Energy and communication aware task mapping for MPSoCs

Tahir Maqsood a, Nikos Tziritas b,*, Thanasis Loukopoulos c, Sajjad A. Madani a, Samee U. Khan d, Chen-Chung Xu b, Albert Y. Zomaya g

a Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan
b Chinese Academy of Sciences, China
c University of Thessaly, Lamia, Greece
d Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA
e National Science Foundation, Arlington, VA 22230, USA
f Wayne State University, Detroit 48202, USA
g University of Sydney, Sydney, Australia

HIGHLIGHTS

- Introduce a new class of bin packing problem, i.e., variable benefit bin packing.
- A Knapsack based bin packing algorithm is proposed for workload consolidation.
- Proposed Knapsack algorithm reduces both energy consumption and network load.
- Various mapping heuristics are proposed to map the bins onto cores of MPSoC.
- Pareto-efficient algorithm to explore solutions in two dimensions simultaneously.

ARTICLE INFO

Article history:
Received 19 August 2015
Received in revised form 31 December 2017
Accepted 28 March 2018
Available online xxxx

Keywords:
Task mapping
Core mapping
Energy and communication aware mapping
Multi-processor system-on-chip (MPSoC)
Network-on-chip (NoC)

ABSTRACT

Minimizing energy consumption and network load is a major challenge for network-on-chip (NoC) based multi-processor systems-on-chip (MPSoCs). Efficient task and core mapping can greatly reduce the overall energy consumption and communication overhead among the interdependent tasks. In this paper, we propose a novel Knapsack based bin packing algorithm for workload consolidation that places tasks in such a manner that utilization of available processing elements is maximized, while network overhead, regarding the communication among the tasks, is minimized. We also propose a task swapping algorithm that attempts to further optimize the task placement produced by the bin packing algorithms. Moreover, several core mapping techniques are implemented and the performance of each technique is evaluated under varying configurations. In addition, we also apply a Pareto-efficient algorithm, on top of the bin packing algorithms, attempting to explore the solution in two dimensions, i.e., energy consumption and network load. The experimental results show that the proposed Knapsack based bin packing algorithm coupled with the Pareto-efficient algorithm achieves significant energy savings and reduction in network load as compared to state-of-the-art algorithms, as well as the greedy algorithm. Particularly, the Pareto-efficient algorithm when applied on top of the Knapsack algorithm shows on average 50% and 55% reduction in energy consumption and network load as compared to the greedy algorithm, respectively. While the proposed Pareto-efficient algorithm applied with Knapsack algorithm also demonstrate superior performance compared to three other state-of-the-art heuristics.

© 2018 Elsevier Inc. All rights reserved.

1. Introduction

With the advancement in integration technologies and nanotechnology, thousands of cores can be integrated onto a single chip referred to as Multi-Processor System-on-Chip (MPSoC). The cores also referred to as Processing Elements (PEs), can be general purpose processors, field programmable modules, or digital signal processors [9]. The PEs of MPSoC often need to communicate and share data with other PEs. In this regard, Network-on-Chip
(NoC) has emerged as a viable solution to provide efficient and scalable communication architecture for next generation MPSoCs [15,20].

In contrast to traditional single processor systems, MPSoCs can provide increased computational capabilities at a low cost with much improved energy efficiency [10]. The current trend clearly indicates the dominance of multi-core processor systems in the forthcoming personal and household devices, such as mobile phones, tablets, and laptops. Researchers anticipate that the majority of future high performance computing systems will be based on MPSoCs [10,31]. However, the performance and energy efficiency of MPSoCs largely depend upon the optimal task and core mapping [15,29]. Energy efficiency and power management are very crucial for MPSoCs, because, it not only affects the operating cost but also influences the lifetime and reliability of the underlying system due to increased heat dissipation [18,28].

Similarly, it is equally important to place the frequently communicating tasks close to each other. Placing interdependent tasks in close proximity leads to reduction in both network load and energy consumption. The rationale for choosing the network load as a performance metric is that reduction in network load leads to efficient utilization of available NoC bandwidth and reduction in latency. Consequently, more applications can be accommodated in the system when considering the same resources. Moreover, reduction in network overhead also leads to reduction in energy consumption. Furthermore, reduction in the communication latency leads to significant improvements in application-level system throughput because the waiting time of interdependent tasks is reduced [25]. This clearly indicates that besides intra-core computation, inter-core communication has significant impact on the overall performance and energy consumption of the whole system [15]. Therefore, the focus of this work is turned on energy efficient and communication-aware task and core mapping for NoC based MPSoCs.

