targeting the channel load minimization for improving the performance [22].

As such, all the previous works mentioned earlier do *not* maximize the system efficiency by considering the possible addition of new applications and NoC-based communication. In this paper, our goal is to optimize the communication energy consumption for *all* possible system configurations (at different time instances), considering that applications can dynamically arrive and leave the system.

### III. PRELIMINARIES

#### A. Motivational Example

We illustrate the incremental mapping process using three applications. For simplicity, the system considered in this example has only a single voltage level. As shown in the optimal mapping solution in Fig. 2(a), whenever a new application arrives in the system, we minimize the average communication distance for the incoming application and *all* existing applications in the system. Applying the optimal mapping in practice would be infeasible since the run time for deciding the configuration which gives the optimal solution and reconfiguring the previous applications by migrating tasks is too high. However, we can get a significant insight from analyzing the results produced by an optimal solution. Indeed, by looking at Fig. 2, we observe that each application tends to cover a convex region, while the PE utilization of the system increases. Therefore, if no task migration is allowed, allocating an incoming application to a region which looks as convex as possible helps minimize the communication overhead for any additional incoming application [see our proposed solution in Fig. 2(b)].

In general, a region is convex if it contains all the line segments connecting any pair of points *inside* it. Bender *et al.* [20] define the region to be optimal if the average distance between all pairs of points is a minimum; as such, they expect the shape of an optimal region to be convex. However, the concept of *near*



Fig. 3. Homogeneous 2-D mesh NoCs with PEs having multiple voltage/frequency levels and interconnect via the data and control networks.

*convex region* we use in this paper is slightly more general; it stands for a region whose area is *close* to the area of its convex hull [15]. The key goals in our approach for selecting a near convex region are the following: 1) Minimize the average communication distance (i.e., number of hops) between the processors assigned to the tasks of the currently incoming application and 2) minimize the noncontiguous regions which may incur a higher communication cost if additional applications are mapped onto them. Consequently, our problem formulation generalizes the optimal region considerations [20] for dynamic system configurations with limited resources.

#### B. Target NoC Architecture

The target NoC architecture consists of identical PEs interconnected by a 2-D  $n \times n$  mesh network. Each PE consists of a processing unit, local memory, and control unit, as shown in Fig. 3. The PEs operate at fixed voltage and frequency levels, which are selected from a finite set  $(V_i, f_i)$ . When the voltage level of a PE is different from that of the network, mixed-clock first-in–first-out (FIFO) need to be utilized. We also assume that the voltage/frequency assignment for PEs (or voltage island partitioning problem) is already determined using an approach similar to the one presented in [16].

As seen in Fig. 3, a GM is included in this architecture. The GM runs the proposed run-time mapping algorithm for allocating the incoming tasks to the appropriate resources. We note that, for large NoCs, a hierarchical control mechanism may be needed. Moreover, we have two types of networks in this architecture, namely, the *data* and *control* networks. The *data* network is responsible for delivering data packets among PEs. The *control* network (i.e., the routers and links represented by dotted lines in Fig. 3) is used to move around the control messages. The data and control networks are separated to ensure that data in the data network do *not* interfere with the control messages in the control network.

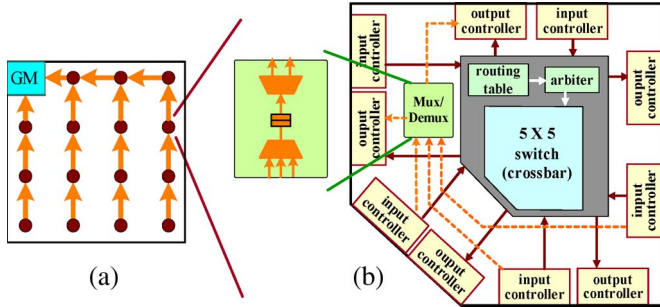


Fig. 4. (a) Logical view of the control network. (b) On-chip router microarchitecture that handles the control network.

TABLE I  
IMPACT OF ADDING THE CONTROL NETWORK ON AREA. THE SYNTHESIS IS PERFORMED FOR XILINX VIRTEX-II PRO XC2VP30 FPGA

	# of slices
one router in ‘pure’ data network	392
one router in our proposed network	401
area overhead	2.3%
4 × 4 mesh network with ‘pure’ data network	6737
4 × 4 mesh network with our proposed network	6891
area overhead	2.2%

*Operation of the GM and the Control Network:* The task of the GM is to continuously track the status of the PEs (idle/available or used/unavailable) in the system. When an incoming application  $Q$  enters the system, the GM runs our incremental mapping process and makes the run-time decision for the incoming application  $Q$ . After the mapping decision is taken, the necessary resources are allocated to the tasks of this incoming application, and the application starts executing. Once the application  $Q$  finishes its execution and leaves the system, the PEs assigned to the application  $Q$  send their address back to the GM through the control network to notify the GM that they become available.<sup>1</sup> Therefore, we do not need a fully connected network but just a tree that accumulates the messages for the GM,<sup>2</sup> as shown in Fig. 4(a).

*Design of the Control Network:* As shown in Fig. 3, the control network has limited connectivity requirements, and it is *physically separated* from the data network. More precisely, these two networks do *not* share any circuitry, such as links or buffers.

*Area Overhead of Our Proposed Network:* In terms of implementation, we employ the router described in [23] for the data network. In order to evaluate the overhead of the control network, we add extra buffers and a multiplexer/demultiplexer pair to the existing router used for data network [see Fig. 4(b)]. After that, we implemented a 4 × 4 mesh network using both the original and modified routers. Finally, the designs are synthesized using Xilinx Integrated Software Environment to evaluate the area overhead (see data in Table I).

<sup>1</sup>This way, the GM always knows the status of the PEs in the system and can take further decisions for new applications.

<sup>2</sup>This structure is equivalent to a broadcast tree. We can obtain the broadcast tree by reversing the direction of all edges in Fig. 4(a).

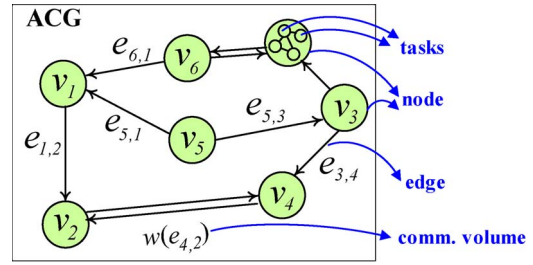


Fig. 5. ACG characteristics. The tasks belonging to the same node are mapped onto the same PE. Each edge represents the communication between two nodes.

*Energy Overhead of the Control Messages in the Control Network:* In terms of the energy overhead for delivering the control messages, we utilize the bit energy model, which is the same metric used when dealing with data messages. As mentioned before, the control network is used only to send the status information from PEs to the GM. This status information includes the address of the PE and an extra bit of information showing the PE is busy or idle. Therefore, the size of control messages is dependent on the network size. For an  $M \times M$  NoC, we only need  $\lceil \log_2(M^2) \rceil$  bits to decode the addresses of the PEs. Obviously, the volume of the control messages is much smaller than the volume of the data messages. Moreover, the control network is much simpler than the data network, as described before. Consequently, the control network is expected to have significantly smaller energy consumption compared with the data network. Indeed, if the energy consumption to send the information from PEs to the GM is comparable to the data communication, then a more sophisticated hierarchical control mechanism is more suitable.

### C. Application Characteristics

The incoming applications are described by the application communication graph (ACG), which is generated using an offline technique like in [11] and [12]. Each  $ACG = G(V, E)$  (see Fig. 5) is a directed graph and contains the following.

- 1) *Nodes.* Each node  $\nu_k \in V$  contains a set of tasks obtained from an offline task partitioning process [11], [12]. The tasks belonging to the same node are allocated to the same idle PE. Note that, while dealing with the offline task partitioning process, in order to meet the application deadline, we use the worst case (WC) communication time for communications between nodes (i.e., longest communication path). Moreover, we use the WC execution time (WCET) for data-dependent tasks, where the WCET of a task is the maximum length of time that the task takes to execute on a specific hardware platform, i.e., PE operating at a certain voltage.
- 2) *Edges.* Each edge  $e_{i,j} \in E$  represents the internode communication from  $\nu_i$  to  $\nu_j$ , where node  $\nu_i$  is any neighbor of node  $\nu_j$ . Weights  $w(e_{i,j})$  characterize the communication volume (bits) between nodes  $i$  and  $j$ .
- 3)  $M(\nu_k)$ . Each node  $\nu_k$  has its *minimum voltage*  $M(\nu_k)$  at which it should operate in order to meet the application deadlines. Since the maximum operating frequency for a given voltage level is selected, the higher voltage levels

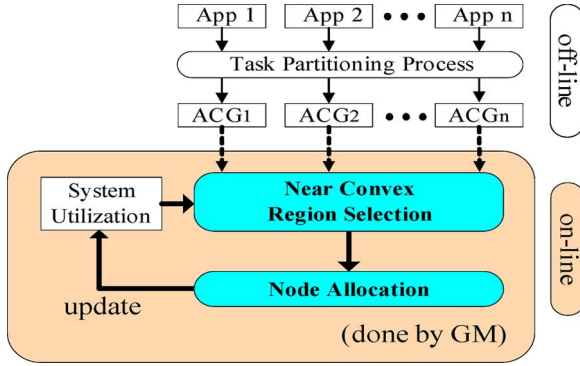


Fig. 6. Overview of the proposed methodology.

result in faster execution, assuming that the task execution times (clock cycles) remain the same.

