Computers and Electrical Engineering 68 (2018) 581–602

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Hotspot-aware task-resource co-allocation for heterogeneous many-core networks-on-chip☆

Md Farhadur Reza*, Dan Zhao[1], Hongyi Wu[2], Magdy Bayoumi[3]

The Center for Advanced Computer Studies, University of Louisiana at Lafayette, 301 East Lewis Street, Lafayette, LA 70503, USA

ARTICLE INFO

Keywords:
Task-resource co-allocation
Network-on-Chip (NoC)
Many-Core NoC
Heterogeneous
Computation and communication
Peak load
Energy hotspots
Cost budget
Linear programming (LP)
Heuristic

ABSTRACT

To fully exploit the massive parallelism of many-core on a chip, this work tackles the problem of mapping large-scale applications onto heterogeneous networks-on-chip (NoCs) while minimizing hotspots. A task-resource co-optimization framework is proposed which configures the on-chip communication infrastructure and maps the applications simultaneously and coherently, aiming to minimize the peak energy under the constraints of computation power, communication capacity, and total cost budget of on-chip resources. The problem is first formulated into a linear programming model to search for optimal solution. A heuristic is further developed for fast design space exploration at design-time and run-time in large-scale NoCs. Extensive simulations are carried out under real-world benchmarks and randomly generated task graphs to demonstrate the effectiveness and efficiency of the proposed schemes. Real system simulations show the significant improvement (30–200%) in NoCs latency and throughput compared to the state-of-the-art minimum-path approach because of the diminishing hotspots and balanced load distribution.

1. Introduction

Multicore chip design has been increasingly deployed in embedded computing systems ranging from small mobile devices (e.g., Nvidia Tegra 3) to large telecommunication servers (such as ARMv8-A) in near future to meet ever-increasing computational demands under a stringent energy budget. A single chip with many-cores can replace a traditional server or a rack of servers in a data center. Many-core on-chip systems have better power efficiency, interconnection, and latency compared to traditional local area network (LAN) based systems [1]. Industries and academicians are working to integrate many-cores on a chip that can be comparable with the big data-center solution. For example, a team of researchers at the University of California Davis has implemented (with the help of IBM) a 1000-core chip [2], which has a maximum computation rate of 1.78 trillion instructions per second and contains 621 million transistors. The companies, who have their own data-center (e.g., Google and Facebook), are working with the chip companies (e.g., Intel) to get a many-core chip solution which can replace their big data-center solutions. Qualcomm has unveiled a 48-core 64-bit ARMv8 processor for servers in data centers [3]. Other companies (e.g., AMD, Amazon) have also taken initiatives towards many-core on-chip systems.

☆ Reviews processed and approved for publication by Editor-in-Chief.

* Corresponding author. Current affiliation: Department of Electrical & Computer Engineering, The George Washington University, 800 22nd Street NW, 5000 Science & Engineering Hall, Washington, DC 20052, USA.

E-mail addresses: mxr7945@louisiana.edu, farhadur_reza@gwu.edu (M.F. Reza), dzhao@louisiana.edu, zhao@cs.odu.edu (D. Zhao), wu@louisiana.edu, h1wu@odu.edu (H. Wu), mab0778@louisiana.edu (M. Bayoumi).

[1] Current affiliation: Department of Computer Science, Old Dominion University, 3300 Engineering & Computational Sciences Building, Norfolk, VA 23529, USA.

[2] Current affiliation: Department of Electrical & Computer Engineering, Old Dominion University, 231 Kaufman Hall, Norfolk, VA 23529, USA.

[3] Current affiliation: Department of Electrical & Computer Engineering, University of Louisiana at Lafayette, 131 Rex Street, Lafayette, LA 70504, USA.

https://doi.org/10.1016/j.compeleceng.2018.04.019
Received 12 May 2017; Received in revised form 24 April 2018; Accepted 25 April 2018
Available online 22 May 2018
0045-7906/

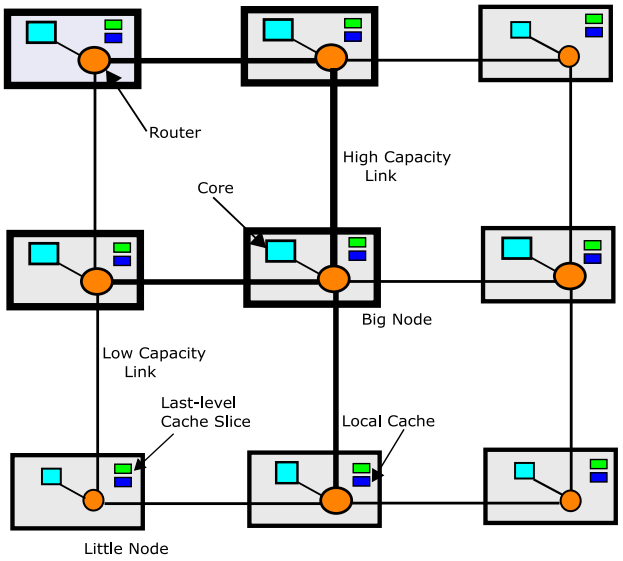

Fig. 1. Heterogeneous many-core architecture on NoC.

On-chip systems are moving towards the integration of tens or hundreds of heterogeneous cores (e.g., central processing unit (CPU), graphics processing unit (GPU), digital signal processing (DSP), accelerator) on a single chip to accomplish emerging applications through low-power highly parallel computing. The heterogeneous many-core architecture offers a feasible approach that embraces high-performance "Big" cores (e.g., CPUs) for computation intensive applications and low-power "Little" cores (e.g., GPUs and DSPs) for majority workloads execution. As the number of cores increases, network-on-chip (NoC) has been adopted as a viable solution to manage complex interconnection networks. NoC offers several important benefits over the traditional bus in terms of scalability, parallelism, throughput, and power efficiency [4]. For example, a many-core architecture on NoC is illustrated in Fig. 1 where heterogeneous processor tiles are placed in a 2D grid. Each tile contains a processor (with distinct computation power), its local cache, a slice of the shared last-level cache, and a router (configured with different communication capacity) for data and control transmission between the tiles via varying bandwidth links.

1.1. Task allocation in many-core NoCs

A many-core server may run multiple applications simultaneously. To fully utilize the computation power (or computation capacity) of the cores, an application can be partitioned into a number of tasks to be simultaneously processed on different processor tiles. Meanwhile, multiple tasks with diverse computation workloads from different applications may share the same tile under the restricted on-chip resource budget, as illustrated in Fig. 2. While task partitioning is beyond the scope of this work, we assume each application has been appropriately partitioned into a set of tasks that can be executed concurrently. Though tasks of the same application are contiguously mapped in Fig. 2, tasks of the same application can be remotely mapped in real scenario, which will be discussed soon. Data are transferred between communication dependent tasks via NoC to fulfill computation of the applications. NoC, as a communication network, also consumes a significant percentage of total chip power [5]. For example, on-chip network consumes 36% of total chip power in 16-tile MIT Raw [5]. The NoC power consumption has been increasing with the increase in core integration on the chip. Heterogeneous traffic loads are common in NoC, e.g., higher uniform traffic loads are exhibited in the center of

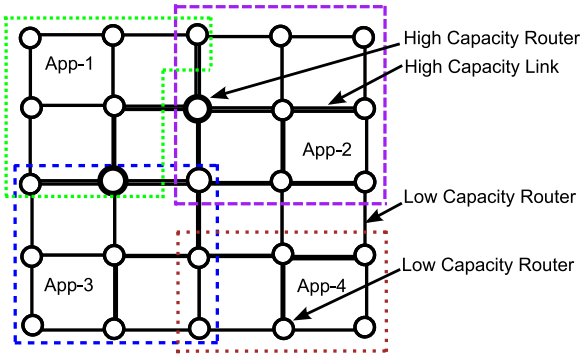

Fig. 2. Application partitioning and mapping in a many-core NoC.

a mesh than in the periphery. Thus different links may carry significantly different data rates and number of traffic flows. Also nodes can have different computation loads. These variable demands arise heterogeneous traffic in different parts of the chip. Power and thermal dissipation also increase proportionally with the traffic loads. Because of that heterogeneity, power and thermal local hotspots can occur, and hotspots can severely hamper the chip performance (e.g., congestion increase, throughput degradation) and reliability (e.g., burn the chip, fault and/or deadlock in the chip). Packets passing through the hotspot area have the higher chance of facing problems, e.g., delay, contention. These problems in a hotspot region can propagate throughout the network as incoming packets will experience problems while trying to use the hotspot resource. That results in blocked resource problem for other incoming traffic as the resources are already occupied by the packets in the hotspot region. To cope with the heterogeneity of traffic workloads, NoC is configured heterogeneously in terms of both link bandwidth and node (core/router) capacity for energy efficiency (reduce power/temperature along with hotspots) while allocating demanded communication and computation capacity under resource sharing.

Given a typical specification describing a vast number of heterogeneous workload requests (i.e., applications and tasks) with diverse computation and communication demands at design-time/run-time, on-chip resource budget (e.g., link bandwidth, cost constraints), and energy-efficient performance objectives, a challenging research problem is to exploit the optimal mapping of resource demands onto various heterogeneous processor tiles while heterogeneously configuring NoC communication capacity to fulfill energy-aware task-resource co-allocation. The mapping optimization in heterogeneous NoC faces new challenges of very limited on-chip resource usage including tight energy and area cost budgets. It becomes even more critical when the core count keeps on increasing with ever shrinking chip size. The heterogeneity in both processing nodes and workloads have important ramifications for the design of on-chip communication architecture underneath as it requires a trade-off exploration of interrelated design components to achieve both computation and communication efficiency without incurring energy hotspots.

The quest for efficient task mapping has led to a verity of research thrusts with different problem settings and optimization criteria, such as minimizing energy consumption, reducing application latency, meeting QoS of realtime jobs, and facilitating dynamic voltage and frequency scaling. Task mapping is fulfilled by static (design-time) or dynamic (run-time) mapping strategies. Static mapping techniques for multiprocessor NoCs are to find dedicated allocation of tasks at design-time along with architectural design and configuration [6–8]. For adaptive systems with dynamic workloads, run-time mapping techniques are required to accurately addressing the workload characteristics [9,10].