The main contributions of this work are summarized as follows.

- We approach the optimization problem of energy and communication-aware mapping by introducing a new class of bin packing problem, the Variable bEnefit Bin pAcing Problem (VEBAP). To address VEBAP, we transform the problem into a variation of knapsack optimization problem. The aforementioned variation is called Variable Benefit Knapsack Problem (VBKP) where the benefits of packing objects are not fixed but variable (details can be found in Section 5.2). Moreover, we use dynamic programming approach to design an elegant algorithm to solve VBKP.
- Employing the well-known 0–1 Knapsack only reduces energy consumption while the immense advantage of proposed knapsack optimization is that it reduces both energy consumption and network overhead simultaneously. The benefits of using the proposed Knapsack optimization are two-fold. Firstly, maximum number of tasks will be packed on the available processing elements leading to better resource utilization and lower energy consumption. Secondly, the communication overhead among the interdependent tasks is minimized.
- Proposed VBKP algorithm maximizes the internal (intra-core) communication by placing heavily communicating tasks on the same bin. However, VBKP algorithm ignores the external (inter-core) communication overhead incurred between tasks that are packed in different bins. Therefore, it is of paramount importance to also minimize the external network load. To this end, we propose a task swapping algorithm (Section 5.3) that attempts to further optimize the task mapping produced by knapsack algorithm, targeted to reduce the external communication overhead.
- Once tasks are packed in bins, mapping of bins onto the processing cores of NoC based MPSoC has significant impact on network load and the resulting communication energy consumption. Therefore, we propose various bin mapping heuristics to map the bins onto the processing cores of MPSoC considering the communication overhead (details can be found in Section 6).
- In addition to bin packing and bin mapping algorithms, we have implemented a Pareto-efficient algorithm (PEA) that attempts to explore the solution space simultaneously in two dimensions, i.e., energy consumption and network load. The PEA finds a set of Pareto optimal solutions belonging to Pareto frontier (Section 7). The experimental evaluation reveals interesting trade-offs between energy consumption and network load incurred within the system. For the reasons of high computational time and slow convergence rate of PEA for large mesh NoCs, we also developed a clustering-based PEA. To improve the convergence rate, clustering-based PEA considers each row of NoC, representing the cores belonging to respective row, as an item of solution tree instead of each individual core.
- We analyze the effectiveness of proposed algorithms through extensive simulations. The experimental results of proposed techniques are compared against three other state-of-the-art heuristics proposed in [17,30,19].

The remainder of the paper is organized as follows. Section 2 reviews the prior research related to our work. The system model and problem formulation are presented in Section 3. Overview of the proposed approach is provided in Section 4, while various bin packing and bin placement heuristics are described in Sections 5 and 6, respectively. Pareto optimization and the proposed Pareto-efficient algorithm are discussed in Section 7. Complexity analysis of the proposed algorithms is presented in Section 8. Experimental evaluation and results are discussed in Section 9. Finally, Section 10 concludes the paper along with some directions for future work.

2. Related work

In NoC based MPSoCs, task mapping plays a vital role in energy efficiency and performance of the system. Therefore, energy efficient task and application mapping have attracted considerable attention in the recent past from both academia and industry. In this regard, several solutions have been proposed in the scientific literature [6,9,14,15,20,26,35]. In [6], authors presented a Genetic Algorithm (GA) based application mapping technique that attempts to reduce the communication energy consumption while improving the buffer utilization of NoC routers. In [14], authors proposed an Integer Linear Programming (ILP) based formulation to optimize the energy yield of MPSoCs considering the effects of process variation. However, the proposed framework considers shared bus based architecture instead of NoC. In [35], authors proposed an optimized branch and bound (B&B) algorithm that attempts to reduce the communication energy consumption by mapping the heavily communicating tasks to closer PEs. Unfortunately, in the proposed architecture only a single task can be mapped on each PE. In [15], authors have presented a unified model for task scheduling and core mapping for regular mesh, irregular mesh, and custom NoCs targeted to reduce the energy consumption and application execution time. The authors proposed a Mixed ILP (MILP) based formulation and a heuristic to speed up the exploration of the solution. However, the proposed technique cannot be compared with our work because it is primarily targeted towards irregular mesh and custom NoCs. Another work proposed in [20] has also considered irregular and custom mesh networks for task
mapping. The authors have proposed a mixed integer quadratic programming (MIQP) based formulation and two heuristics. The first heuristic is based on successive relaxation that attempts to reduce the algorithm's run-time, while the second heuristic uses GA to achieve a higher quality solution in terms of reduced communication energy. However, proposed heuristics attempt to reduce the average hop count and the resulting communication energy consumption of NoC, with the computational energy consumption being not considered at all.