- 4)  $S_i$ . All nodes in the node set  $S_i$  will be allocated to the PEs with the voltage level  $i$  later.  $|S_i|$  is the node set size, i.e., the number of nodes in that set.

#### D. Energy Model

We model the energy consumption of the network using the bit energy metric proposed in [8]. Suppose that the source ( $PE_s$ ) and destination ( $PE_d$ ) operate at static voltage/frequency levels  $(V_i, f_i)$  and  $(V_j, f_j)$ , respectively, while the entire network uses a (constant) voltage/frequency level  $(V_n, f_n)$ . The average energy consumption to send one bit of data from  $PE_s$  to  $PE_d$  is as follows:

$$E_{\text{bit}}(s, d) = E_{R_{\text{bit}}(V_i \rightarrow V_n)} + E_{\text{Network}}(s, d) + E_{R_{\text{bit}}(V_n \rightarrow V_j)} \quad (1)$$

where the  $E_{R_{\text{bit}}(V_i \rightarrow V_n)}$  and  $E_{R_{\text{bit}}(V_n \rightarrow V_j)}$  terms represent the energy consumed while transferring data between the network and the processors operating at different voltage levels. For instance, when  $PE_s$  operates at  $V_i$  and the network operates at  $V_n$ , the energy consumption of sending one bit of data from the processor to the network is  $E_{R_{\text{bit}}(V_i \rightarrow V_n)}$ . On the other hand, the  $E_{\text{Network}}(s, d)$  is defined as

$$E_{\text{Network}}(s, d) = E_{R_{\text{bit}}(V_n \rightarrow V_n)} \times (\text{MD}(s, d) - 1) + E_{\text{Link}}(s, d) \times \text{MD}(s, d). \quad (2)$$

The routers attached to nodes  $s$  and  $d$  are not included in this equation since they are already accounted for in (1). Finally, the total communication energy consumption is as follows:

$$E_{\text{comm}} = \sum_{\forall e_{i,j}=(s_i,d_j) \in E} w(e_{i,j}) \times E_{\text{bit}}(s_i, d_j). \quad (3)$$

## IV. INCREMENTAL RUN-TIME MAPPING PROBLEM

### A. Problem Formulation

Our proposed methodology is summarized in Fig. 6. All applications are described by ACGs which result from an offline task partitioning similar to [11] and [12]. Our online mapping

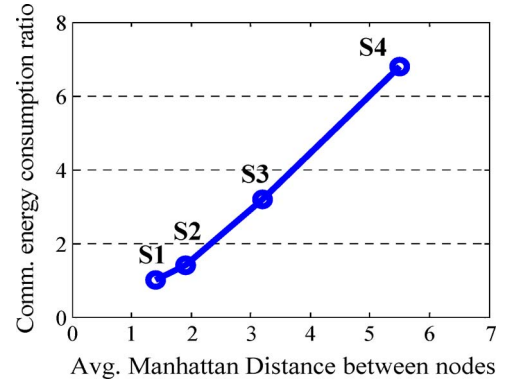


Fig. 7. Impact of MD on communication energy consumption for four different scenarios (S1–S4).

process is activated only when an application arrives in the system. Our objective is to first select a near convex region and then decide on which PE within this region should each node in the ACG be mapped to, such that the communication energy consumption is minimized under given timing constraints. To formulate this problem, we need a few notations as follows.

- 1)  $PE_{ij}$  is the PE located at the intersection of the  $i$ th row and  $j$ th column of the network.  $PE_{11}$  is the GM.
- 2)  $V(PE_{ij})$  is the voltage level that processor  $PE_{ij}$  belongs to.
- 3)  $\text{MD}(PE_{ij}, PE_{kl})$  is the MD between  $PE_{ij}$  and  $PE_{kl}$ .

Using this notation, the problem of dynamic incremental mapping for NoCs can be formulated as follows:

**Given** the current system behavior and the ACG of the incoming application

**Find** a near convex region  $R$  and a node mapping function  $\text{map}()$ ,  $\forall \nu_k \in V$ ,  $\text{map}(\nu_k) \rightarrow PE_{ij}$  in  $R$ , with the objective

$$\min \left\{ \text{Energy} = \sum_{\forall e_{i,j}} w(e_{i,j}) \times \text{MD}(\text{map}(\nu_i), \text{map}(\nu_j)) \right\} \quad (4)$$

**such that**  $\forall \nu_k \in V$ ,  $V(PE_{ij}) \geq M(\nu_k)$ .

### B. Significance of the Problem

To prove that the MD metric in the problem formulation heavily affects the communication energy consumption, we consider the following experiment. An ACG is generated using the task graphs for free (TGFF) package [9]. Then, we implement four scenarios for mapping this application onto an  $8 \times 8$  homogeneous NoC. Scenario 1 (S1) in Fig. 7 uses our method, scenario 2 (S2) uses the nearest neighbor heuristic proposed in [22], scenario 3 (S3) randomly maps the application nodes inside a  $4 \times 4$  rectangle region, and scenario 4 (S4) randomly maps the application nodes onto any PEs in an  $8 \times 8$  NoC. The  $x$ -axis in Fig. 7 represents the average MD between two nodes, while  $y$ -axis represents the communication energy consumption normalized to that of the first scenario. As we can see, the MD between application nodes is an effective way to minimize the communication energy consumption of the applications.

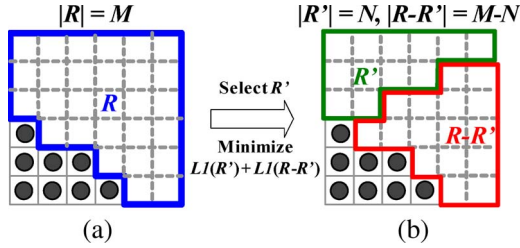


Fig. 8.  $L_1(R') + L_1(R - R')$  minimization problem: Select a region  $R'$ , such that the sum of the total MD between any pair of tiles inside regions  $R$  and  $R - R'$  is minimized.

### V. SOLUTION TO THE NEAR CONVEX REGION SELECTION PROBLEM

When dealing with region selection problem for the incremental mapping process, we need to minimize the communication cost of the incoming application and, at the same time, minimize the communication cost overhead for any additional incoming application. To generalize and formulate this problem, the  $L_1$  distance is defined as follows.

*Definition 1:* The  $L_1$  distance of a region  $R$  with  $N$  tiles, denoted as  $L_1(R)$ , is the total MD between any pair of these  $N$  tiles inside  $R$ .

The scenario in Fig. 8(a) covers the general case of the incremental mapping problem. Such a scenario includes some preexisting applications (i.e., the black circles in Fig. 8) running in the system, as well as a new incoming application which needs to be allocated on the remaining/available PEs [these  $M$  PEs in the thick line region  $R$  are shown in Fig. 8(a)].

Assume that this incoming application requires  $N$  PEs (with  $N < M$ ). Our objective is to find a subregion  $R'$  with  $N$  PEs to assign to this incoming application which minimizes the metric  $L_1(R') + L_1(R - R')$ , as shown in Fig. 8(b). Intuitively, it is difficult to consider these two terms  $L_1(R')$  and  $L_1(R - R')$  at the same time. In Section V-A, we first focus on minimizing the first term  $L_1(R')$  (we call this the  $L_1$  problem), which is a special case of the general allocation problem. This gives us an insight into the problem of minimizing  $L_1(R') + L_1(R - R')$ , which is discussed in Section V-B. Finally, in Section V-C, the region selection algorithm is proposed for NoC platforms with multiple voltage levels.