1.2. Contribution of this work

In this work, we address, for the first time, the task-resource co-allocation problem in heterogeneous many-core NoCs, aiming to minimize the hotspots (i.e., balanced workloads) subject to computation and communication constraints. The problem will be formally formulated in Section 3. Briefly, we consider a set of applications and each application consists of a number of tasks. The computation demands of the tasks and the communication requirements between relevant tasks are all given. At the same time, a NoC based many-core server system is considered with heterogeneous cores of dedicated computation power. Each link in the many-core NoC can be configured in different bandwidths related to certain costs, but the total link cost of the chip cannot exceed a given cost budget. We have allowed sharing of all the links (irrespective of their position in the topology) depending on the corresponding link capacity. We have also permitted sharing of the cores by the tasks of multiple applications (if the corresponding core capacity allows) to increase task parallelism and improve system performance (e.g., throughput). Most of the previous work did not consider the sharing of the core resources.

To fully utilize the diverse computation resources of many cores, the tasks of applications need to be reasonably assigned to the tiles to ensure that no computation bottleneck will occur to compromise the system performance. With shared resources, it's likely that multiple tasks of different applications may be assigned to the same processing tile, potentially causing computation hotspots. It becomes essential to balance the workloads among many cores when assigning the tasks. The problem of workload balancing however is determined not only by the application tasks running on the processing tiles, but also by the traffic contention in the network. In observation of communication dependency between tasks, it's essential to effectively utilize the communication bandwidth during task mapping. The traffic flows need to be as evenly distributed as possible to efficiently share cost-sensitive communication resources (with proper link and router configuration), and thus to avoid communication hotspots caused by severe traffic contention. The goal is to map the tasks of various applications onto the heterogeneous cores coping with their computation power and route the traffic between tasks so as to balance the total workload (including computation and communication) as much as possible, in order to avoid energy hotspots chip wide. Reduction of the hotspots (peak energy/load) reduces maximum load at a node and/or a link and balances the load throughout the NoC. Lower peak energy (hotspots) minimizes congestion and queuing delay in the NoC, which results in improved latency and throughput. The NoC is heterogeneously configured to optimally allocate the link bandwidth under the total cost budget. Aiming at the mapping of large scale task and resource allocation onto many-core NoCs, we develop a linear programming (LP) model to search for the optimal solution using IBM CPLEX. A fast and practical approximate algorithm is further proposed to deliver feasible solutions. The uniqueness of this work is outlined below:

- *Explicit Hotspot Minimization:* Chip overheating due to workload (distributed computation and communication) asymmetry is a crucial issue in chip design. Although some prior work (e.g., [11]) attempts to avoid energy hotspots due to regional overcrowded communication loads, this is the first work that explicitly tackles the hotspot minimization problem in many-core NoCs, taking into the consideration of both computation and communication workloads.
- *Task-Resource Co-Allocation:* This is the first attempt that task mapping and NoC configuration are co-designed for energy

orchestration of a heterogeneous NoC. The tasks of applications are mapped onto a set of tiles coping with their computation capability, and the links connecting the tiles and associated routers are cost optimally configured to offer sufficient communication capacity. Thus a heterogeneous NoC is established with non-identical routers and varying bandwidth links under the restricted on-chip resource budget. Mathematical optimization model of task-resource co-allocation problem has been designed. A fast heuristic is proposed for task-resource co-allocation solution for large NoC and/or large number of tasks and applications in quick time in both static and dynamic scenarios.

- *Extensive Exploration of Solution Space:* Most prior work presumes a reduced solution space, e.g., by mapping tasks of an application to nearby cores [6,7,9,10,12] or in an empty rectangle [13]. Such approaches, although intuitively reasonable, cannot guarantee an optimal solution. In this work, we search though the entire solution space for mapping optimization and NoC configuration. For example, a trade-off study is explored in a way that the application tasks are mapped to nodes properly apart to even out the workload distribution while reducing the communication latency.

Related work is discussed in Section 2. We formulate the problem and develop a LP model to search for optimal task mapping in Section 3, aiming to minimize the peak load[4] (maximum computation and communication load at a node and its associated links) under computation and communication constraints and at the same time configure heterogeneous NoC link bandwidth subject to a total cost budget. We further propose a heuristic algorithm, *MinHotspot* in Section 4 for fast design space exploration in large-scale many-core NoCs. To make our approach also applicable for dynamic scenario (besides static mapping), we have extended our proposed *MinHotspot* heuristic for dynamic mapping and configuration in Section 5. We conduct extensive simulations under real world benchmarks and randomly generated task graphs in Section 6 to demonstrate the efficiency of the proposed schemes.

2. Related work

Resource allocation and configuration are driven by various energy and performance goals, e.g., minimize energy, reduce latency, improve throughput, and meeting QoS (quality of service). Depending on the application needs, a mapping strategy maps the tasks and allocates the NoC resources. Mapping can be done at design-time when the requirements and demands of all the applications are known. If an application arrives at the system at run-time, dynamic allocation strategy is needed to map the tasks. This paper considers both design-time and run-time resource allocation. Multiple applications mapping is addressed in this paper, where traditional mapping focuses only on single application mapping. Various task mapping algorithms has been developed for NoC-based multi-core chips, some are based on graph theoretic, some use mathematical programming, and others are heuristic algorithms. For example, variations of minimum-path (shortest tree) algorithms [6,7,9,10,12] has been devoted to mapping highly correlated tasks on the task graph to nearby cores, in order to shorten the communication delay, to reduce the routing distance, or to improve locality of data access. In [6], the communication traffic is split onto multiple paths to improve bandwidth efficiency during task mapping. Routing flexibility is exploited in [7] to expand the mapping solution space, and a branch-and-bound algorithm has been proposed to solve the mapping problem wherein the time complexity increases exponentially with network size. An agile task allocation scheme was presented in [10] that maps the tasks of an application as nearest as possible by applying hill-climbing optimization. It was proposed in [12] to map the closely communicating threads to neighboring cores to improve locality of data accesses. In a similar spirit, the work in [13] intends to map the tasks in the same application together. A multi-application mapping method proposed in [13], which allows multiple applications to use the boundary links that are shared by the corresponding applications. Although it ([13]) effectively reduces overall communication volume, resource contention can be introduced in communication and computation intensive regions.

Some work (e.g., [11]) focuses on the overlapped traffic over shared communication links, aiming to minimize the traffic contention in the network. A mapping heuristic was proposed in [11] along with LP approach for minimizing contention in the network. In addition, various user applications are considered in [8] such that the tasks are mapped to either meet the worst case requirement or dynamically switch between different user-application requirements.

This is the extended version of our previous work [14]. In this work, we address for the first time the placement of computation and communication demanded tasks of multiple applications onto heterogeneous nodes under computation and communication capacity constraints and cost budget, where link capacity is cost minimally configured to fulfill communication interactions. Both computation and communication workloads are as evenly distributed as possible to avoid energy hotspots. We have proposed a LP optimization model and a fast heuristic for task-resource co-allocation for minimizing hotspots in NoC. The proposed approaches (LP model and heuristic) try to spread out the tasks throughout the network to distribute loads evenly on the nodes and links of the network. Because of the balanced load distribution, it reduces the possibility of hotspot(s) at a particular node or link or region. The main principle of our approaches is that it will try to avoid overloaded node and link load to reduce congestion as congestion can prevent network from progressing and completing the tasks. In contrast, traditional minimum-path approaches try to map the tasks in close neighborhood irrespective of the loads scenario in the network. As a result, those (minimum-path) approaches increase the possibility of hotspot(s) because of the overloaded nodes or links.

[4] Peak load and peak energy are used interchangeably in this paper.

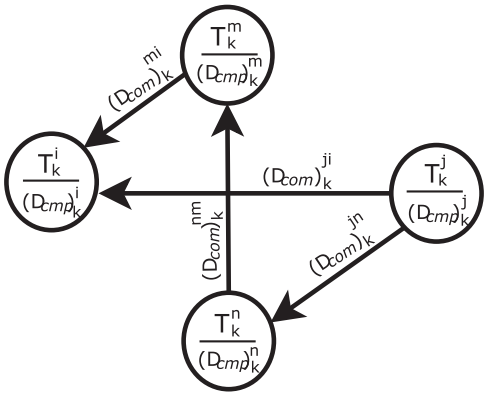

Fig. 3. An example of application task graph $G(V, E)$.

3. Problem formulation and LP solution

In this work, the task-resource co-allocation problem in many-core NoC considers core computation power and link communication capacity constraints and the tightly restricted NoC cost (e.g., area) budget. In order to study the task-resource co-allocation problem in heterogeneous many-core NoC, we develop an optimal solution to allocate the tasks onto the NoC in a way to minimize the peak workload on-chip. Given a many-core chip plane with n processor tiles connected into a $\sqrt{n} \times \sqrt{n}$ 2D mesh, each node i (or tile interchangeably) has its maximum computation power of P_i. Two adjacent nodes i and j are connected by a bidirectional link l_{ij} with a maximum communication capacity of W_{ij}. Here we have assumed a 2D mesh architecture that employs XY-routing for data delivery. However, it is worth mentioning that our proposed mapping algorithm is generally applicable to any network topology and routing scheme, with only minor modification in the calculation of traffic load.

Given a set of K applications $A = \{A_1, A_2, ..., A_k, ...A_K\}$ to be executed on the many-core chip, and each application consists of a set of tasks to be run on different computing nodes in parallel. For example, application A_k is partitioned into M tasks, i.e., $A_k = \{T_k^1, T_k^2, ..., T_k^m, ..., T_k^M\}$, where T_k^m is the m^{th} task for application A_k. The task computation demands and the communication dependency among the tasks are represented by a task graph $G(V, E)$ as shown in Fig. 3, where each vertex v_k^m denotes a task T_k^m with computation demand of $(D_{cmp})_k^m$, and each directed edge e_k^{mn} represents communication from task T_k^m to task T_k^n with a bandwidth demand of $(D_{com})_k^{mn}$.

Let $(X_k^m)^i$ be a binary variable to indicate if a task T_k^m is mapped to node i, i.e.,

$$(X_k^m)^i = \begin{cases} 1, \text{ if task } T_k^m \text{ is mapped to Node } i, \\ 0, \text{ otherwise.} \end{cases} \tag{1}$$

$(X_k^m)^i$ is to be determined in order to achieve the optimization goal.

Next, based on $(X_k^m)^i$, we first outline the communication and computation constraints and then discuss the optimization objection function.

3.1. Communication constraints and link configuration

The problem of workload balancing is determined not only by the application tasks running on the nodes, but also by the traffic contention in the network. In observation of communication dependency between tasks, it is essential to efficiently utilize the link capacity during task mapping. The traffic loads need to be properly distributed to share the cost-constrained communication resources (with proper link and router configuration) so as to avoid communication hotspots.