Similarly, reduced network overhead and communication latency is another important aspect that needs to be considered by NoC based MPSoCs. Consequently, several techniques have been proposed in this regard [1,4,8,11–13,33]. In [13], authors proposed mapping algorithms to reduce the communication energy consumption, while eliminating the communication contention. The authors in [12] have proposed a task to core mapping heuristic based on ant colony optimization technique to reduce the network traffic and communication energy consumption. However, in contrast to our approach, the authors have only considered the network load and resulting communication energy consumption, while ignoring computational energy consumption. Moreover, the proposed technique supports mapping of single task on each PE. Furthermore, the proposed technique has been evaluated on smaller NoC size, i.e., 2 × 4 mesh with less number of tasks.

The authors in [11], have proposed an extension to their previous work [12] by considering the simultaneous mapping of multiple applications. They reduce the algorithm's run-time by introducing an optimized initial mapping employing Tabu search. However, differently to our approach, the proposed technique only considers the communication energy consumption within NoC.

In [8], authors have proposed an ILP based task mapping technique to reduce the communication energy consumption. Moreover, the authors have also proposed migration of tasks from the faulty cores to improve the reliability of MPSoCs.

Considering the aforementioned related work, it is quite evident that majority of the works attempt to reduce network overhead and the corresponding communication energy consumption. Similarly, a small proportion of those works focused on reducing the computational energy consumption. There are also traces of works that attempt to jointly optimize the communication and computational energy consumption. There are also traces of work that attempt to jointly optimize the communication and computational energy consumption. Therefore, we have considered the following problem statement:

**Problem Statement:**

Consider a task graph $G=(V,E)$. $V$ is a set of tasks $v_i$, where $i=1,2,...,n$. $E$ is a set of edges $(v_i,v_j)$ representing the communication dependencies between $v_i$ and $v_j$. Each task $v_i$ has a computational requirement $C_i$ and a communication volume $V_i$. The task graph $G$ is a directed acyclic graph (DAG).

The problem is to find a mapping $f: V \rightarrow \{1,2,...,m\}$ of tasks to processing elements $P = \{1,2,...,m\}$ such that:

1. The total computational energy consumption is minimized:

   $E_{\text{comp}}(f) = \sum_{i=1}^{n} C_i f_i$

2. The total communication energy consumption is minimized:

   $E_{\text{comm}}(f) = \sum_{(v_i,v_j) \in E} V_{ij} f_i f_j$

3. The communication demands are satisfied:

   $\forall (v_i,v_j) \in E, (f_i,f_j,\Delta) \in \mathcal{M}$

where $\mathcal{M}$ is a set of available mappings of task pairs.

4. The mapping is feasible:

   $\forall v_i \in V, f_i \in P$

5. The maximum load on any processing element $P_j$ is not exceeded:

   $\sum_{v_i \in V} f_i \leq \text{load}(P_j)$

where $\text{load}(P_j)$ is the computational capacity of processing element $P_j$.

**Optimization:**

The objective is to minimize the total energy consumption $E_{\text{total}}(f) = E_{\text{comp}}(f) + E_{\text{comm}}(f)$ subject to the above constraints.

**Solution Approach:**

We propose an ILP based formulation that aims to minimize both communication and computational energy consumption for NoC based MPSoCs. However, the major drawback of ILP based techniques is that their run-time is very high and these are not scalable for large problem instances [29].

The authors in [11] have also proposed an extension of their previous work [8] by considering both computational and communication energy consumption. At design-time stage, for each fault scenario a corresponding energy efficient mapping is generated and stored in memory. When a specific fault occurs, the corresponding mapping is identified and applied at run-time. The drawback of this approach is that it requires additional memory space to hold the mapping for each fault scenario. Moreover, only the faults at the processing cores are considered, while the router or NoC link faults are not considered. However, differently to our proposed approach, the authors have specifically addressed the problem of task mapping in case of faults while minimizing the energy consumption.