#### A. Minimization of $L_1(R')$

Minimizing  $L_1(R')$  for a region  $R'$  with  $N$  tiles is the special case of the general allocation problem in [17]. To find a lower bound for  $L_1(R')$ , we first implement the best case (BC) solution to conjecture the best shape of the region. Then, the WC solution in a contiguous region for this problem is derived as the upper bound. Note that, since the incremental mapping process is done online, we need to look for near-optimal solutions with very low cost (i.e., low computation time). Therefore, we also propose four suboptimal solutions to see if any lower cost solution gets close enough to the optimal case.<sup>3</sup>

<sup>3</sup>Note that neither the BC algorithm nor the suboptimal solutions have known closed form formula in terms of  $N$ . Therefore, we can only obtain the optimal result from exhaustive search and the results for suboptimal solutions from simulation.

Fig. 9 plots one region (with  $N = 20$ ) generated by each of the six cases which include two possible extreme cases, the BC and WC, and our proposed solutions [Euclidian minimum (EM), fixed center (FC), random frontier (RF), and Neighbor-aware Frontier (NF)].

- 1) **BC.** This corresponds to the optimal solution generated by an exhaustive search, but obviously, this works only for moderate values of  $N$ .
- 2) **WC.** The closed-form solution for the worst case of a contiguous region  $R$  with  $N$  tiles

$$L_1^{(\text{worst})}(R) = \frac{N \times (N - 1) \times (N + 1)}{6}. \quad (5)$$

*Proof:* The WC of  $L_1(R)$  with  $N$  tiles inside  $R$  is to place the  $N$ th tile to the resulting region with distance  $N - k$  from the  $k$ th tile, i.e.,

$$\begin{aligned} L_1^{(\text{worst})}(R) &= (1) + (1 + 2) + \dots + (1 + 2 + \dots + N - 1) \\ &= (1) \times (N - 1) + (2) \times (N - 2) + \dots \\ &\quad + (N - 1) \times (1) \\ &= \sum_{i=1}^{N-1} i \times (N - i) = \frac{N \times (N - 1) \times (N + 1)}{6}. \end{aligned}$$

- 3) **EM.** While adding the  $N$ th tile into the region, the EM heuristic updates the center  $(x_c, y_c)$  by recalculating the arithmetic mean of  $N - 1$  tiles and then selects a tile  $(x, y)$  with minimum Euclidean distance  $\sqrt{|x - x_c|^2 + |y - y_c|^2}$  to the updated center.
- 4) **FC.** The first tile of the region is always set as the FC  $(x_c, y_c)$ . While adding the  $N$ th tile into the region, the FC heuristic selects a tile  $(x, y)$  with minimum MD  $|x - x_c| + |y - y_c|$  to the FC.
- 5) **RF.** While adding the  $N$ th tile into the region, the RF heuristic randomly selects a tile from the frontier of the region consisting of  $N - 1$  tiles.
- 6) **NF.** Every tile has four neighbors. The tile is considered to be available if it has not been selected into the region. While adding the  $N$ th tile into the region, the NF heuristic searches the frontier of the resulted region and then selects a tile with a minimal number of available neighbors.

We should note that there exists more than one solution for these four cases (EM, FC, RF, and NF), even for the BC and WC scenarios. Fig. 9 shows a few concrete instances of regions generated by each of the six cases for  $N = 20$ . The numbers on tiles in Fig. 9(b)–(f) represent the selection order when forming the regions. We initially set the tile (5, 5) as the first tile of the region.

As can be seen in Fig. 9(a), the resulting shapes of BC are almost “circular.” However, this solution cannot be applied to run-time incremental mapping due to its high run-time overhead. For example, it takes more than 40 min to get the optimal solution for  $N = 20$  while running on an Intel Pentium 4 CPU with 2.60 GHz. On the contrary, the EM, FC, and RF heuristics take less than 10  $\mu$ s. We observe that, for EM and FC cases, they

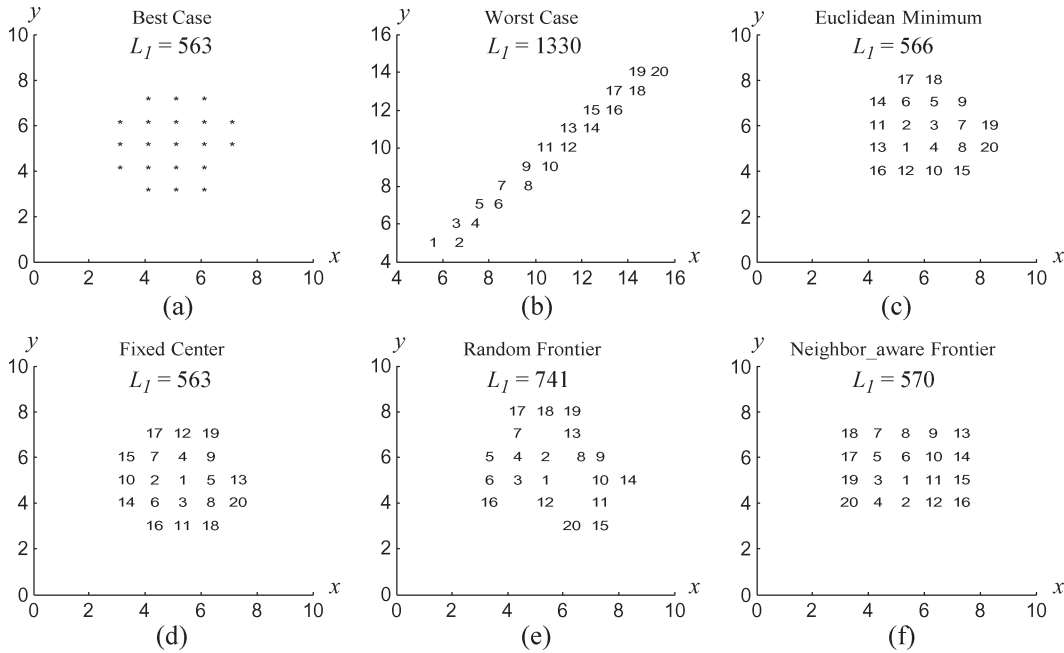


Fig. 9. Region with  $N = 20$ , resulting from several distinct methods, namely, (a) BC, (b) WC, (c) EM, (d) FC, (e) RF, and (f) NF. Note that the shape of the resulted regions would be the same even if shifted to other coordinates. Here, we only consider minimizing the total MD between any pair of these  $N$  tiles inside  $R'$ , i.e.,  $L_1(R')$ .

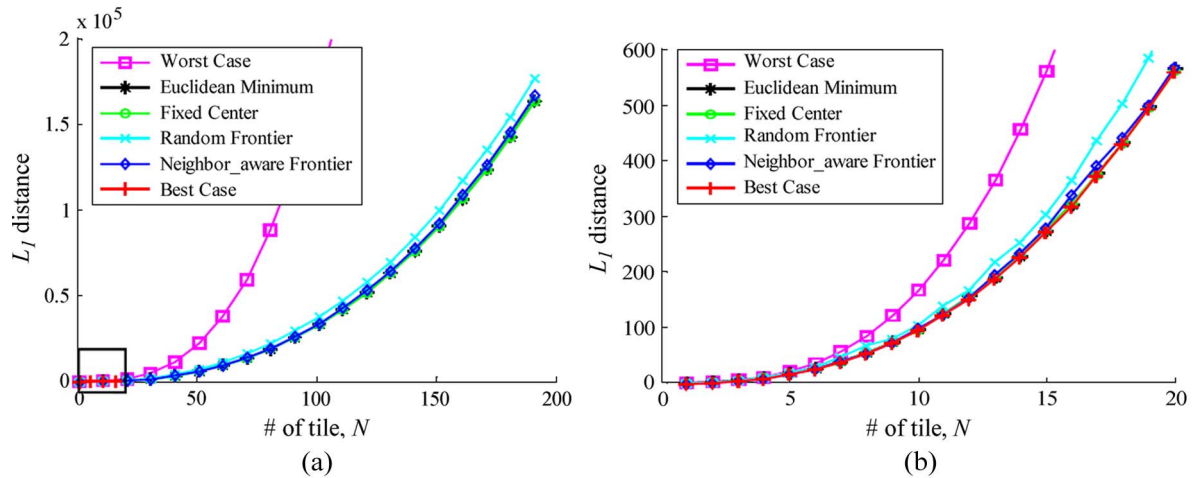


Fig. 10.  $L_1$  distance results showing the scalability of the solutions obtained via the BC, WC, and four heuristics (EM, FC, RF, and NF).

differ by less than 1% compared with the optimal solution [see Fig. 9(c) and (d)]. For the RF case, there may exist holes inside the region, and this would greatly increase the  $L_1$  distance [e.g., for  $N = 20$ , 31.6% increase of the  $L_1$  distance compared with the optimal solution in Fig. 9(e)]. In order to reduce the probability of getting holes inside a region, we propose the NF heuristic, which includes the neighbors' information. Indeed, the solution produced by the NF for  $N = 20$  is only by 1.24% away compared with the optimal solution, and it can be obtained within 5  $\mu$ s.