The adjacent tasks (e.g., T_k^n and T_k^m) of an application k communicate with each other, with a traffic load of $(D_{com})_k^{mn} + (D_{com})_k^{nm}$. When they are mapped to two nodes, e.g., p and q, the traffic is delivered through a path in the NoC, denoted as Ph_{pq} which consists of a set of NoC links. Such a path is determined by the underlying routing algorithm. Note that multiple traffic flows may simultaneously traverse through a communication link l_{ij}, thus the total communication traffic loads via link l_{ij} is:

$$\omega_{ij} = \sum_{\substack{1 \le p,q \le n \\ 1 \le m,n \le M \\ 1 \le k \le K \\ \exists\, l_{ij} \in Ph_{pq}}} (D_{com})_k^{mn} \cdot (X_k^m)^p \cdot (X_k^n)^q. \tag{2}$$

Since each communication link l_{ij} has a maximum bandwidth capacity of W_{ij}, we arrive at the communication constraint:

$$\omega_{ij} \le W_{ij}, \forall\ 1 \le i, j \le n. \tag{3}$$

If the link bandwidth capacity (i.e., W_{ij}) is given as a constant, the communication constraint is straightforward. However, W_{ij} is often non-uniform and configurable. For example, a list of typical link width configuration options $B = \{B_h\}$ and their associated cost

Table 1
Link configuration with varying width and cost.

Link Configuration (h)	1	2	3	...	H
Link Width (B_h, bit)	8	16	24	...	256
Link Cost (C_h, μm^2)	1600	3200	4800	...	51,200

C_h are given in Table 1. The cost is measured by the area of chip occupied by the link. For example, in 22 nm process technology, chip area required for 1-bit width link is 200 μm^2, where wire pitch and length are 80 nm and 2.5 mm, respectively. It is up to the NoC IP designers to determine the best bandwidth for each link. Since the task mapping result may significantly affect the traffic distribution, it is essential to configure W_{ij} according to the traffic needs while meeting the total NoC cost budget.

Given the link configuration options, the bandwidth of link l_{ij} can be configured as,

$$W_{ij} = \left(\sum_{h=1}^{H} Y_{ij}^{h} \cdot B_h \right) \cdot f_{NoC},\ 1 \le i, j \le n, \tag{4}$$

where f_{NoC} is the uniform frequency of the NoC,[5] and Y_{ij}^{h} is a binary variable to indicate the chosen h^{th} link width configuration for l_{ij}, i.e.,

$$Y_{ij}^{h} = \begin{cases} 1, \text{ if the } h^{th} \text{ link configuration is chosen,} \\ 0, \text{ otherwise.} \end{cases} \tag{5}$$

Note that, only one configuration can be chosen for a link, thus we have

$$\sum_{h=1}^{H} Y_{ij}^{h} = 1. \tag{6}$$

Combining Eqs. (2)–(4), we have the communication constraints with adaptive link bandwidth configuration.

3.2. Area budget constraints

Based on the selection of the link configuration Y_{ij}, the cost of link l_{ij} can be calculated as,

$$C_{ij} = \sum_{h=1}^{H} Y_{ij}^{h} \cdot C_h. \tag{7}$$

Obviously, the overall link cost should not exceed the total cost budget of the NoC, denoted as C_{NoC}:

$$\sum_{1 \le i,j \le n} C_{ij} \le C_{NoC}. \tag{8}$$

3.3. Computation constraints

The computation constraint is straightforward. Given that each node i has a maximum computation power of P_i, the total computation power of all tasks assigned to node i must not exceed P_i, i.e.,

$$\rho_i = \sum_{\substack{1 \le k \le K \\ 1 \le m \le M}} (D_{cmp})_k^m \cdot (X_k^m)^i \le P_i,\ \forall\ 1 \le i \le n. \tag{9}$$

3.4. Optimization objective function

Our objective is load-balanced mapping and configuration of NoC to avoid energy hotspots due to excessive computation and overcrowded communication in the NoC. In the meantime, the computation and communication requirements of the applications must be fulfilled to accomplish the jobs. We define the load of node i as the weighted sum of its overall computation and communication workloads, i.e., $\gamma_1 \cdot \rho_i + \gamma_2 \cdot \sum_{\forall j \in l_{ij}} \omega_{ij}$, where ρ_i denotes the total computation load at node i (see Eq. (9)) and ω_{ij} is the total communication traffic load going over link l_{ij} (as shown in Eq. (2)). The parameters γ_1 and γ_2 are the energy coefficient of computation and communication loads at node i, respectively. For instance, under 22 nm process technology, computation and communication energy dissipation are around 45 picojoule/FLOPS and 65 picojoule/bit, respectively [15].

The maximum load among all nodes in the NoC (i.e., the peak load) should be minimized, i.e.,

[5] Note that NoC frequency scaling is out of scope of this paper. We assume f_{NoC} is given in this work.

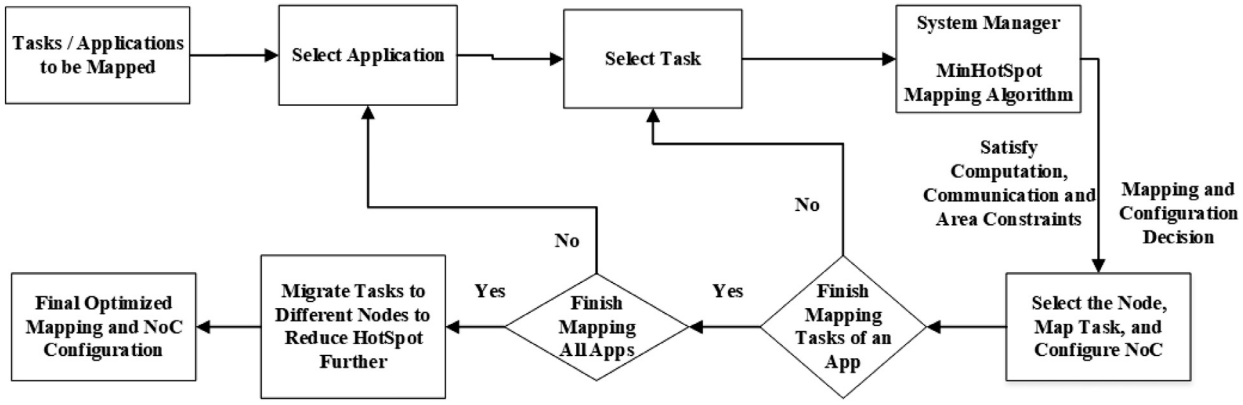

Fig. 4. Proposed *MinHotSpot* mapping and configuration flow.

$$Minimize: max\left(\gamma_1 \cdot \rho_i + \gamma_2 \cdot \sum_{\forall j \in l_{ij}} \omega_{ij}\right), 1 \le i \le n, \tag{10}$$

subject to the constraints given in Eqs. (2)–(9).

The above objective function along with the communication, computation, and cost constraints form a LP model. Its solution suggests a task mapping strategy and a link capacity configuration of the many-core NoC, aiming at minimizing chip-wide energy hotspots while meeting computation and communication requirements under strict cost budget. We have implemented the LP model with a commercial LP solver, i.e., IBM CPLEX [16], which yields optimized task-resource co-allocation. The results will be reported in Section 6.

4. MinHotSpot mapping heuristic

The above LP model achieves optimal mapping of the tasks. However, it is not always possible to get optimal solution within finite time as the integer programming problems (like the above LP model) are NP-hard. Therefore, high computation complexity of the LP model restricts its application in large-scale many-core NoCs. To this end, we further develop a near-optimal heuristic algorithm, namely *MinHotspot*, for fast design space exploration. The summary of step-by-step flows of *MinHotspot* algorithm is shown in Fig. 4. The algorithm is greedy in nature. The greedy strategy is applied in the selection of both applications/tasks and the processor nodes, as outlined below.

4.1. Selection of applications and tasks

First, the applications are sorted in a descending order according to their total computation and communication demands, i.e., $TD(A_k) = \gamma_1 \cdot \sum_{m=1}^{M} (D_{cmp})_k^m + \gamma_2 \cdot \sum_{\forall e_k^{mn} \in G} (D_{com})_k^{mn}$, where G is the task graph introduced earlier. It is desirable to first allocate applications that demand more resources, because it is often more difficult to map such applications in later stage when the remaining resources become fragmentary. Thus the application with the highest total demands $max(TD(A_k))$ is mapped first, and the remaining applications are mapped in the sorted order.

Similarly, for a given application, its tasks are sorted according to their total load, which comprises the computation demand of the task and the communication demands with its neighboring tasks. But note that, when we choose the next task to be mapped, we are only concerned about the communication load between a candidate task and the tasks that have already been mapped. Thus the load of a task is calculated as follows: $TD(T_k^m) = \gamma_1 \cdot (D_{cmp})_k^m + \gamma_2 \cdot \sum_{\forall e_k^{mn} \in G, n \in G'} (D_{com})_k^{mn}$, where G' is the partial task graph that has been mapped so far. An exception is given to the first task of each application because G' is empty. In this case, we replace G' with G. The task with the highest $TD(T_k^m)$ is chosen as the next task to be mapped. If there is a tie, the task with the highest connectivity with the prior mapped tasks will be chosen.

4.2. Node selection and link configuration

Once a task is selected, we examine all hypotheses of placing the task onto every possible node. Computation and communication loads are updated for every possible node option. Shortest-path deterministic routing (e.g., XY-routing in 2D mesh NoC) is applied to route the traffic between source and destination pairs, and the communication load of each link along these paths is updated. Under each hypothesis, we calculate its peak load, as introduced in the LP model. The calculation is similar to Eqs. (2), (9) and (10), but based on the already mapped tasks and the candidate task only. The node with the lowest peak load is chosen to map the task. Note that, we consider a hypothesis only if the computation and communication constraints are satisfied. Such constraints are checked according to Eqs. (3) and (9).

Link configuration is also incorporated here. As discussed earlier, each link in the NoC can be configured with different bandwidth (see Table 1). When a hypothesis is considered, we allocate the smallest bandwidths for the involved links that satisfy the following two conditions. First, the bandwidth should be sufficient to meet the communication constraint (i.e., $\omega_{ij} \leq W_{ij}$), and second, the overall link cost should not exceed the total cost budget of the NoC, i.e., $\Sigma_{1 \leq i,\ j \leq n} C_{ij} \leq C_{NoC}$.