Prior works that closely match our proposed techniques include [17,19,30]. In [17], authors have proposed a simulated annealing based task mapping algorithm that attempts to minimize the communication and computation energy consumption. Instead of using a random initial placement for simulated annealing, authors calculate an initial baseline mapping that aims to minimize the communication overhead among tasks. However, a major drawback of simulated annealing based solutions is the slow convergence rate. In [19], authors propose a task and core mapping heuristic, called MPMAP, that attempts to optimize computational and communication energy consumption. MPMAP works in two phases: (a) the initial task mapping phase, whereby tasks are mapped regarding the computational energy consumption; and (b) the iterative remapping phase, whereby tasks are moved to reduce communication energy consumption. Similarly, the author in [30] proposed a task mapping heuristic, called CastNet that attempts to minimize the energy consumption by placing the highly communicating tasks in close proximity. However, a major drawback of both MPMAP and CastNet is their large execution time. Moreover, in contrast to our approach, MPMAP optimizes computation and communication energy consumption separately. Whereas, CastNet does not explicitly consider the computational energy consumption while mapping tasks. The readers interested in further exploring the prior work in more detail are encouraged to see the surveys reported in [27,29]. Differently to the prior work mentioned above, in this paper we jointly tackle the issue of minimizing the computational energy and communication overhead for task and core mapping in NoC based MPSoCs using a new class of bin packing problem, i.e., VEBAP.

### 3. System model and problem statement

#### 3.1. Task model

In task model, the set $T$ represents the collection of heterogeneous tasks to be executed. The computational demands for a given task $t_i \in T$ are described by $\text{req}_i$. The logical ordering of sub-tasks (or simply tasks) of a large job is called a workflow. A workflow $G(T, E)$ is usually represented in the form of an Application Task Graph (ATG) as shown in Fig. 1. Each node in the ATG represents a task $t_i \in T$, while edges between nodes represent the volume of data needed to be exchanged between tasks. The set of all of the edges is denoted by $E$, while each edge $e_{ij}$ contained in $E$ signifies the communication dependencies between $t_i$ and $t_j$. The communication volume between $t_i$ and $t_j$ is captured by the weight $w_{ij}$ of $e_{ij}$. For instance, as shown in Fig. 1, the communication volume between $t_1$ and $t_2$ is 2.

#### 3.2. NoC model

The NoC model used here is similar to the model originally proposed in [28]. The set $P$ consists of a number of processing unit (PU) types. Each PU type $p_i$ can have multiple instances such that $P_i$ represents the $k$th instance of PU type $p_i$. The capacity of each processing element $P_j \in P$ is described as available computing...
resources per time unit and is denoted by $\lambda_j$. Each core, representing a processing element $p_j \in P$, is connected to the mesh based NoC through a router. NoC provides a scalable and modular architecture, whereby cores are interconnected through a router-based architecture. In Fig. 2, various components of NoC based MPSoC are presented. PE represents a processing element $p_j \in P$ while $R$ represents the routers that are used to route and forward data and control flits. NI refers to network interfaces used to connect PEs to the routers. Each router is connected to multiple routers through bidirectional links that can be divided into multiple logical channels. Moreover, the routers implement routing protocols and use buffers to temporarily hold the flits.

3.3. Energy model

The energy consumed within the system is mainly because of the communication between PEs and the computation performed by PEs. The amount of energy consumed due to communication and computation is analyzed below.

3.3.1. Communication energy

Bit energy calculation model has been used to calculate the communication or network energy consumption [2,16,23]. Eq. (1) is used to calculate the energy consumed for transmitting a single bit from core$_j$ to core$_k$.

$$EBit_{j,k} = (\eta_{j,k} \times E_{bit}) + (\eta_{j,k} \times 1 \times E_{lat}) + (2 \times E_{cput}) (1)$$

Specifically, $EBit_{j,k}$ represents the energy consumed for transmitting one bit from core$_j$ to core$_k$ in NoC. $E_{bit}$ is the dynamic energy consumed by the router (including wires, buffers, and logic gates) to transmit a single bit. Similarly, $E_{cput}$ is the bit dynamic energy consumed on the links between the router and the processing core. Whereas, $E_{lat}$ represents the bit dynamic energy consumed on links between two adjacent routers. Moreover, $\eta_{j,k}$ corresponds to the number of routers or hops that the bit passes through to reach core$_k$ from core$_j$. The value of $\eta_{j,k}$ is calculated through Eq. (2), which represents the minimum Manhattan distance between core$_j$ and core$_k$.

$$\eta_{j,k} = |X_j - X_k| + |Y_j - Y_k|$$

In Eq. (2), $X_j$ and $Y_j$ represent the column and row index of core$_j$ in the mesh based NoC, respectively. Let $B$ be a matrix of size $|P| \times |P|$ capturing the communication volume between PEs. Specifically, $b_{jk}$ reflects the amount of data exchanged between core$_j$ and core$_k$. It must be noted that $b_{jk}$ equals zero when $j = k$.