To show the scalability of these heuristics, we need to test regions containing a large number of tiles. The simulation results for the  $L_1$  distance under BC, and the WC scenarios and these four heuristics (EM, FC, RF, and NF) are shown in Fig. 10(a). We plot the  $L_1$  distance from running 1000 experiments with  $N$  varying from 1 to 200 [see Fig. 10(a)]. Since it takes more than

40 min to get the result for the BC if  $N = 20$ , we do not report results for BC scenario when  $N$  is greater than 20 tiles. We do show, however, results for BC when  $N$  varies from 1 to 20 [see Fig. 10(b)].

From Fig. 10(a), we observe that the results obtained from EM and FC cases are close to each other. Of note, the NF heuristic has only 2.63% increase of the  $L_1$  distance compared to the FC heuristic for  $N = 200$ . Moreover, from Fig. 10(b), EM, FC, and NF cases are close to the optimal solution (i.e., the BC scenario) for  $N$  varying from 1 to 20.

### B. Minimization of $L_1(R') + L_1(R - R')$

Now, we have a better sense about minimizing  $L_1(R') + L_1(R - R')$ . Considering the configuration in Fig. 8(a), assume that the new incoming application has 12 nodes, i.e.,

TABLE II  
 $L_1(R') + L_1(R - R')$  MINIMIZATION PROBLEM WHEN USING THE EM, FC, AND NF HEURISTICS

Heuristics	$L_1(R') + L_1(R - R') = \text{distance sum}$	Standard deviation/mean	Min(distance sum)	Max(distance sum)
Euclidean Minimum (EM)	155.190 + 426.076 = 581.266	67.095/581	504	696
Fixed Center (FC)	159.086 + 404.782 = 563.868	55.503/564	502	672
Neighbor_aware Frontier (NF)	167.604 + 342.268 = 509.872	14.544/510	498	568

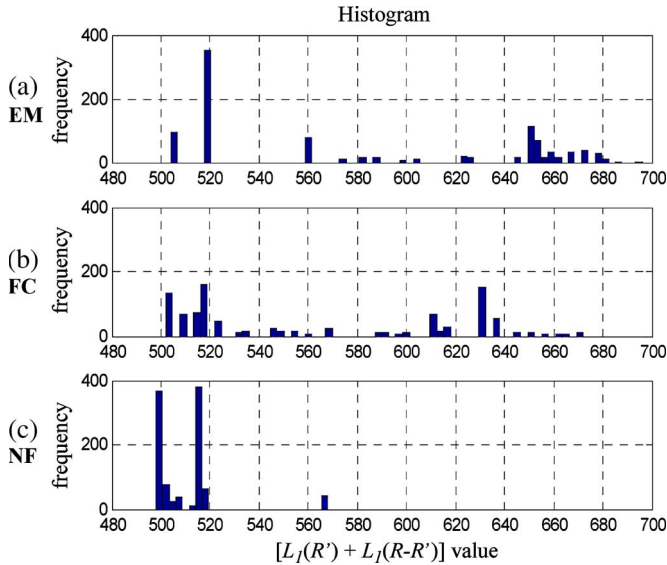


Fig. 11. Histogram over 1000 runs for  $L_1(R') + L_1(R - R')$  minimization problem. We represent  $[L_1(R') + L_1(R - R')]$  distances on the  $x$ -axis and their frequency of occurrence on the  $y$ -axis.

$|R'| = 12$ . To minimize  $L_1(R') + L_1(R - R')$ , we implement the EM, FC, and NF heuristics described earlier. The first tile of each approach is selected from any boundary tile of  $R$ . Note that compared with the problem discussed in Section V-A, the  $L_1(R') + L_1(R - R')$  minimization problem has one obvious limitation: the boundary constraint. Under this limitation, for the EM, FC, and NF cases, we need to add one more constraint, i.e., the grid which is selected should be *inside* the region  $R$ . In addition, for the NF case, since the boundary condition greatly influences the neighboring information, we need additional modifications. Initially, every tile in the grid has four neighbors, except the corner tiles which have only three neighbors. Other steps are the same as in Section V-A.

Fig. 11 shows the histogram of  $[L_1(R') + L_1(R - R')]$  values derived from 1000 runs for each heuristic (EM, FC, and NF). For example, in Fig. 11(c) which uses the NF approach, we can get the  $[L_1(R') + L_1(R - R')]$  distance equal to 498 for 369 times. Of note, neither EM nor FC can obtain this small value. We also summarize these data in Table II, which lists the  $L_1$  distance of the selected region  $R'$  and the remaining region  $R - R'$  for an average of 1000 runs. Moreover, we list the standard deviation over the mean of  $L_1(R') + L_1(R - R')$  in 1000 runs and the best/worst results for each heuristic.

From Fig. 11 and Table II, we observe that the EM and FC do not work well for solving the  $L_1(R') + L_1(R - R')$  minimization problem. As seen in Table II, even though the value of  $L_1(R')$  of EM and FC heuristics is quite small, the

pairwise distance *outside* the region, namely,  $L_1(R - R')$ , is relatively high, i.e., the selected region does not help for additional mappings. On the contrary, the NF with the neighboring information included helps for the additional mappings (the decrease in  $L_1(R - R')$  distance is 19% and 15% compared to the EM and FC, respectively). Even if the distance  $L_1(R')$  of the NF is about 8% and 5% larger than that of the EM and FC, respectively, the total distance  $L_1(R') + L_1(R - R')$  of the NF is still 10% less than the solutions provided by the EM and FC. In addition, when observing the NF in Fig. 11(c), we have a higher probability to get the smaller  $L_1(R') + L_1(R - R')$  value compared to EM and FC heuristics shown in Fig. 11(a) and (b). This matches the goals of the incremental mapping process, namely, to minimize the interprocessor communication cost for the incoming application (i.e., smaller  $L_1(R')$ ) and easily add additional applications to the resulting system with minimal interprocessor communication overhead [i.e., smaller  $L_1(R - R')$ ].

C. Solution to the Region Selection Problem for Run-Time Incremental Mapping Process

From the discussion in Sections V-B and V-C, we decide to apply the NF heuristic to the region selection problem (see Fig. 6). Since, for the aforementioned discussion, all grids were considered to be the same (i.e., homogeneous system), we need to define two new terms in order to deal with the heterogeneity of our proposed platform (see Section III-B).

- 1) *Dispersion factor (D)*. The dispersion factor of a PE,  $D(\text{PE})$ , is defined as

$$D(\text{PE}) = C - \text{number of utilized neighbors of that PE} \tag{6}$$

where  $C$  is a constant. For the corner PEs,  $C = 3$ ; for other PEs inside (including the boundary),  $C = 4$ .

The PEs with the smaller  $D(\text{PE})$  value indicate a higher likelihood to be included into the current region. Indeed, a PE that has most of its neighbors utilized [i.e., PE with a small  $D(\text{PE})$  value] is very likely to be later isolated; then, selecting this PE for the current region helps reduce its dispersion probability.

- 2) *Centrifugal factor (C)*. The PE centrifugal factor  $C(\text{PE})$  is defined as the MD between any PE and the border of the current region. PEs with the smaller  $C(\text{PE})$  value indicate a high likelihood to be included into the current region. Indeed, since every PE in a near convex region should be close to the borders of that region, the PE with smaller

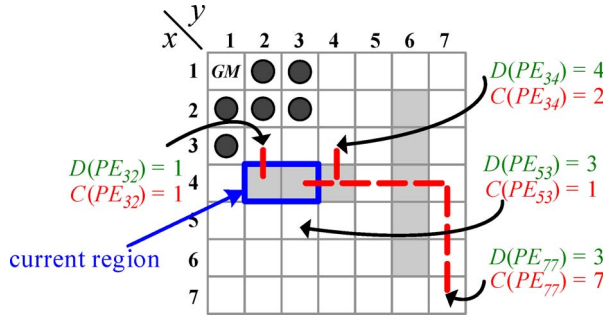


Fig. 12. Dispersion and centrifugal factor calculation example.