4.3. Iterative assignment procedure

The task-to-node assignment follows a greedy iterative procedure. Briefly, we pick the first unmapped task from the sorted list, and assign it to the node that results in the lowest peak load. If there is a tie in lowest peak load, the total load of the NoC and the capacity utilization are used as tie-breaker. The process repeats until all tasks are mapped, or is aborted if the constraints cannot be satisfied. This greedy procedure aims to achieve the lowest peak load in each iteration. Our research reveals that it can effectively guide the mapping process in general but has a drawback in dealing with the first task in each application. First, if the first task is mapped to a node with insufficient resources around it, the rest tasks have to be mapped to remote nodes, increasing the overall communication load, and accordingly the peak load. The problem is due to the fact that the first task is essentially the anchor point of the application. Hence, during first-task mapping, we should give the higher priority to a node that has higher number of available neighbors (that have enough capacity to accommodate other tasks in the same application). Second, if the first task is mapped to a heavily loaded node, then the rest tasks can further increase the load (due to communication with the heavily loaded node), and eventually can become a bottleneck in the network. Because of that, we should give the higher priority to a node that has lower load.

To this end, we introduce a parameter, named node selection factor (f_i), which is the weighted sum of the number of available neighbors (N_i) and the reciprocal of existing computation load (ρ_i) at node i, i.e., $f_i = \delta_1 \cdot N_i + \delta_2 \cdot \frac{1}{\rho_i}$, where δ_1 and δ_2 are the constant weights. The former indicates the likelihood that, after the first task is placed at the node, other tasks of the same application can be accommodated in the nearby region. Only the neighboring nodes that have enough computation and link capacity to meet the computation and communication requirements of the next task will be included in the neighbor count. The latter contributes to minimizing the peak load while providing computation capacity for the incoming (first) task. It is desirable to find a light-loaded node with sufficient resources in its neighborhood to accommodate the first task of a highly demanding application. Thus the node with the highest selection factor, i.e., $max\{f_i | 1 \leq i \leq n\}$ is chosen to map the first task. Note that for the very first node wherein no any application task is mapped yet, neighbor count is the only determinant factor. The settings of δ parameters are sampled by running preliminary experiments (profiling benchmark) under a variety of network sizes and node capacities. The combination which potentially leads to the minimum peak load is chosen as the δ values. Fig. 5 illustrates the δ values distribution in the mapping of two applications to 4 × 4 and 5 × 5 NoCs at node capacity of 15 and 20, respectively. The best setting is obtained at $\delta_1 = 0.5$ and $\delta_2 = 2$ or $Capacity_{core}$.

As shown in Fig. 6(a), assume the tasks of application 1 (i.e., T_1^0 to T_1^4) have been placed on some nodes, our approach can identify an appropriate anchor point for application 2, leading to lower peak load/energy, for example, in joule. In contrast, Fig. 6(c) and (d) show the scenarios if we only consider neighbor count and node load, respectively. Because of the inappropriate selection of anchor node for first task mapping, both peak load and total load in the network increase. Note that the nodes and links highlighted in red are hotspots carrying high load.

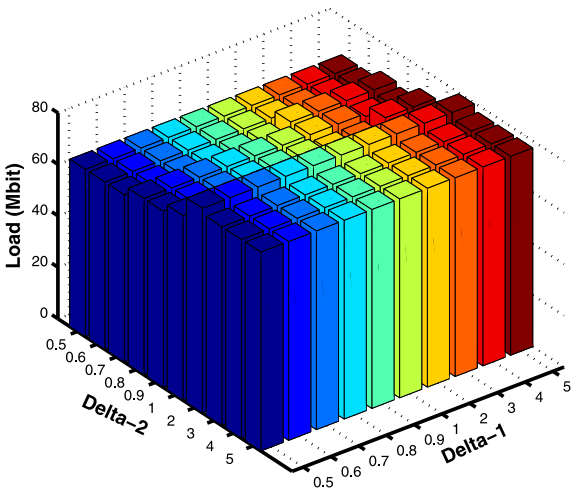

Fig. 5. δ Parameter sampling during application mapping.

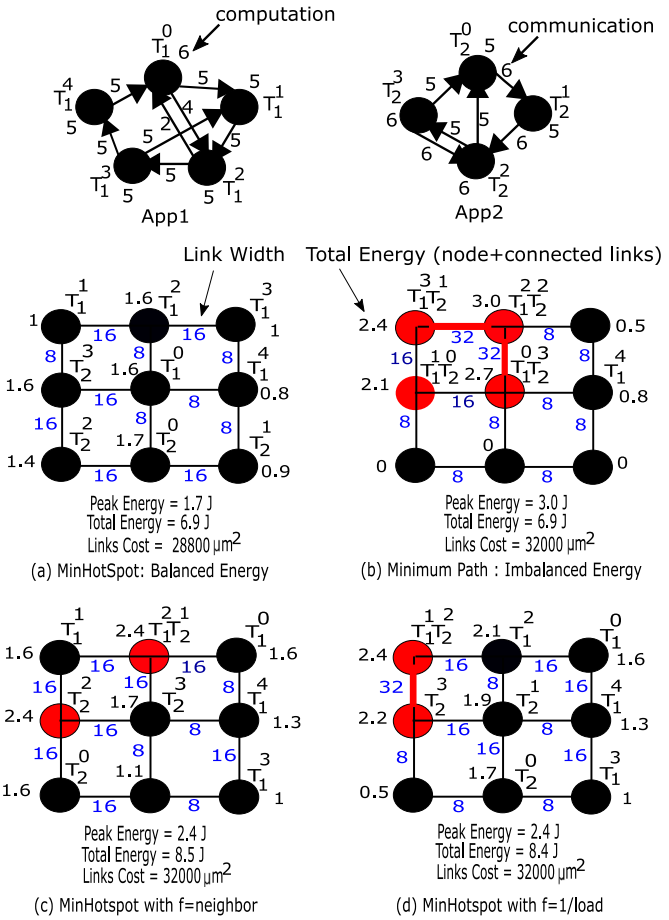

Fig. 6. Comparison of *MinHotSpot* with other approaches, where the capacity per core is 30 GFLOPS and the link cost budget is 33,600 μm^2. Note that we have set the capacity as a constant for simplicity. However, cores may have different capacity. The nodes and links in red are hotspots carrying high load. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The pseudocode of task-to-node assignment phase (above three steps) is shown in Algorithm 1.

4.4. Further refinement

To further reduce the peak load (improve the load-balancing in NoC), we refine the mapping result by exploring options to relocate the tasks like graph partitioning heuristics [17,18]. More specifically, we randomly choose a task currently assigned to a node, and evaluate its peak load would it be moved to every other node. The task is reassigned to the node that results in the lowest peak load (if it is different from the current node). In order to facilitate the move, the computation loads at the two nodes where the task is moved from one to another will be updated and their computation power constraints should be met. The communication traffic of the moved task with their communicating tasks are rerouted in the NoC while their old routing paths will be removed. The communication loads on all the links of these rerouted paths and old paths will be updated and their associated link capacity will be reconfigured according to the traffic needs (such that the link capacity on the removed paths may be decremented while granting higher capacity to rerouted paths). Certainly, the overall cost budget as well as computation and communication constraints should be satisfied at all times. If neither of these computation and communication load constraints and cost budget is satisfied, the move of this task between a pair of nodes won't be considered. We repeat the process until the peak load (hotspot) cannot be further reduced. The pseudocode of task movement heuristic is given in Algorithm 2.

We choose task moving approach [17] over pair-wise exchange method [18] because of several reasons as follows. First, task moving approach [17] has linear time complexity compared to quadratic time complexity of pair-wise exchange method [18]. Second, task moving approach explores all the node options for a task to be moved, whereas pair-wise task exchange only exchanges task between nodes that have traffic. If a node doesn't have any traffic, pair-wise task exchange won't consider that node for swapping but task moving will consider all the nodes for a task movement. Because of that, task moving can yield better performance than pair-wise task exchange. Third, one task swap involves movement of two tasks, while one task moving only moves a task. Therefore, task moving approach has less overhead than pair-wise task exchange as task movement (migration) incurs overhead.

input : Application Gs, NoC Topology Graph
output: Task Placement on NoC
sort $A = \{A_k | k \in K\}$ in $L_A \leftarrow TD(A_k)$;
for *all applications in* L_A **do**
  find first task $FT \leftarrow max(TD(T_k^{F_t}) | F_t \in M)$;
  select first node $FN \leftarrow max(f_i | i \in N)$
    that satisfy computation capacity constraint;
  map(FT)=FN;
  update task list by excluding FT;
  update topology with FN node load;
  for *all unmapped tasks* **do**
    find next task $NT \leftarrow max(TD(T_k^{N_t}) | N_t \in M)$;
    generate candidate node list $L_C \leftarrow n_i | i \in N$ with
      link bandwidth configuration along routing paths that
      satisfy capacity and cost budget constraints;
    select next node $NN \leftarrow min(max(Load_i | i \in N))$ in L_C;
    map(NT)=NN;
    update task list by removing NT;
    update topology with NN node load and links weight;
    configure NoC links width;
  end
end
return map;

Algorithm 1. Task-to-node assignment heuristic.

input : Initial Task Mapping, Weighted Topology Graph
output: Final Task Placement on NoC
for $(i = T_1^1 : T_K^M)$ **do**
  for $(j = 1 : N)$ **do**
    temp-placement=move(N_i, N_j);
    if *temp-placement reduces peak load && satisfies capacity constraints and cost budget* **then**
      update map;
      update topology with node loads and link weights;
    end
    else
      discard move;
    end
  end
end
return map;

Algorithm 2. Task movement heuristic.

The overall complexity of the proposed *MinHotspot* algorithm is analyzed as follows. During the iterative assignment procedure, we must examine *n* hypotheses for each task in each application, resulting in a complexity of *O*(*KMn*). The refinement is also an iterative process. Since the algorithm must monotonically reduce the peak load, the number of iterations is bounded by the maximum possible peak load (which can be considered as a constant for a given set of applications). Thus the complexity of refinement is also *O*(*KMn*). Therefore, the overall complexity is *O*(*KMn*), where K, M, and n are number of applications to be mapped, average number of tasks per application, and number of cores, respectively.

5. MinHotSpot dynamic mapping and configuration

Our proposed *MinHotSpot* static heuristic has a time complexity of *O*(*KMn*), where K, M, n are application count, average no. of tasks per application, and no. of cores in the network. Because of the low time complexity of *MinHotSpot*, this heuristic has been extended to support run-time mapping of incoming tasks and applications. In the worst case, all the tasks and applications will arrive in the system for mapping. In that case, time complexity of dynamic mapping will be same as static mapping. On average, run-time arrival of number of tasks is lower than the total number of tasks to be mapped. Only few changes are needed to make the *MinHotSpot* heuristic applicable for run-time task-resource co-allocation, as outlined below.