Consequently, given a task to core mapping, total network energy consumed by the NoC platform is calculated based on Eq. (3):

$$Energy_{NoC} = \sum_{V \in P} \sum_{V \in P} b_{jk} \times EBit_{j,k}$$

3.3.2. Computational energy

An MPSoC platform may be composed of heterogeneous processors. The source of heterogeneity may vary from one processor to another. The above is because processors may be structurally different or they may belong to different voltage-frequency islands. The energy consumed by a processing element $p_j \in P$ as a function of a given placement $F$ for the duration of time interval $[t_1, t_2]$ can be calculated based on the following model adapted from [3]:

$$ECore_{j}(t_1, t_2) = (idle_{j}^{t_1, t_2} \times p_{idle}) + \left(\frac{util_{j}^{t_1, t_2}}{p_{util}} \times p_{util} \right)$$

The energy consumed by processing core $p_j \in P$ during the time interval $[t_1, t_2]$ is expressed by Eq. (4). Note that Eq. (4) is split into two parts: (a) the first part concerns the static energy consumption, where $p_{idle}$ reflects the static power consumption, while $idle_{j}^{t_1, t_2}$ represents how long $p_j$ has been commissioned during $[t_1, t_2]$. And (b) the second part concerns the energy consumed when varying core’s utilization ignoring the static energy consumption. $util_{j}^{t_1, t_2}$ represents the utilization of $p_j$ during $[t_1, t_2]$, while $p_{util}$ reflects the dynamic power consumption. Note that $p_{util} = \frac{P_{max} - p_{idle}}{p_{idle}}$, where $P_{max}$ signifies $p_j$’s maximum power consumption. The total computational energy consumption of all the processing cores for a given placement $F$ and time interval $[t_1, t_2]$ can be calculated by Eq. (5):

$$Energy_{comp}(t_1, t_2) = \sum_{j=1}^{p} ECore_{j}(t_1, t_2)$$

3.4. Problem formulation

To outline a mathematical formulation of the problem at hand, we need to introduce the following. The mapping of tasks onto the processing cores is captured by a $|T| \times |P|$ matrix denoted by $F$. In $F$, $F_{ij}$ indicates whether task $t_i$ is hosted on processor $p_j$, or not. More specifically, $F_{ij}$ is one if $t_i$ is mapped on $p_j$, otherwise it is zero. The computing capacity of a core $p_j$ is represented by $cap_{p_j}$, while the computing demands of a task $t_i$ are denoted by $req_{cap_{p_j}}$. The utilization of $p_j$ is represented by $U_{j}(F, t)$ and can be calculated through Eq. (6). Eq. (7) states that utilization of $p_j$ cannot be greater than one. Energy consumption of a processing core $p_j$, given a placement $F$ and a time interval $[t_1, t_2]$, is represented by $ECore_{j}(t_1, t_2)$ and captured by Eq. (4). The total energy consumed by all the processing cores as a function of a placement $F$ for the time interval $[t_1, t_2]$ is represented by $Energy_{comp}(t_1, t_2)$ and is calculated through Eq. (5). The total data exchanged between processing cores $p_i$ and $p_j$ during the time interval $[t_1, t_2]$ is encoded by a $|P| \times |P|$ matrix denoted by $c_{i,j}^{t_1, t_2}$ ($c_{i,j}^{t_1, t_2} = c_{j,i}^{t_1, t_2}$). The total network load incurred within the system due to communication among the interdependent tasks under a placement $F$ and time interval $[t_1, t_2]$ is captured by Eq. (8). Based on the above formulation, the total energy consumed within the system can be calculated by adding the communication energy and computational energy consumption as shown in Eq. (9).

$$U_{j}(F, t) = \sum_{k=1}^{t} \frac{req_{cap_{p_j}}(t)}{cap_{p_j}}$$

$$\sum_{k=1}^{T} f_{k,x} \times \frac{\text{req} \_\text{cap}_x(t)}{\text{cap}_x} \leq 1, \forall x \in P \quad (7)$$

$$\text{comm}(F, t_1, t_2) = \sum_{k=1}^{T} \sum_{m=1}^{P} \sum_{n=1}^{P} c_{1,t_1} \times f_{k,x} \times f_{m,y} \times \eta_{k,y} \quad (8)$$

$$\text{Energy}_{\text{total}} = \text{Energy}_{\text{comp}} + \text{Energy}_{\text{NoC}} \quad (9)$$

The problem is formally stated as: Given a set of tasks $T$, a set of edges $E$, a set of workflows $G$, and a set of cores (PEs) $P$, try to find a feasible placement $F$ of these tasks onto the PEs such that to minimize: (a) computational energy consumption due to processing of these tasks; and (b) network overhead due to communication among the interdependent tasks.