Step 1): Assign each node  $v$  to  $S_i$  where  $i$  is greater than  $M(v)$ , and sort them out in non-decreasing order,  $|S_1| \leq |S_2| \leq \dots \leq |S_k|$ .  
 Step 2): Start with  $S_1$ , select a  $PE_{ij}$  with minimum code transfer energy consumption and include it into the region.  
 Step 3): Update the  $D(PE)$  and  $C(PE)$  for unselected and idle PEs of that set. Select  $PE_{ij}$  with lowest  $D(PE_{ij}) + C(PE_{ij})$  in the region. Continue with Step 3 until the number of PEs in the selected region matches the size of this set.  
 Step 4): Repeat Step 3 for the remaining sets.

Fig. 13. Near convex region selection algorithm.

$C(PE)$  is better suited for selection to form a near convex region.

Examples of calculation of the dispersion and centrifugal factors are shown in Fig. 12.  $PE_{12}$ ,  $PE_{13}$ ,  $PE_{21}$ ,  $PE_{22}$ ,  $PE_{23}$ , and  $PE_{31}$  are running preexisting applications and considered to be unavailable. The current region is framed with thicker lines. We demonstrate how to select and bring PEs into the current region while keeping its shape as convex as possible. In Fig. 12, four PEs ( $PE_{32}$ ,  $PE_{34}$ ,  $PE_{53}$ , and  $PE_{77}$ ) are selected as examples for calculating  $D(PE)$  and  $C(PE)$ . For  $PE_{32}$ ,  $D(PE_{32}) = 1$  since its neighbors  $PE_{22}$ ,  $PE_{31}$ , and  $PE_{42}$  are unavailable, while  $C(PE_{32}) = 1$  since it has an MD of one to the region boundary. Other three PEs with dispersion and centrifugal factor calculation are shown in Fig. 12. Since PE with minimum  $D(PE) + C(PE)$  value indicates a higher likelihood to form a near convex region, under the current region,  $PE_{32}$  is more likely to be selected to become part of the region compared with  $PE_{53}$ ,  $PE_{34}$ , and  $PE_{77}$ . The steps of region selection (similar to maze routing [19]) are used in Fig. 13 (assuming  $k$  voltage levels in the system and  $m_k$  is the number of available PEs in the  $k$ th voltage level).

We now consider a simple example and describe our approach step by step. The ACG of the incoming application and system behavior are given in Fig. 14(a) and (b), where the black dots show the preexisting applications in the system. The number marked on each PE (e.g., {3} on  $PE_{32}$ ) in Fig. 14(c) represents the selection order in forming a near convex region. PEs with the same number show that they are selected into the region at the same time.

We can see from Fig. 14(a) that there are two nodes ( $\nu_4$  and  $\nu_6$ ) with  $M(\nu_4) = M(\nu_6) = "H,"$  which are supposed to be mapped onto the PEs at high ("H") voltage level; the other nodes can be mapped onto the PEs in "L" voltage level, namely,  $|S_H| = 2 (S_H = \{\nu_4, \nu_6\})$  and  $|S_L| = 7$  (Step 1). Let us start

with  $S_H$  (Step 2), and assume that  $PE_{42}$  is selected first to become part of the region for minimizing the code transfer energy consumption [Fig. 14(c)]. Then,  $PE_{43}$  is the second node selected for the region (Step 3) since it has the lowest  $D(PE_{43}) + C(PE_{43}) = 3 + 1 (D(PE_{43}) = 3,$  because  $PE_{33}$ ,  $PE_{44}$ , and  $PE_{53}$  are all idle. Comparing this to  $D(PE_{44}) + C(PE_{44}) = 4 + 2$  or  $D(PE_{46}) + C(PE_{46}) = 4 + 4$  implies that  $PE_{43}$  gets selected. Step 3 terminates since the PEs in  $S_H$  are all selected inside the region. Now, we deal with the selection of  $S_L$  (Step 4). Going back to Step 3,  $PE_{32}$  is selected since it has the lowest  $D(PE) + C(PE) = 1 + 1$ . After that,  $PE_{33}$  is selected with  $D(PE_{33}) + C(PE_{33}) = 1 + 1$ , and then,  $PE_{41}$  is selected with  $D(PE_{41}) + C(PE_{41}) = 2 + 1$ . With the same rule,  $PE_{51}$ ,  $PE_{52}$ , and  $PE_{53}$  are selected with the lowest  $D(PE) + C(PE) = 4$ . Finally,  $PE_{61}$  is randomly selected among  $PE_{61}$ ,  $PE_{62}$ ,  $PE_{63}$ ,  $PE_{34}$ , and  $PE_{54}$ , with all of them getting the same value of  $D(PE) + C(PE)$ .

#### D. Complexity of the Region Selection Algorithm

In order to determine the time complexity of the region selection algorithm, assume that the ACG =  $(V, E)$  and the system contains a total of  $n \times n$  PEs organized in  $k$  voltage levels, where  $|V| < n^2$ . Therefore,  $|S_1| + |S_2| + \dots + |S_k| = |V|$ , where  $S_i$  is the size of node set which is about to be selected for the  $i$ th voltage level, and  $m_1 + m_2 + \dots + m_k \leq n^2$ , where  $m_i$  is the number of available PEs in the  $i$ th voltage level.

In Step 1 of Fig. 13, calculating the size of PE sets takes  $O(V)$  time, and the run time for sorting them is  $O(V \log V)$  if using QUICKSORT. Steps 2–4 need  $O(m_1^2 + m_2^2 + \dots + m_k^2)$  since, in Step 3, we need to update  $D(PE) + C(PE)$  for each PE in a certain set, which takes linear time. The worst case scenario occurs when only one PE is selected into the region each time. Thus, the total run time of region selection algorithm is  $O(n^4 + V \log V)$ , i.e.,  $O(n^4)$  since  $V \log V < n^2 \log n^2 < n^4$ . However, in Steps 3–5, we can record the frontier of the region and store the information  $D(PE) + C(PE)$  of this wavefront in a HEAP. Using this data structure, the run time for Steps 3–5 is reduced from  $O(m_1^2 + m_2^2 + \dots + m_k^2)$  to  $O(S_1 \log S_1 + S_2 \log S_2 + \dots + S_k \log S_k) = O(V \log V)$ . Thus, the total time complexity of the region selection algorithm becomes  $O(V \log V)$ .

### VI. SOLUTION TO THE NODE ALLOCATION PROBLEM

After the near convex region is selected, we continue allocating nodes of the incoming application to the PEs with specific voltage levels in the selected region (see Fig. 6) while minimizing the interprocessor communication. To keep track of the node allocation process, we color each node white, gray, or black. A gray node indicates that it has some tentative PE locations but its precise location will be decided later. On the contrary, a black node indicates that it has been already mapped onto some PE, and this mapping will not change anymore. All nodes start out being white and may later either become gray and then black or become directly black. A PE is set to be unavailable after a black node is mapped onto it.

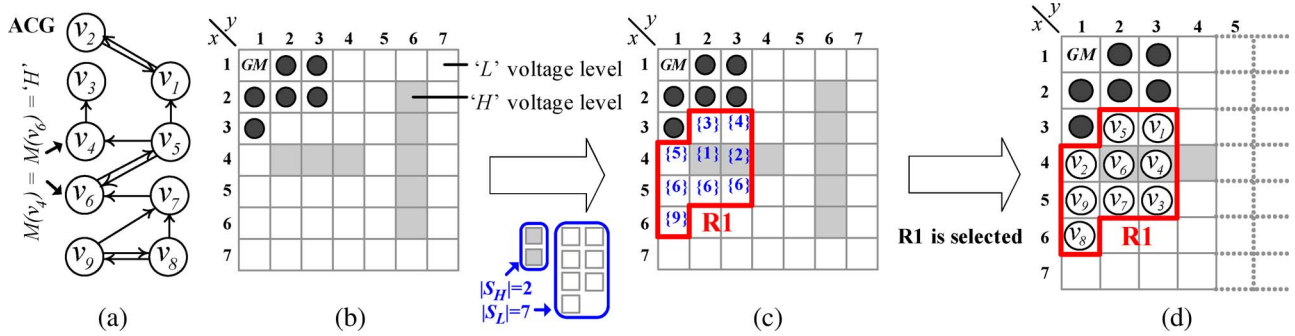


Fig. 14. Incremental run-time mapping process. (a) ACG of the incoming application. (b) System behavior. (c) Near convex region selection process. (d) Node allocation process.

Step 1): Color all nodes white. Then, start with the first white node  $t$  in the smallest node set based on the ordered set.  
 Step 2): **IF** neighbors of node  $t$  are neither gray nor black, then do **DISCOVER** for node  $t$ .  
 Step 3): **IF** neighbors of node  $t$  are either gray or black, then do **FINISH** for node  $t$ .  
 Step 4): Go back to the first node of the ordered set, do Steps 2 and 3 for each nonblack node  $t$  until the color of any nonblack node changes. Then go to Step 5.  
 Step 5): Repeat Step 4 if there exists any nonblack node in the ordered set; otherwise, stop the algorithm.