- *Selection of Applications and Tasks:*Task is selected depending on the run-time arrival of a number of tasks or applications in the system queue. If only one task arrives in the queue, then that task is selected as this is the single task. If multiple tasks or multiple applications arrive in the system, then task and application selection follows the same procedure as listed in Section 4.1.
- *Node Selection and Link Configuration:*Node selection and link configuration step follows the same procedure as mentioned in the static version of *MinHotSpot* heuristic in Section 4. After the node selection, depending on the communication demands of the arrived tasks, required link-widths are assigned to the corresponding NoC links. This different communication capacity at different links reduces energy and hotspots further compared to homogeneous capacity assignment, which assigns maximum communication capacity at all the links. Link configurations are modified by adding or removing (configuring) links at run-time.
- *Iterative Assignment Procedure:*Iterative assignment procedure follows the same procedure as mentioned in the static *MinHotSpot* heuristic depending on the computation and communication demands of the tasks at run-time.
- *Further Refinement:*Further refinement phase remains same as mentioned in the *MinHotSpot* heuristic. However, we introduce a penalty for task migration as the refinement happens at run-time. Task migration penalty is measured in communication energy consumption penalty for routing a task from a source node to a target node via the control network in NoC. During run-time, we explore the options of task refinement (migration) after a certain interval, as frequent task migration can introduce fluctuation in the system. Also during task migration, all the communication to that tasks needs to be paused by the system manager. That's why, task migration needs to be performed after a certain interval or during a change in application(s) traffic.

6. Simulation & comparison

We implement the LP optimization model as described in Section 3 using a commercial LP solver IBM CPLEX [16] which searches for the optimal solution for the task-resource co-allocation problem. The MATLAB toolbox YALMIP [19] is used to support linear and quadratic programming by interfacing to the IBM CPLEX solver. The proposed *MinHotSpot* mapping heuristic algorithm is implemented to deliver practical solutions for extremely large-scale problems. Both static and dynamic versions of *MinHotSpot* algorithm have been implemented. *MinHotSpot* and *MinHotSpot-Static* are used to indicate the static *MinHotSpot* solution, while *MinHotSpot-Dynamic* is used to indicate dynamic solution of *MinHotSpot*. Mapping dynamism is achieved by mapping incoming tasks and applications at separate times, where the demands of all the tasks of the application (s) are not known beforehand.

We have also implemented a state-of-the-art minimum-path approach as the baseline to compare with our proposed solutions. Minimum-path mapping algorithm maps the communicating tasks of an application as close as possible (using shortest-path algorithm) to reduce the total communication delay and load in the NoC. Minimum-path algorithm also uses pair-wise task exchange (swap) for task migration, where *MinHotSpot* uses task movement among nodes as shown in Algorithm 2. For fair comparison with *MinHotSpot* algorithm, we have modified minimum-path mapping algorithm in [6] by integrating task computation, node capacity heterogeneity, and task computation-node allocation. Both the *MinHotSpot* and the minimum-path approaches are implemented using in-house developed mapping and configuration simulator in C++.

Real system experiments are further carried out using gem5 [20] and GARNET [21] platforms to evaluate the NoC performance in terms of end-to-end latency and network throughput. DSENT [22] tool is used for power consumption evaluation of proposed solutions.

The experiments are carried out based on three platforms: embedded system synthesis benchmarks suite (E3S) [23], COSMIC benchmark suite [24], and randomly generated problem settings using task graph generator [25]. We run the algorithms on a host machine with 6-core Intel Xeon E5-2667 2.9GHz processors, 15MB LLC, and 16GB DDR3.

6.1. Performance comparison under E3S Benchmarks

We first use E3S benchmarks to evaluate the performance of the proposed CPLEX LP optimization model and the near-optimal *MinHotSpot* algorithm. We also evaluate the performance of dynamic solution of *MinHotSpot*. E3S comprises applications in Consumer, Auto Industry, Office-Automation, Networking, and Telecom, where AMD ElanSC520 processors (133 MHz) are used for

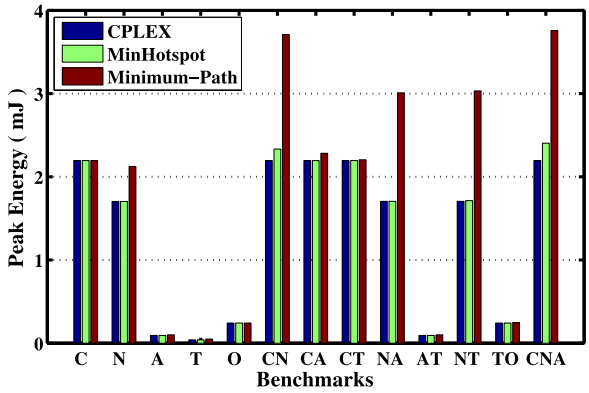

Fig. 7. Peak load comparison in a 3 × 3 NoC under E3S Benchmarks.

data processing. We attempt to simulate complex multi-application domain systems by mixing multiple applications in E3S, e.g., CN for Consumer and Networking applications. The peak load and cost of the NoC are evaluated under various scenarios by varying the number of applications and combinations of computation and communication demands and NoC sizes. For optimal solution comparison, we map the applications in small networks (3 × 3 NoC) as CPLEX cannot provide a solution within a feasible time for a larger NoC size. For other solutions comparison, the applications are mapped in a 4 × 4 to 10 × 10 2D-mesh NoC. The parameters and configurations are kept same under all approaches for a fair comparison.

As shown in Fig. 7 for 3 × 3 NoC, *MinHotSpot* delivers nearly the same peak load/energy (in milli-joule (mJ)) performance as CPLEX under all five E3S benchmarks: Consumer (C), Networking (N), Auto-Industry (A), Telecom (T), and Office-Automation (O). Except office-automation, all these benchmarks have multiple applications. Furthermore, multi-application systems are configured under eight different combinations of applications to evaluate the robustness of the proposed schemes. As can be seen, *MinHotSpot* again produces the same peak load performance as CPLEX except two cases, i.e., CN and CNA, where the peak loads of *MinHotSpot* are slightly higher than that of CPLEX by 6% and 9%, respectively. Our proposed approaches achieve significantly better performance than the traditional minimum-path scheme [6] by 10–80%. This is also evidenced by the load distribution shown in Fig. 8, where the load/energy is extremely unevenly distributed among the nine nodes of the NoC under the minimum-path scheme, as it only considers communication distance during mapping. While it is essentially impossible to achieve perfectly uniform distribution, the CPLEX and *MinHotSpot* result in more evenly distributed load, and accordingly less hotspots in the network.

We also notice the long computation time for CPLEX to discover the optimal solution compared to *MinHotSpot* heuristic in Table 2. For example, as shown in Table 2, for an 8-application mapping (such as NA) on 3 × 3 NoC, it takes about 5 hours for CPLEX to search for the solution while *MinHotSpot* can finish it in only one-seventh of a second. When the number of applications (to be mapped) and/or NoC size increase, CPLEX cannot deliver a solution in a reasonable time period, for example, for 4 × 4 NoC. We have also presented execution times of *MinHotSpot* with the increase in NoC size for showing its (*MinHotSpot*) fast solution in many-core NoC. In Fig. 9, we have simulated CNA applications with total 49-tasks for NoC size up to 256 cores. For NoC size of up to 49 cores, *MinHotSpot* provides the solution for 49-tasks within seconds (up to a minute). In 100-core NoC, *MinHotSpot* takes 27 seconds (per task) to generate near-optimal solution for a task. As execution time of the heuristic depends on the simulation platform, the execution time can be improved further with the increase in capacity of the platform (e.g., number of cores, processor frequency, memory). This fast solution also makes *MinHotSpot* applicable for run-time mapping. Therefore, in general, the *MinHotSpot* heuristic offers a fast and effective solution feasible for practical application and/or NoC size.

Next we evaluate the performance of both static and dynamic solution of *MinHotSpot* for larger network and application sizes.

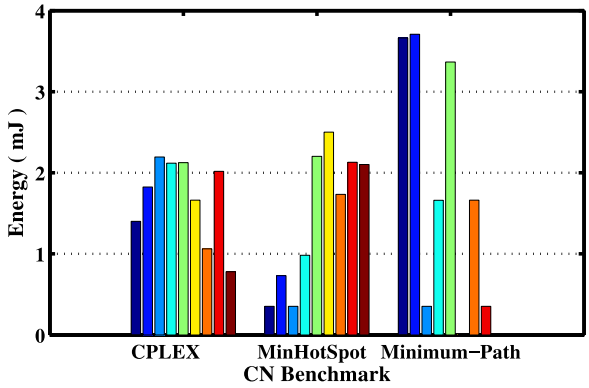

Fig. 8. Load distribution in a 3 × 3 NoC under CN E3S Benchmarks.

Table 2
Execution time (in seconds) comparison between CPLEX and *MinHotSpot*.

Applications	CPLEX	*MinHotSpot*
C	2300	0.02
N	2448	0.03
A	4224	0.05
T	17,307	0.06
O	2217	0.05
CN	14,053	0.06
CA	16,213	0.06
CT	56,947	0.08
NA	18,102	0.06
AT	61,830	0.09
NT	60,157	0.08
TO	40,315	0.11
CNA	49,617	0.09

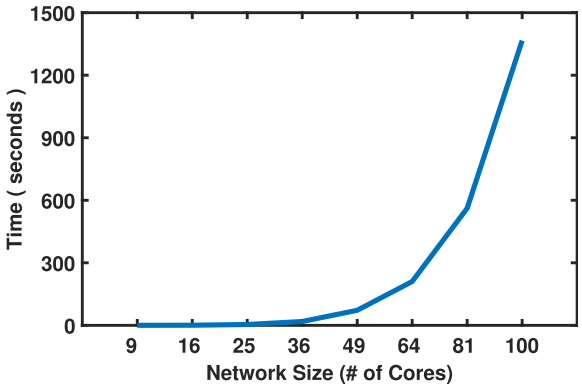

Fig. 9. Execution time of *MinHotSpot* at different network scales under CNA Benchmarks.