4. Proposed approach

The algorithms proposed in this work are classified into three categories. In the first category, the tasks are packed onto bins, while in the second one the bins are mapped onto the cores of the underlying NoC based MPSoC. In the third category we apply Pareto optimization to explore the solution space in two dimensions simultaneously.

Regarding the first category, each computing resource is considered as a bin, while the tasks play the role of the items to be packed onto the bins. Because the conventional bin packing problem (BPP) is NP-hard, we claim that the aforementioned problem is also NP-hard. Therefore, we have developed a heuristic based solution for solving the bin packing through Knapsack optimization. To this end, a modified Knapsack based bin packing algorithm has been developed to pack (place) a set of tasks onto bins (knapsacks) such that the total benefit of bins is maximized. In the proposed knapsack variation, the benefit of an object (i.e., task) is not fixed (as in traditional knapsack) but variable. In addition, we have also implemented a Greedy algorithm that attempts to place the tasks with heavy edges to the same bin. Moreover, we propose a task swapping algorithm that can further optimize the task placement produced by the Knapsack or Greedy algorithms. The aforementioned bin packing algorithms are presented and discussed in Section 5.

Considering the second category, various bin mapping algorithms have been developed to map the bins onto the cores of the underlying NoC based MPSoC. The bin mapping algorithms include: (a) Extended Nearest Neighbor (Ex-NN), (b) Single Cluster (SC), (c) Multiple Clusters (MC), and (d) Bin Swapping (BS) algorithms. The performance of each of the aforementioned bin mapping algorithms has been analyzed by applying these algorithms on top of each of the aforementioned bin packing algorithms. Bin mapping heuristics are presented in Section 6. Finally, in Section 7, we apply Pareto optimization to find a set of non-dominated solutions with respect to two optimization criteria, i.e., energy consumption and network load. Figs. 3 and 4 provide taxonomy and illustrate the execution flow of the algorithms proposed as part of this work, respectively.

5. Task mapping algorithms

Bin packing has been used extensively for task scheduling in high-performance computing (HPC) systems, such as clusters, grid, and cloud [7,34]. However, in this work we tackle the aforementioned optimization problem, i.e., the problem of jointly optimizing the computation and communication energy consumption, by reducing it to a new class of bin packing problem, the Variable Benefit Bin Packing Problem (VBBP). We define the VBAP as “Given a set of interdependent (or related) objects, each with a weight and variable value (or benefit). We must find an assignment of the objects to a finite number of bins of variable volumes in such a way that the total number of bins is minimized and the total benefit of the objects is maximized.” The difficulty in solving such a problem is due to the fact that the benefit of packing an object is unfixated (variable) but related to the weighted edges connecting the respective object and the adjacent (or co-located) objects. For instance, consider the ATG depicted in Fig. 1, in which each task represents a VEBAP object. The benefit of packing $t_1$ in any bin will be zero if no other adjacent objects are packed or packable. However, the benefit will increase to 4 if $t_1$ and $t_2$ are co-located (or packed in the same bin). Moreover, the benefit can be increased to 36 (=32+4) if $t_1$, $t_2$, and $t_4$ are co-located. Last, we also propose a Greedy algorithm, as well as a task swapping algorithm that acts on top of VEBAP and Greedy.

5.1. Greedy algorithm

In Greedy algorithm, the edges in the ATG are sorted in a descending order according to their communication volume. Initially, the edge with the largest communication volume is chosen to be placed on the bin with the largest capacity. The capacity occupied by an edge $e_{ij}$ placed on a bin equals the size of $t_i$ plus the size of $t_j$. In case the size of $t_i$ and/or $t_j$ has already been counted by another edge, then the aforementioned size is ignored when assigning $e_{ij}$ on the respective bin. In case the assignment of an edge violates the capacity of the respective bin, then the next bin with the highest capacity is chosen. The above procedure iterates until there are no edges to be considered. It must be noted that when investigating the placement of an edge, we first consider placing the respective edge on an already open bin to improve the utilization of available resources, i.e., already open bins that leads to reduction in energy consumption. If there is no such open bin to accommodate the edge, then a new bin is opened.
5.2. VEBAP algorithm

As discussed earlier, a solution to our problem can be found by solving VEBAP. To address VEBAP, we use a routine to solve a variation of knapsack problem, where the benefits of the objects are not fixed but variable. The aforementioned variation is called variable benefit knapsack problem (VBKP), with its definition being: Given a set of objects, each of fixed weight but variable benefit, determine a collection of them such that the total benefit is maximized, under the constraint that total weight of the chosen items does not exceed a fixed size (called knapsack size).