Fig. 15. Node allocation algorithm.

We define two actions for nodes.

- 1) **DISCOVER**. This consists of the following: 1) Select available PEs with a specific voltage level for node  $t$  and 2) color node  $t$  gray; then, node  $t$  is considered as “discovered.”
- 2) **FINISH**. This consists of the following: 1) Select a specific PE for node  $t$  such that the distance between node  $t$  and its gray or black neighboring nodes is minimized (note that, if more than one PE gets the minimum distance, we select the  $PE_{ij}$  with its  $D(PE_{ij})$  closest to the number of nonblack neighbors of node  $t$ ) and 2) color node  $t$  black; then, node  $t$  is considered as “finished.”

In short, we first sort nodes into an ordered set using the nonincreasing order of their total communication volume, i.e., the higher communication volume a node has, the earlier it is discovered or finished. The node allocation algorithm is summarized in Fig. 15.

Let us follow now the same example in Fig. 14. The ACG in Fig. 14(a) is going to be mapped onto the region  $R_1$ , which has been selected in Section V-C. The final result is shown in Fig. 14(d); Fig. 16 shows the node allocation process step by step. Remember that, for this ACG, the smallest node set is  $S_H = \{v_4, v_6\}$ .

Assume now that, based on the total communication volume, the node ordered set is  $\{9, 6, 7, 5, 8, 4, 1, 3, 2\}$ . Moreover, assume that all nodes are initially white [Fig. 16(a)]. We start with node 6, since it has the smallest order among the nodes in  $S_H$  (Step 1). Since, at this time, its neighbors’ nodes 5 and 7 [see in Fig. 16(a)] are white, we **DISCOVER** this node (i.e., color it gray) and select  $PE_{42}$  and  $PE_{43}$  (because these are the only PE locations at “H” voltage level in region  $R_1$ ) to map it

(Step 2), namely, node 6 will be allocated onto  $PE_{42}$  or  $PE_{43}$  later [Fig. 16(b)]. Then, continuing with Step 4, we go back to the first node in the ordered set. At this moment, the color of node 9 changes from white to gray since all neighbors of node 9 are white; we select  $PE_{32}$ ,  $PE_{33}$ ,  $PE_{41}$ ,  $PE_{51}$ ,  $PE_{52}$ ,  $PE_{53}$ , and  $PE_{61}$  for it [Fig. 16(c)]. With the following repeat of Step 4, the color of nodes 9 and 6 remains unchanged. Then, we consider node 7; its color changes from white to black directly since nodes 6 and 9 are gray. We allocate node 7 onto  $PE_{52}$  [Fig. 16(d)] and then repeat Step 4. Node 9 becomes black since its neighbor node 7 is colored black [Fig. 16(e)], and we allocate node 9 onto the precise location  $PE_{51}$  [Fig. 16(f)]. We continue this process until all nodes are colored black and each node is allocated a precise PE location. Fig. 14(d) shows the final result of the node allocation process.

*Complexity of the Node Allocation Algorithm.* The total run time of our algorithm has a complexity of  $O(V^2 + E)$ . This is because the body of the loop (Steps 4–5) executes  $|V|$  times while reaching at most  $|V|$  vertices each time. In addition, since each node will be reached at most two times (i.e., DISCOVER and FINISH), and the adjacency list of each node is scanned only when the node is reached, the total time of scanning the adjacency lists is  $O(E)$ .

## VII. EXPERIMENTAL RESULTS

We first evaluate the impact of the near convex region selection and node allocation steps of the incremental mapping process using synthetic benchmarks (see Sections VII-A and VII-B, respectively). To show the potential of our proposed mapping algorithms for real applications, we later apply it to the embedded system synthesis benchmark suite E3S [10] (Section VII-C).

### A. Evaluation of Region Selection Algorithm on Random Applications

To show that the choice of a near convex region heavily impacts the communication cost of the incremental mapping process, we consider the following experiments. Several sets of applications are generated using the TGFF package [9]. The node number and the communication volume are randomly generated according to some specified distributions. Then, applications are randomly selected for mapping onto the system

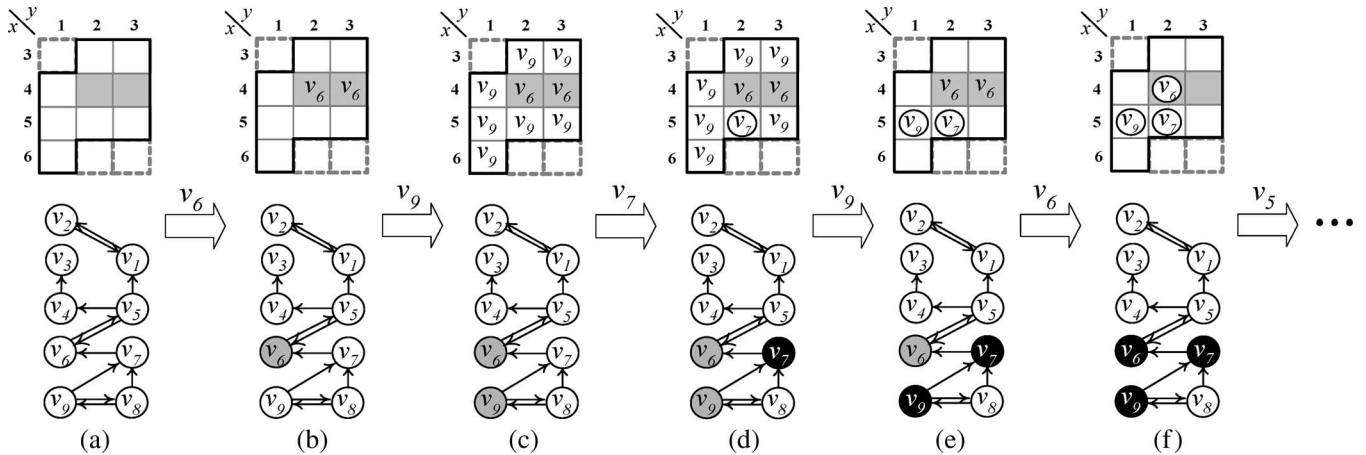


Fig. 16. Node allocation process based on the example in Fig. 14. (a) Initial configuration with every node white. (b) Node 6 is discovered. (c) Node 9 is discovered. (d) Node 7 is finished and colored black. (e) Node 9 is colored from gray to black. (f) Node 6 is colored from gray to black.

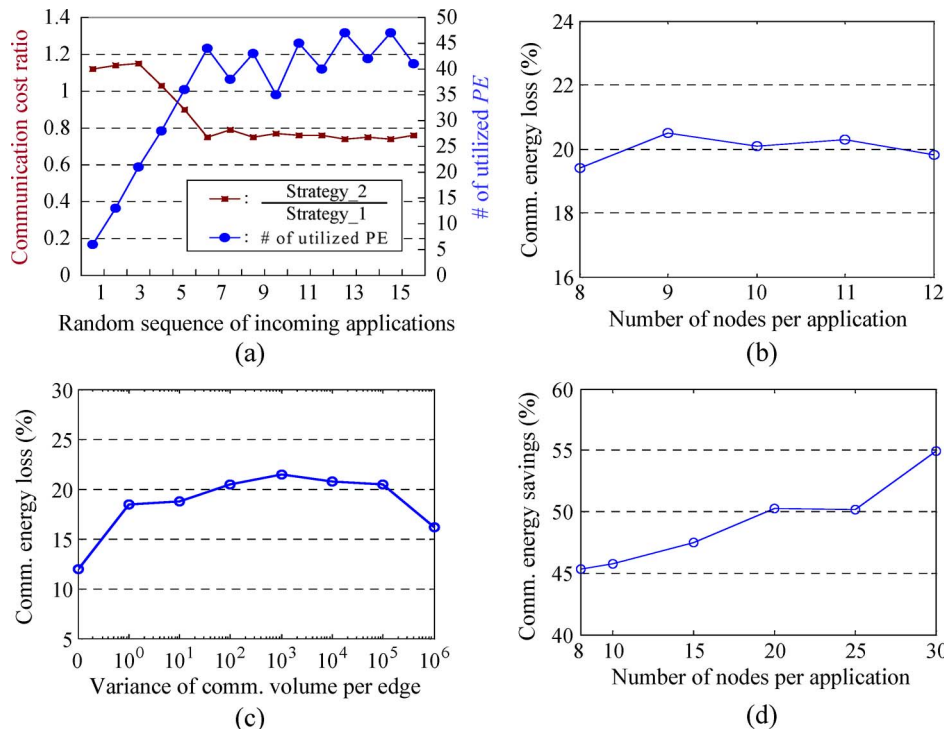


Fig. 17. (a) Impact of selection region process on interprocessor communication. (b) Communication energy loss: Optimal mapping versus our allocation algorithm given a selected region. (c) Optimal versus our allocation algorithm under different communication volumes. (d) Communication energy savings: Arbitrary mapping versus our allocation algorithm.

or being removed from it. For mapping onto the resulting system while preexisting application remains fixed, two different strategies are implemented: 1) a greedy approach minimizing the interprocessor communication cost of the current configuration but *without* considering the newly incoming applications and 2) a near convex region is first selected using the proposed approach, and then, the application is optimally mapped onto this region using exhaustive search.