Dynamic solution of *MinHotSpot* is simulated to evaluate its effectiveness (in terms of hotspots and scalability) in run-time mapping. All the 19 applications (including sub-applications) with total 82-tasks in E3S benchmarks are combined and simulated for different network scales of up to 100-cores. As shown in Fig. 10, *MinHotSpot* heuristic performs exceedingly well (30-70% better in terms of peak load) compared to minimum-path. When the network resource exceeds the application demand by a huge margin, then the minimum-path solution reaches close to the *MinHotSpot* performance. Dynamic *MinHotSpot* solution provides almost (within 0–5%) the same energy (in milli-joule (mJ)) performance as static solution. This proves that dynamic *MinHotSpot* can also find global near optimal solution like static *MinHotSpot*, though dynamic *MinHotSpot* mapping doesn't know about all the tasks and applications demand before the allocation begins.

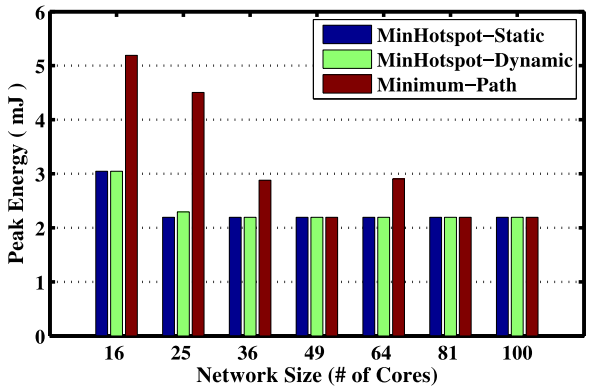

Fig. 10. Peak load comparison at different network scales under combined E3S Benchmarks.

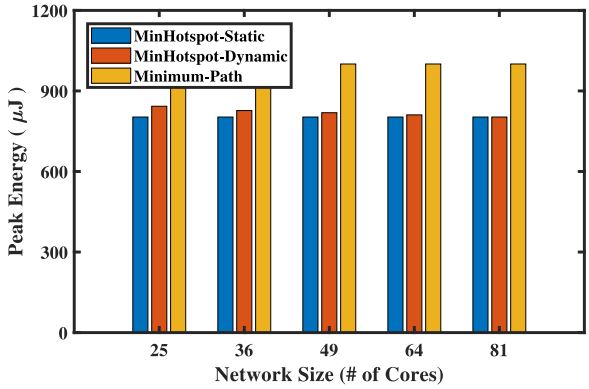

Fig. 11. Ultrasound Application load comparison.

6.2. *Performance trend under large-scale problem settings*

Next we evaluate the heuristics performance trend by varying different system parameters under large-scale settings. First, we study the impact of NoC size in energy hotspots. COSMIC benchmark suite is used here because it is based on real applications with a large number of tasks per application. Three applications from COSMIC are mapped in our simulations: Ultrasound (U) – 82 tasks, Reed-Solomon code decoder (RS-Decoder) – 182 tasks, and Reed-Solomon code encoder (RS-Encoder) – 262 tasks. We also simulate complex multiple applications by mixing all the above mentioned three applications as U-D-E. Because of the large task count in this suite, we allow mapping of multiple tasks from an application to a node based on a condition that those tasks do not communicate with other. Two communicating tasks are not mapped to the same node to increase the task parallelism.

As shown in Figs. 11–14, both static and dynamic *MinHotSpot* heuristics have 25–100% lower peak loads (in micro-joule (μJ)) than minimum-path approach under various applications. Performance of *MinHotSpot-Dynamic* is within 5–10% of *MinHotSpot-Static*. Peak load (energy) decreases with the expansion of the network sizes for RS decoder and encoder applications for *MinHotSpot*. For the Ultrasound application, peak load remains stable for all network scales because of the presence of a few dominating tasks. It is inspirational to observe the fast running speed of *MinHotSpot* that can generate solution for large number of tasks and network nodes in a short time (in seconds or up to minutes). The peak load optimization is performed in a way that assigns high-demanding tasks to light-loaded nodes to evens out the workload distribution. Fig. 15 illustrates the traffic distribution in a 5 × 5 NoC where RS-Decoder with 182 tasks are placed on it. As we found, the peak load is 1335 Mega-bit (Mbit) while the average node load is 1324 Mbit which implies the effectiveness (load-balanced distribution for minimizing hotspots) of the heuristic optimization.

Peak load and per-link cost of *MinHotSpot* solution under different network scales and application sizes are observed based on application task graphs generated by the task graph generator [25]. The experimental setup is summarized in Table 3. For fair comparison, the ratio of overall communication to computation demands is kept the same throughout the experiments. As can be seen in Fig. 16, the peak load/energy (in joule (J)) generally decreases when the network size scales up. It (peak load) drops sharply at the beginning as loads are distributed to higher number of resources. When the workloads are already evened out with excessive resources in a large network, no further reduction can be achieved and eventually the curve becomes flat. Under the same network scale, the peak load grows when the number of tasks increases because, given the same demand per task, more tasks result in higher total demand, competing for the limited computation and communication resources. The network size shows similar impact on the link cost (in milli-meter square (mm^2)) as shown in Fig. 17.

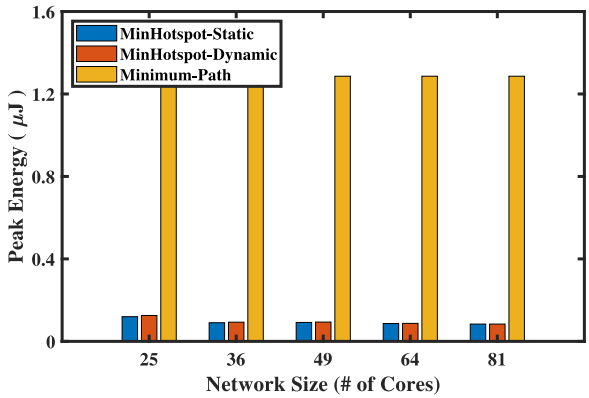

Fig. 12. RS-Decoder Application load comparison.

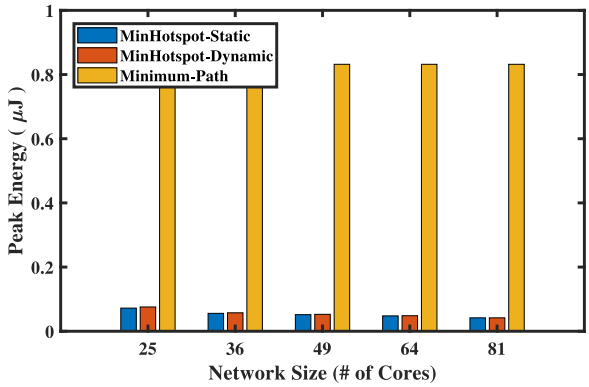

Fig. 13. RS-Encoder Application load comparison.

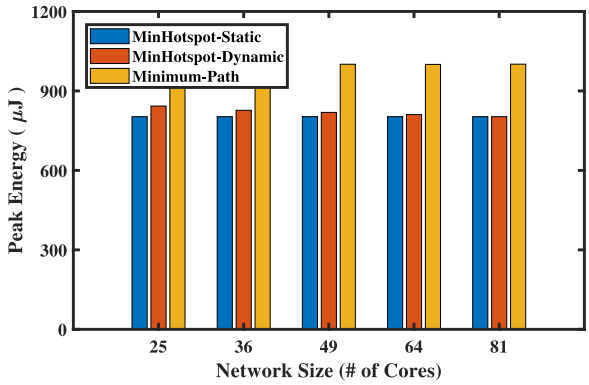

Fig. 14. U-D-E Applications load comparison.

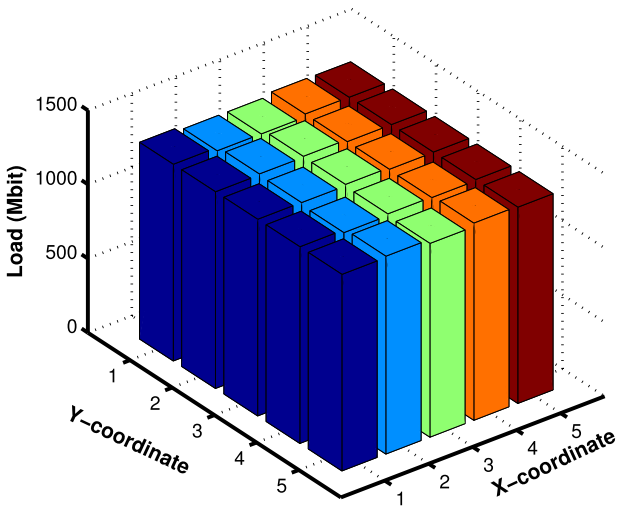

Fig. 15. Traffic distribution of RS-Decoder in a 5 X 5 NoC under *MinHotSpot*.

We further study the performance trend by varying the number of applications. In Fig. 18, each curve corresponds to a given set of tasks, which are randomly distributed among different number of applications. We observe that the workloads of the same set of tasks are always evenly distributed no matter how we group them into different applications. When the number of applications increases, we also observe a slight decrease in the peak load (in joule) as the tasks are more loosely coupled when separated into more applications. Traffic can thus be more evenly distributed with less communication dependency. The link cost (in mm^2) is illustrated in Fig. 19. It is aware that the network resource configuration (e.g., link capacity) is closely related to the workload distribution. As the communication dependency decreases with the increase of the number of applications, the tasks can be more contiguously allocated

Table 3
Simulation configuration of Randomly Generated Problems.

Application Count	4–10
Task Count	20–60
Total Computation Demand	100–500 GFLOPS
Total Communication Demand	150–750 Gbps
Communication to Computation Demand Ratio	1.5
No. of Cores	25–100
Core Capacity	30 GFLOPS
Data Rate	1 Gbps
Computation/Communication Energy Coefficient	50 picojoule
Total Link Cost Budget (area)	2.5 mm^2

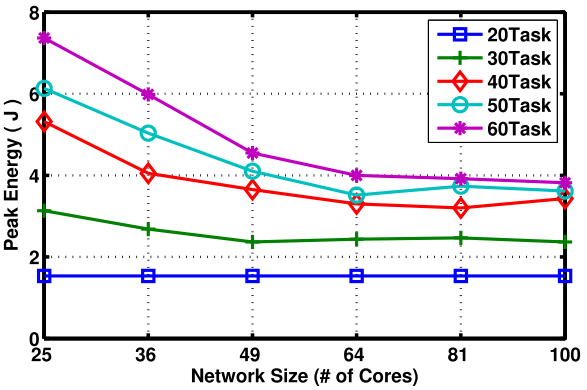

Fig. 16. Peak Load under different network scales (random graphs) under *MinHotSpot*.