To describe the routine solving VBKP, we first give some details on how part of the problem is transformed into VBKP. Particularly, each node in ATG corresponds to a VBKP object, with the benefit of the latter being equal to the total communication volume of the edges that are adjacent to objects inside knapsack. The weight of the respective object equals the computing requirements of the respective node. In that way, by maximizing the total benefit of VBKP, the communication load that does not burden the underlying network is also maximized. The above is because heavily communicated tasks are placed into the same bin (core).

The problem is actually approached in a slightly different way as described previously. Specifically, the edges contained in ATG play the role of VBKP objects, instead of having nodes playing that role. The weight of the rth object (edge) equals the sum of the computing requirements of the nodes adjacent to the corresponding edge. On the other hand, the benefit of an object equals the communication volume of the corresponding edge. The problem being solved remains the same as previously. VBKP routine (shown in Table 1) takes as input a list of edges that are in the form of VBKP objects (ObjList). The rth object inside the routine is denoted by ObjList[r]. The benefit and the weight of the rth object is defined by ObjList[r].benefit and ObjList[r].weight, respectively. The nodes adjacent to the edge represented by the rth object are defined as ObjList[r].n1 and ObjList[r].n2, while their computing requirements are denoted as ObjList[r].n1.weight and ObjList[r].n2.weight, respectively.

The objects participating in the explored solutions are stored into an S matrix, while the total benefit of them is stored into a K matrix. Given that the total number of Objects equals N and the capacity of knapsack equals W, the size of each matrix is \((N + 1) \times (W + 1)\). The solutions are explored in a dynamic programming manner. Particularly, each element of \(S/K\) is set to null/zero (lines 1–2). The rth row of \(S/K\) represents the rth object, while the rth column represents the current knapsack weight and equals c. When the indices of \(S/K\) are r and c, then they correspond to the sub-problem solution for r objects assuming that the size of knapsack equals c. Specifically, \(K[r][c]\) represents the benefit of the solution containing r objects and knapsack size of c. On the other hand, \(S[r][c]\) represents the objects used by the sub-problem solution containing r objects and knapsack size of c. The calculation of \(K[r][c]\) takes place as follows. If r or c equals zero, then \(K[r][c]\) equals zero. When the weight of the object under consideration is greater than c, then the current sub-problem solution equals to that of \(r - 1\) objects with knapsack size of c (line 17). Otherwise (lines 6–15), we choose the maximum of the below sub-problem solutions (line 15): (a) the sub-problem solution for \(r - 1\) objects with knapsack size of c (line 7). (b) The benefit of rth object plus the benefit of the sub-problem solution of \(r - 1\) objects with knapsack size of c minus the weight of the rth object (line 8). (c) The benefit of rth object plus the benefit of the sub-problem solution of \(r - 1\) objects with knapsack size of c minus the weight of the first node of the rth object (line 10). Note that the above holds only when ObjList[r].n1 belongs to one of the objects contained in \(S[r-1][\cdot] - ObjList[r].n1.weight\) (line 9). (d) The benefit of rth object plus the benefit of the sub-problem solution of \(r - 1\) objects with knapsack size of c minus the weight of the second node of the rth object (line 13). Note that the aforementioned holds only when ObjList[r].n2 belongs to one of the objects contained in \(S[r-1][\cdot] - ObjList[r].n2.weight\) (line 12). The intuition behind (c) and (d) is to seek for a sub-problem solution with \(r - 1\) objects that employs one of the nodes of the current object to be explored. In that way, we can reuse a sub-problem solution of \(r - 1\) objects by reducing the current knapsack size by only ObjList[r].n2/1.weight units.

As discussed earlier, VEBAP algorithm (shown in Table 2) employs the VBKP routine to solve the problem. Specifically, it takes as input the ATG and a list of bins. It converts all of the edges contained in ATG into VBKP objects and puts them in a list called ObjList (lines 1–2). Next, for each bin in the list, it calls VBKP routine giving it as input the list of VBKP objects and the size of the bin under consideration (line 4). The objects returned by VBKP routine are stored in a list and removed from ObjList (line 7). The nodes contained in the objects returned by VBKP routine are assigned bin under consideration (line 5), with the size of the latter being updated accordingly (line 6).