The number of nodes per application ranges between five and ten. The system consists of  $7 \times 7$  PEs with PE<sub>11</sub> being used as the GM. The variance of the communication volume per edge in one application is set arbitrarily between 0 and 10<sup>6</sup>. Initially, there is no application in the system. The sequence of events

in the system is incremented whenever an application comes to or departs from the system. If the number of idle PEs in the system is smaller than the number of nodes of the incoming application, then the incoming application is not accepted.

Fig. 17(a) shows the interprocessor communication cost ratio between the mapping using Strategy<sub>1</sub> (i.e., without selecting a region) and that in Strategy<sub>2</sub> (*with* selecting a near convex region). Here, the interprocessor communication contains all communications (i.e., preexisting and the incoming applications) in the system. We also show the number of utilized PEs (except the GM) in that particular system configuration.

As shown in Fig. 17(a), there is a slight increase in the communication ratio in the beginning because the greedy approach

performs well when the number of utilized PEs in the system is small. Once the number of utilized PEs increases due to the incoming applications, the benefit of our proposed algorithm becomes obvious. Finally, the ratio becomes stable since, for Strategy\_1, when the application leaves the system, there is always a scattered region left for additional mapping. This example demonstrates that near convex region selection definitely helps the incremental mapping process.

*B. Evaluation of Node Allocation Algorithm on Random Applications*

We first compare the run time and solution of our algorithm against exhaustive approach. Experiments in this paper are performed on an Intel Pentium 4 CPU (2.60 GHz with 768-MB memory). The run time for finding the optimal mapping within selected region increases exponentially with the number of nodes in each application: For 8, 9, 10, 11, and 12 nodes in one application, it takes 0.2 s, 1.5 s, 4 min, 10 min, and 2 h, respectively, to obtain the optimal mapping. On the other hand, the run time of our algorithm stays within 3  $\mu$ s, when the number of nodes varies between 8 and 20. Since finding the optimal mapping for a region with 13 nodes takes more than 26 h, we vary the number of nodes per application from 8 to 12 [see points on *x*-axis in Fig. 17(b)]. More specifically, there are five categories ( $|V| = 812$ ), each category containing 40 applications generated with TGFF.

We denote the energy consumption of our allocation algorithm by  $E_h$  and the energy consumption of the optimal mapping with the same region by  $E_e$ . Thus,  $(E_h - E_e)/E_e \times 100\%$  is the percentage of reported energy loss compared to the optimal solution. As shown in Fig. 17(b), the energy loss for each category is always less than 21%. Therefore, our node allocation algorithm provides good results for large designs.

Now, we address the impact of variance of communication volume per edge on the energy consumption in Fig. 17(c). The node number in one application used in this experiment is fixed to ten, and the variance of communication volume per edge is set from  $10^0, 10^1, 10^2, \dots, 10^6$  (seven categories). For each category, we run 50 different ACGs and calculate the averages. It can be seen from Fig. 17(c) that our average communication energy loss in all categories is within 21.5% compared to the optimal solution under the same region.

Last, in Fig. 17(d), we compare the solution of an arbitrary mapping against our algorithm. Since the run time of arbitrary mapping is very small, we can consider ACGs with a large number of nodes [i.e., 30 in Fig. 17(d)] and see if our algorithm scales well. The number of nodes per application used in this experiment ranges between 8 and 30 (i.e., six categories,  $|V| = 8, 10, 15, 20, 25, \text{ and } 30$ ). We generate 20 different regions with PE locations corresponding to the number of nodes in each category and then run 20 different applications on each selected region. The variance of communication volume per edge and per application is set between 0 and  $10^6$ .

We denote the energy consumption of our allocation algorithm by  $E_h$  and the energy consumption of the arbitrary mapping solution by  $E_a$ ; then  $(E_a - E_h)/E_a \times 100\%$  is the percentage of the reported energy savings compared

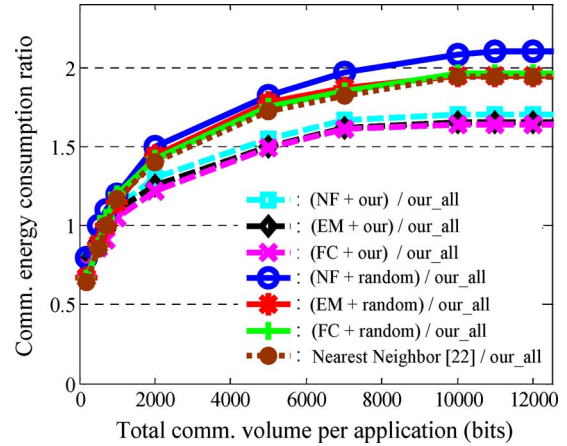


Fig. 18. Communication energy consumption comparison using random applications.

to the arbitrary mapping solutions, which are averaged from 500 random results. As shown in Fig. 17(d), at least 45% savings can be achieved in all categories; of note, the savings increase as the node size scales up.

*C. Random Applications Considering Energy Overhead for the Entire Incremental Mapping Process*

We compare seven scenarios to our near convex region selection and node allocation technique (denoted as “our\_all” in Fig. 18) in terms of the communication energy consumption.<sup>4</sup> For all these scenarios, we first perform the region selection process using NF, EM, FC, or our near convex region selection technique (our). Then, we perform node allocation using either random mapping (random) or our proposed approach (our) presented in Section VI.

For example, in Fig. 18, the notation “NF + our” indicates that we implement the “NF” algorithm (in Section V-A) for region selection step and then use “our” node allocation solutions (in Section VI) for mapping nodes into the selected region. The experimental results in Fig. 18 already include the energy overhead of running these algorithms, i.e., “NF,” “EM,” or “FC,” and “our.”<sup>5</sup>

Finally, the last scenario we consider is the approach proposed in [22]. In this case, we allocate the nodes as close as possible without considering a particular region. The comparison with this scenario is marked as “Nearest Neighbor” in Fig. 18.

As observed in Fig. 18, when the total communication volume per application is over 10 000 b, we can achieve more than 37.5% communication energy savings compared to all other scenarios.

<sup>4</sup>In this comparison, we consider the energy overhead of running our online processes. This overhead is measured by executing the C programs on MicroBlaze processor running on Xilinx Virtex-II Pro XC2VP30 field-programmable gate array (FPGA). The energy overhead of delivering control messages (see Section III-B) is also included here.

<sup>5</sup>Note that the results in Fig. 18 assume the “random” algorithm has zero energy overhead.

#### D. Real Applications Considering Energy Overhead for the Entire Incremental Mapping Process

The communication energy overhead of online processes contains the message transmission into the control network and our online algorithms (i.e., near convex region selection and node allocation). The incremental mapping process is activated only when a new application arrives, and therefore, the PEs need to send their status to GM. The communication volume for all control messages is  $[a \text{ bits (for showing PE address, which depends on network size)} + 1 \text{ bit (PE status)}] \times \text{MD}$  (MD of all PEs to GM). For the  $6 \times 6$  network,  $a = 6(2^6 > 36)$  and  $\text{MD} = 180$ ; therefore, all control bits for one incoming application are within 1 kb. Compared to the communication volume in real applications (which is in the megabit range), the energy overhead for transmitting the control messages is negligible.

Next, we evaluate the extra energy overhead of our online algorithms. Our system contains  $6 \times 6$  PEs of AMD ElanSC520 (133 MHz), AMD K6-2E (500 MHz), and one MicroBlaze core (100 MHz) for the GM running our online algorithms. To evaluate the potential of our online algorithms for real applications, we apply it to embedded system synthesis benchmark suit E3S [10]. We first do the offline partitioning process for each benchmark. The communication energy consumption is measured by a C++ simulator using the bit energy model [8]. We start with a given system configuration running a set of preexisting applications. We denote some terms for energy saving calculation.