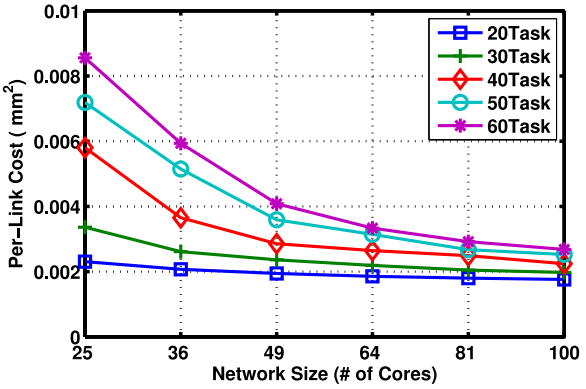

Fig. 17. Per-Link cost under different network scales (random graphs) under *MinHotSpot*.

to improve bandwidth usage, resulting in more efficient bandwidth utilization. Consequently, the link cost for heterogeneous configuration can be well controlled and improved.

6.3. Real system simulation

We have evaluated mapping solutions from *MinHotSpot* and minimum-path using gem5 platform [20] with GARNET, a detailed cycle-accurate NoC model, to verify the performance (latency and throughput) of heuristics in real system. We have modified GARNET by using different latencies (to mimic different link widths) for different links to support heterogeneous link bandwidth requirements in task-resource allocated NoC. We have integrated mapping solutions (of E3S benchmarks) in gem5 as a traffic type. We have simulated mapping solutions using 4 × 4 2D-mesh NoC on gem5 as our communication architecture. We run the simulations in fixed-pipeline mode for 1000 cycles with maximum number of packets to be injected by each node is set at 50,000. gem5 configurations are shown in Table 4. In almost all the E3S benchmarks, latency and throughput performance of *MinHotSpot-Static* are 30–200% better than minimum-path approach, as shown in Figs. 20 and 21. Latency and throughput from *MinHotSpot-Dynamic* are within 0–10% of *MinHotSpot-Static* performance, which shows the adaptability of *MinHotSpot* in dynamic systems. Figs. 22 and 23 show the benefits of heterogeneous links of low and high capacity bandwidths (e.g., *MinHotspot-Hetero*) over high capacity

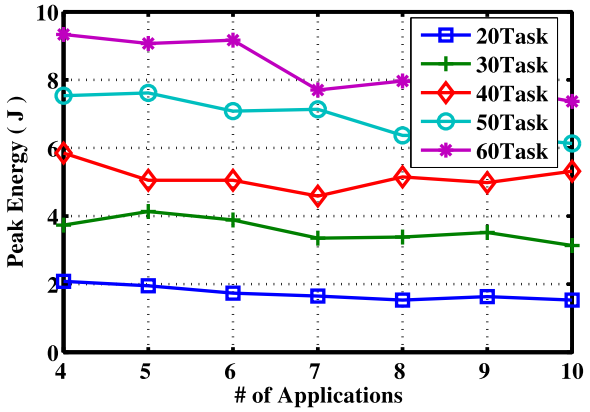

Fig. 18. Peak Load under different no. of applications (random graphs) under *MinHotSpot*.

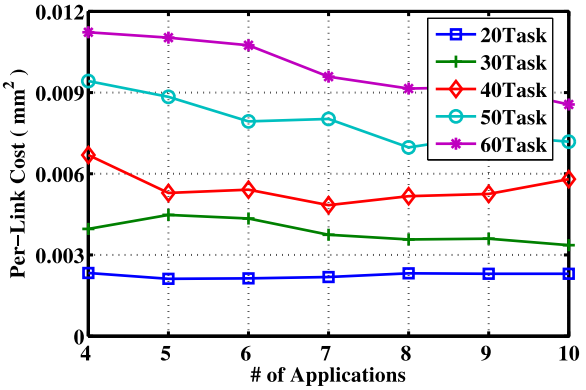

Fig. 19. Per-Link cost under different no. of applications (random graphs) under *MinHotSpot*.

Table 4
gem5 Simulation configuration.

CPU Frequency	2 GHz
Link Frequency	1 GHz
No. of Virtual Networks(VN)	3
No. of Virtual channels(VC) per VN	4
No. of Buffers per VC	4
Network Size	4 × 4 2D Mesh
No. of Cores	16
No. of Routers	16
Routing	XY-Routing
Flit Width	128-bit
Max. Packets per Node	50,000
Simulation Cycle	1000

homogeneous link bandwidth (e.g., *MinHotspot-1Cycle*) for multi-application CAN benchmarks. A higher bandwidth link has lower latency (in cycle) compared to a low-bandwidth link. So *MinHotspot-Hetero* (with both low and high capacity links) achieves almost the same overall chip latency (in nanoseconds (ns)) and throughput (in gigabit per sec (Gbps)) performance as all high capacity *MinHotspot-1Cycle* solution. This shows the advantages of heterogeneous link solution, which can achieve the same performance as homogeneous link (with high capacity) solution while reducing the energy hotspots and overall chip energy consumption.

6.4. Power evaluation

Power data for E3S benchmarks is derived by using DSENT [22] power tool in a 4 × 4 2D-Mesh NoC. DSENT configurations are shown in Table 5. We retrieve the power consumption of individual link using the link width and latency (delay). Then, we calculate the overall power consumption by summing up the links power based on the heterogeneous link design (from mapping heuristics). Leakage power is not considered here as it remains same irrespective of mapping heuristics result as we are not turning-off the resources in this work. As shown in Fig. 24, dynamic power consumption (in milli-watt (mW)) of *MinHotSpot* heuristics are almost same as minimum-

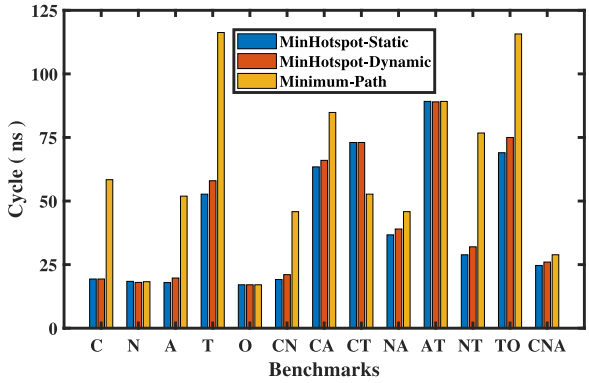

Fig. 20. Latency comparison in a 4 × 4 NoC under E3S Benchmarks.

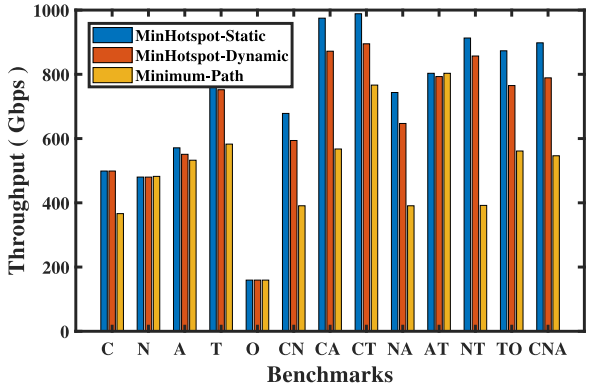

Fig. 21. Throughput comparison in a 4 × 4 NoC under E3S Benchmarks.

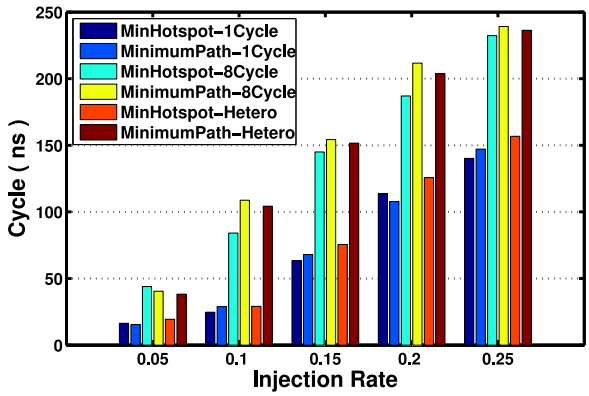

Fig. 22. Latency comparison in a 4 × 4 NoC under CAN Benchmarks.

path power consumption, where performance (hotspots, latency, and throughput) of *MinHotSpot* heuristic is significantly better than minimum-path. Power consumption of *MinHotSpot-Dynamic* is slightly higher than that of *MinHotSpot-Static*. So *MinHotSpot* heuristic improves the NoC performance (latency and throughput) and reduces energy hotspots without (almost) increasing the power.

7. Possible extensions

This work can be extended in several different directions, as described below.

7.1. Memory controller traffic

In this work, we mainly focus on cache traffic, where two cores or tasks communicate with each other. It can happen that a task's

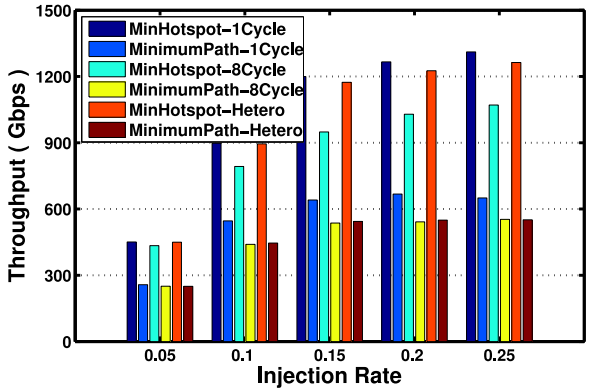

Fig. 23. Throughput comparison in a 4 × 4 NoC under CAN Benchmarks.

Table 5
DSENT Simulation configuration.