- 1)  $P_h$  is the communication power consumption of our mapping algorithms.
- 2)  $P_a$  is the communication power consumption of the implementation of arbitrary allocation scheme which maps tasks on a contiguous region as far as possible.
- 3)  $P_{\text{online}}$  is the power consumption of running the online algorithms (a constant, obtained from MicroBlaze datasheet).
- 4)  $T_{\text{et}}$  is the execution time of that application.
- 5)  $T_{\text{online}}$  is the execution time of running our online algorithms (obtained from MicroBlaze processor running on Xilinx Virtex-II Pro XC2VP30 FPGA).

Thus, the communication energy savings of our algorithms compared to an arbitrary mapping is calculated as follows:

$$\frac{(P_a \times T_{\text{et}}) - (P_h \times T_{\text{et}} + P_{\text{online}} \times T_{\text{online}})}{P_a \times T_{\text{et}}} \times 100\%.$$

As shown in Table III, if running the *telecom* benchmark for only 0.007 ms, we cannot achieve any communication energy savings; the energy overhead of running our algorithms plus the energy with our algorithms applied is almost the same as the communication energy with arbitrary allocation scheme. However, if running the *telecom* benchmark longer (0.03 ms for example), we already gain 25% communication energy savings compared to the arbitrary mapping solution; we note that the overhead of running our algorithms is included.

We observe that about 50% communication energy savings can be achieved, on average, compared to an arbitrary implementation when the execution time of applications is over

TABLE III  
ARBITRARY MAPPING VERSUS OUR ALGORITHM RESULTS

Benchmark	$T_{\text{et}}$ , Application exec. time (msec)	Communication Energy savings
<i>telecom</i>	0.007	0%
	0.03	25%
	> 0.2	50.3%
<i>consumer</i>	0.0005	0%
	0.008	25%
	> 0.04	47%

0.2 ms. The run-time overheads of executing incremental mapping process on MicroBlaze (i.e.,  $T_{\text{online}}$ ) for *telecom* and *consumer* are 53 and 42  $\mu\text{s}$ , respectively.

#### VIII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a run-time approach to incrementally map a number of applications onto a homogeneous tile-based NoC with multiple voltage levels. As shown, using the near convex region selection technique, the mapping results of our algorithms can be obtained very efficiently; in addition, they are not far from the optimal case. Moreover, additional incoming applications can be added into the system with minimal communication overhead.

We plan to extend this work in several directions. First, for SoC applications, we intend to consider more heterogeneity (i.e., cores with different functionality), while including energy consumption as well. Another direction is to address the run-time incremental scheduling and processor sharing problems. Last but not least, we plan to consider the resource management for platforms allowing dynamic voltage and frequency scaling, as this can provide further energy savings.

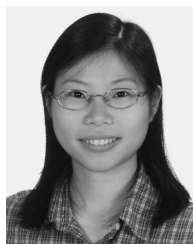
#### ACKNOWLEDGMENT

The authors would like to thank Dr. F. Catthoor of Interuniversity Microelectronics Center for pointing out several issues related to the experimental part.

#### REFERENCES

- [1] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 21–40, Aug. 2007.
- [2] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. ACM/IEEE DAC*, Las Vegas, NV, Jun. 2001, pp. 684–689.
- [3] V. Raghunathan, M. B. Srivastava, and R. K. Gupta, "A survey of techniques for energy efficient on-chip communication," in *Proc. ACM/IEEE DAC*, Anaheim, CA, Jun. 2003, pp. 900–905.
- [4] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.
- [5] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proc. ACM/IEEE DATE*, Paris, France, Feb. 2004, pp. 896–901.
- [6] L. T. Smit *et al.*, "Run-time assignment of tasks to multiple heterogeneous processors," in *Proc. Progress Embedded Syst. Symp.*, Nieuwegein, The Netherlands, Oct. 2004, pp. 185–192.

- [7] S. Murali *et al.*, "Mapping and configuration methods for multi-use-case networks on chips," in *Proc. ACM/IEEE ASP-DAC*, Yokohama, Japan, Jan. 2006, pp. 146–151.
- [8] T. T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Proc. ACM/IEEE DAC*, New Orleans, LA, Jun. 2002, pp. 524–529.
- [9] *Task graphs for free (TGFF v3.0)* Keith Vallerio, 2003. [Online]. Available: <http://ziyang.ece.northwestern.edu/tgff/>
- [10] R. Dick, *Embedded system synthesis benchmarks suites (E3S)*. [Online]. Available: <http://www.ece.northwestern.edu/~dickrp/e3s>
- [11] A. M. Pastnak, P. H. N. de With, S. Stuijk, and J. van Meerbergen, "Parallel implementation of arbitrary-shaped MPEG-4 decoder for multiprocessor systems," in *Proc. Visual Commun. Image Process.*, San Jose, CA, Jan. 2006.
- [12] J.-M. Chang and M. Pedram, "Codex-dp: Co-design of communicating systems using dynamic programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 7, pp. 732–744, Jul. 2000.
- [13] V. Nollet, T. Marescaux, and D. Verkerst, "Operating-system controlled network on chip," in *Proc. ACM/IEEE DAC*, San Diego, CA, Jun. 2004, pp. 256–259.
- [14] P. Pop, P. Eles, T. Pop, and Z. Peng, "An approach to incremental design of distributed embedded systems," in *Proc. ACM/IEEE DAC*, Las Vegas, NV, Jun. 2001, pp. 450–455.
- [15] J. Kao and F. B. Prinz, "Optimal motion planning for deposition in layered manufacturing," in *Proc. Des. Eng. Tech. Conf.*, Atlanta, GA, Sep. 1998, pp. 1–10.
- [16] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage–frequency island partitioning for GALS-based networks-on-chip," in *Proc. ACM/IEEE DAC*, San Diego, CA, Jun. 2007, pp. 110–115.
- [17] M. A. Bender *et al.*, "Communication-aware processor allocation for supercomputers," in *Proc. Workshop Algorithm Data Struct.*, Waterloo, ON, Canada, Aug. 2005.
- [18] S. Bertozzi *et al.*, "Supporting task migration in multi-processor systems-on-chip: A feasibility study," in *Proc. ACM/IEEE DATE*, Munich, Germany, Mar. 2006, pp. 1–6.
- [19] C. Y. Lee, "An algorithm for path connection and its applications," *IRE Trans. Electron Comput.*, vol. EC-10, pp. 346–365, Sep. 1961.
- [20] C. M. Bender, M. A. Bender, E. D. Demaine, and S. P. Fekete, "What is the optimal shape of a city?" *J. Phys. A, Math. Gen.*, vol. 37, no. 1, pp. 147–159, Jan. 2004.
- [21] R. M. Karp, A. C. McKellar, and C. K. Wong, "Near-optimal solutions to a 2-dimensional placement problem," *SIAM J. Comput.*, vol. 4, no. 3, pp. 271–286, Sep. 1975.
- [22] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proc. IEEE/IFIP Workshop Rapid Syst. Prototyping*, Porto Alegre, Brazil, May 2007, pp. 34–40.
- [23] U. Y. Ogras, R. Marculescu, H. G. Lee, and N. Chang, "Communication architecture optimization: Making the shortest path shorter in regular networks-on-chip," in *Proc. IEEE/ACM DATE*, Munich, Germany, Mar. 2006, pp. 712–717.



**Chen-Ling Chou** (S'04) received the B.S. and M.S. degrees in electrical control engineering and electronics engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2002 and 2004, respectively. She is currently working toward the Ph.D. degree in electrical and computer engineering in the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA.

Her research focuses on communication-centric design methodologies for large-scale system on chip, with a special emphasis on modeling, analysis, and run-time optimization for network-on-chip architectures.



**Umit Y. Ogras** (S'00) received the Ph.D. degree in electrical and computer engineering from the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, in 2007.

He is a Postdoctoral Research Associate with the Department of Electrical and Computer Engineering, Carnegie Mellon University. His research interests are in the areas of embedded systems and electronic design automation. In particular, his research focuses on communication-centric design methodologies for nanoscale systems on chip, with a special interest on

networks-on-chip communication architectures.



**Radu Marculescu** (S'94–M'98–SM'07) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1998.

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

Dr. Marculescu is a member of the Association for Computing Machinery. He is a recipient of the National Science Foundation's CAREER Award in 2001 in the area of design automation of electronic systems. He received the 2005 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS Best Paper Award from the IEEE Circuits and Systems Society, 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 and South Pacific Design Automation Conference in 2003. He was also awarded the Carnegie Institute of Technology's Ladd Research Award in 2002.