Technology	45 nm
Operating Frequency	1 GHz
No. of Virtual Networks(VN)	3
No. of Virtual channels(VC) per VN	4
No. of Buffers per VC	4
Network Type	2D Mesh
No. of Cores	16
No. of Routers	16
No. of Cores per Router	1
No. of Input Ports	5
No. of Output Ports	5
Flit Width	64-bit
Buffer Model	64-bit
Crossbar Model	Multiplexer Crossbar
Switch Allocator Model	Matrix Arbiter

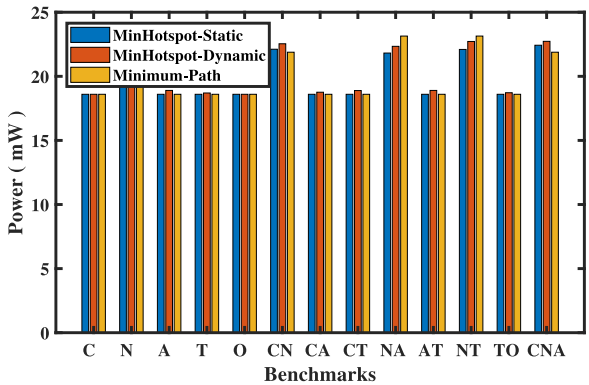

Fig. 24. Power comparison in a 4 × 4 NoC under E3S Benchmarks.

demand may not be present in shared cache or local cache. In which case, the request has to go through memory controller to get the required data from main memory. Memory controller placement is another NP-hard problem. Therefore, we assume that memory controller placement is configured before the task mapping and optimization starts. Depending on the placement of the memory controller(s), our proposed LP modeling will work for memory-controller traffic in the same way as cache traffic except for computation traffic. Computation traffic and corresponding constraint will not be applicable for memory-controller node, as memory-controller is only used for communication. Our *MinHotspot* heuristic also doesn't need any major change for memory-controller traffic. However, because of the existence of two different traffic types, *MinHotspot* heuristic now needs to check and identify traffic as memory-controller or cache traffic and optimizes the mapping and configuration accordingly. Based on the placement of memory-controller(s), task mapping options will be explored and optimal placement of task will be finalized by considering both cache and memory controller traffic.

7.2. Voltage and frequency scaling

Instead of assigning maximum voltage at every NoC resources, we can assign the required voltage at NoC nodes and links depending on the task demands. For example, based on the required computation at nodes, node voltage level can be dynamically configured to reduce power consumption. Voltage level of links can be scaled based on the communication demands of tasks. Voltage regulator can select the voltage level from available voltage configurations, like 0.8, 0.9,1.0,1.1, and 1.2 V (Volt). In the same way, frequency of the node and link can be scaled depending on the task demands, where lower frequency reduces voltage requirement. This dynamic assignment of voltages and frequencies can significantly reduce the hotspot (maximum) power and overall chip power consumption.

7.3. Resource power gating

In dynamic *MinHotspot* solution, idle resources (core, router, and link) can be power-gated to reduce power consumption further. Due to the advancement of transistor technology, transistor size is decreasing, and that eventually increases leakage power proportion relative to the overall component power consumption. Power gating eliminates leakage power consumption at idle resources, and thus can significantly reduce overall power consumption.

7.4. Express channel-based NoC mapping

This work can be extended for express channel (long link) based many-core on-chip networks. Express channels can be any of the following interconnect technologies: wired, wireless, and optical. Express channel minimizes the communication distance (hop-count) and therefore reduces the latency. Further, the communicating tasks can be placed at distant nodes to minimize the contiguous allocation for preventing the formation of hotspots while corresponding communicating tasks can use the express channels to reduce communication latency. Hierarchical routing can be used for communication between short (regular) channels and express channels layers. Dimension-order routing (e.g., XY-Routing) can be used for communication within the same layer. Express channels can be effectively used to minimize latency while the loads are distributed uniformly throughout the network to minimize the hotspots.

8. Conclusion

The capacity and cost-constrained application mapping problem in many-core networks-on-chip has been addressed in this paper, which aims to balance the load distribution to avoid energy overheating in the networks and overall chip system. During task-resource co-allocation, networks-on-chip are configured heterogeneously according to the computation and communication demands of the applications. The optimization of task placement and resource configuration has been modeled in linear programming, and implemented by using an optimization solver. When the problem dimension becomes extremely large, the high computation complexity renders optimal solution intractable. To this end, a fast and practical heuristic algorithm namely *MinHotspot* has been proposed to provide feasible solutions for both design-time and run-time task-resource co-allocation in large-scale systems. The simulation results obtained from the real world benchmark suites and random generated problem domains have demonstrated the effectiveness and efficiency of the proposed mathematical model and heuristic schemes. Future directions of this work have been discussed with the possible integration of memory-controller traffic, voltage/frequency scaling, power gating, and express channels in many-core on-chip networks.

Acknowledgments

The authors would like to thank anonymous reviewers and the editor for valuable recommendations to improve this article. This work was partially supported by the National Science Foundation, under grant CCF-0845983.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.compeleceng.2018.04.019.

References

[1] Kas M. Toward on-chip datacenters: a perspective on general trends and on-chip particulars. J Supercomput 2012;62(1):214–26. http://dx.doi.org/10.1007/s11227-011-0703-4.
[2] World's First 1,000-ProcessorChip, https://www.ucdavis.edu/news/worlds-first-1000-processor-chip/.
[3] 48-core SoC, https://www.qualcomm.com/news/onq/2017/08/20/introducing-qualcomm-falkor-cpu-core-purpose-built-cloud-workloads.
[4] Micheli GD, Seiculescu C, Murali S, Benini L, Angiolini F, Pullini A. Networks on chips: from research to products. Proceedings of ACM/IEEE design automation conference (DAC). 2010. p. 300–5. http://dx.doi.org/10.1145/1837274.1837352.
[5] Hoskote Y, Vangal S, Singh A, Borkar N, Borkar S. A 5-ghz mesh interconnect for a teraflops processor. IEEE Micro 2007;27(5):51–61. http://dx.doi.org/10.1109/MM.2007.4378783.
[6] Murali S, De Micheli G. Bandwidth-constrained mapping of cores onto noc architectures. Proceedings of IEEE design, automation and test in Europe (DATE). 2004. p. 896–901. http://dx.doi.org/10.1109/DATE.2004.1269002.
[7] Hu J, Marculescu R. Energy and performance aware mapping for regular noc architectures. IEEE Trans TCAD 2005;24(4):551–62. http://dx.doi.org/10.1109/

TCAD.2005.844106.
[8] Murali S, Coenen M, Radulescu A, Goossens K, De Micheli G. A methodology for mapping multiple use-cases onto networks on chips. Proceedings of IEEE design, automation and test in Europe (DATE). 2006. p. 1–6. http://dx.doi.org/10.1109/DATE.2006.244007.
[9] Chou C-L, Marculescu R. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. Proceedings of IEEE/ACM international conference on hardware/software codesign and system synthesis (CODES + ISSS). 2007. p. 161–6.
[10] Fattah M, Daneshtalab M, Liljeberg P, Plosila J. Smart hill climbing for agile dynamic mapping in many-core systems. Proceedings of ACM/IEEE design automation conference (DAC). 2013. p. 1–6.
[11] Chou C-L, Marculescu R. Contention-aware application mapping for network-on-chip communication architectures. Proceedings of IEEE international conference on computer design (ICCD). 2008. p. 164–9. http://dx.doi.org/10.1109/ICCD.2008.4751856.
[12] Chen G, Li F, Son S, Kandemir M. Application mapping for chip multiprocessors. Proceedings of ACM/IEEE design automation conference (DAC). 2008. p. 620–5.
[13] Lu H, Yan G, Han Y, Fu B, Li X. RISO: Relaxed network-on-chip isolation for cloud processors. Proceedings of ACM/IEEE design automation conference (DAC). 2013. p. 1–6.
[14] Reza MF, Zhao D, Wu H. Task-resource co-allocation for hotspot minimization in heterogeneous many-core nocs. Proceedings of ACM international great lakes symposium on VLSI (GLSVLSI). 2016978-1-4503-4274-2. p. 137–40. http://dx.doi.org/10.1145/2902961.2903003.
[15] Borkar S. Exascale computing - a fact or a fiction? Keynote at IPDPS. 2013. http://dx.doi.org/10.1109/IPDPS.2013.121.
[16] IBM CPLEX Optimization Studio, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[17] Fiduccia CM, Mattheyses RM. A linear-time heuristic for improving network partitions. Proceedings of the 19th design automation conference (DAC). 19820-89791-020-6. p. 175–81.
[18] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell Syst Techn J 1970;49(2):291–307. http://dx.doi.org/10.1002/j.1538-7305.1970.tb01770.x.
[19] Lofberg J. YALMIP: a toolbox for modeling and optimization in matlab. Proceedings of CACSD. 2004. p. 284–9. http://dx.doi.org/10.1109/CACSD.2004.1393890.
[20] Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, et al. The gem5 simulator. SIGARCH Comput Archit News 2011;39(2):1–7. http://dx.doi.org/10.1145/2024716.2024718.
[21] Agarwal N, Krishna T, Peh LS, Jha NK. GARNET: a detailed on-chip network model inside a full-system simulator. Proceedings of ISPASS. 2009. p. 33–42. http://dx.doi.org/10.1109/ISPASS.2009.4919636.
[22] Sun C, Chen C-HO, Kurian G, Wei L, Miller J, Agarwal A, et al. Dsent – a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. Proceedings of IEEE/ACM international symposium on Networks-on-Chip (NOCS). 2012. p. 201–10. http://dx.doi.org/10.1109/NOCS.2012.31.
[23] Dick R. Embedded system synthesis benchmarks suites (e3s), http://robertdick.org/tools.html.
[24] Wang Z, Liu W, Xu J, Li B, Iyer R, Illikkal R, et al. A case study on the communication and computation behaviors of real applications in noc-based mpsocs. Proceedings of IEEE computer society annual symposium on VLSI (ISVLSI). 2014. p. 480–5. http://dx.doi.org/10.1109/ISVLSI.2014.36.
[25] Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel and Distributed Syst 2002;13(3):260–74. http://dx.doi.org/10.1109/71.993206.

Md Farhadur Reza is currently a Post-Doctoral Scientist in the George Washington University. He received Ph.D. and M.Sc. degrees in computer science from University of Louisiana at Lafayette in 2017 and 2014, respectively. He received B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology in 2005. His research interests include energy-efficient and high-performance many-core chip.

Dan Zhao is currently an Associate Professor in the Department of Computer Science at Old Dominion University (ODU). Before joining ODU, she was an Associate Professor at University of Louisiana at Lafayette. She received Ph.D. degree from State University of New York at Buffalo in 2004. Her research focuses on system-on-chip, network-on-chip, embedded systems, design and test, etc.

Hongyi Wu is the Batten Chair of Cybersecurity and a Professor in the Department of Electrical and Computer Engineering at Old Dominion University (ODU). Previously he was a Professor at University of Louisiana at Lafayette. He received Ph.D. degree from State University of New York at Buffalo in 2002. His research focuses on networked cyber-physical systems for security/safety/emergency management.

Magdy Bayoumi is a Professor in the Department of Electrical & Computer Engineering at University of Louisiana at Lafayette. He received Ph.D. degree from University of Windsor. He has published over 300 papers. He is an IEEE Fellow. He has graduated about 50 Ph.D. and 175 Master's students. His research focuses on VLSI, system-on-chip, internet-of-things, communication and networks, etc.