5.3. Task swapping algorithm

The task swapping algorithm (TSA), shown in Table 3, attempts to further optimize the task placement produced by the Greedy/VEBAP algorithm. Specifically, Greedy/VEBAP algorithms optimize the communication overhead by placing heavily communicating tasks on the same bin. In other words, they maximize the communication load (internal load) between tasks placed on the same bin. However, they ignore the network overhead incurred between

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBKP Routine.</td>
</tr>
<tr>
<td><strong>Algorithm 1: VBKP Routine</strong></td>
</tr>
<tr>
<td>Input: ObjList (list of Objects), W (size of knapsack)</td>
</tr>
<tr>
<td>1 (K[r][c] = 0) for each (r, c)</td>
</tr>
<tr>
<td>2 (S[r][c] = NULL) for each (r, c)</td>
</tr>
<tr>
<td>3 FOR (r = 1) to ObjList.size</td>
</tr>
<tr>
<td>4 FOR (c = 1) to (W)</td>
</tr>
<tr>
<td>5 IF ObjList[r].weight (\leq c)</td>
</tr>
<tr>
<td>6 (B3 = B4 = 0)</td>
</tr>
<tr>
<td>7 (B1 = K[r-1][c])</td>
</tr>
<tr>
<td>8 (B2 = ObjList[r].benefit + K[r-1][c-ObjList[r].weight])</td>
</tr>
<tr>
<td>9 IF ObjList[r].n1 (\in S[r-1][\cdot] - ObjList[r].n1.weight)</td>
</tr>
<tr>
<td>10 (B3 = ObjList[r].benefit + K[r-1][c-ObjList[r].n1.weight])</td>
</tr>
<tr>
<td>11 ENDF</td>
</tr>
<tr>
<td>12 IF ObjList[r].n2 (\in S[r-1][\cdot] - ObjList[r].n2.weight)</td>
</tr>
<tr>
<td>13 (B4 = ObjList[r].benefit + K[r-1][c-ObjList[r].n2.weight])</td>
</tr>
<tr>
<td>14 ENDF</td>
</tr>
<tr>
<td>15 (K[r][c] = Max(B1, B2, B3, B4))</td>
</tr>
<tr>
<td>16 ELSE</td>
</tr>
<tr>
<td>17 (K[r][c] = K[r-1][c])</td>
</tr>
<tr>
<td>18 ENDF</td>
</tr>
<tr>
<td>19 Store in (S[r][c]) the objects used to result in (K[r][c])</td>
</tr>
<tr>
<td>20 ENDF</td>
</tr>
<tr>
<td>21 END FOR</td>
</tr>
<tr>
<td>22 Return the objects used in (S[ObjList.size][W])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEBAP Algorithm.</td>
</tr>
<tr>
<td><strong>Algorithm 2: VEBAP algorithm</strong></td>
</tr>
<tr>
<td>Input: ATG and Blist (list of bins)</td>
</tr>
<tr>
<td>1 Convert each edge in ATG into a VBKP object</td>
</tr>
<tr>
<td>2 Put all VBKP objects into ObjList</td>
</tr>
<tr>
<td>3 FOR each bin (\in Blist)</td>
</tr>
<tr>
<td>4 VBKPList = VBKP(ObjList, size of bin, (i))</td>
</tr>
<tr>
<td>5 Assign the nodes contained in VBKPList in bin_i</td>
</tr>
<tr>
<td>6 Update the size of bin_i</td>
</tr>
<tr>
<td>7 ObjList(i) = ({})</td>
</tr>
<tr>
<td>8 END FOR</td>
</tr>
</tbody>
</table>
tasks that are not on the same bin (external load). Therefore, it is of paramount importance to address also the minimization of the external load. The minimization of the external load is captured by the TSA. Particularly, TSA takes as input the placement produced by Greedy or VEBAP and investigates whether the current placement can be further optimized as follows.

TSA proceeds by iterating all of the combinations between pairs of bins. For each pair, it evaluates whether a better placement can occur by swapping tasks between the pair of bins under consideration using the task swapping routine presented in Table 4. The task swapping routine works in the following manner. First, we calculate the network overhead, in terms of communication benefit, of the Greedy/VEBAP placement of the pair of bins being evaluated. Next, for each task contained in the bins under consideration, we calculate the benefit of moving it to the opposite bin. Afterwards, tasks are sorted in a list in decreasing order according to the ratio (benefit of the respective task)/(computing requirements of the respective task). TSA iterates all of the tasks contained in the aforementioned list and performs the following procedure for each of them. If the ratio of the task under consideration is negative, then TSA terminates. Otherwise, TSA considers whether the candidate task can be moved without violating the available capacity of the destination bin. In case of no violation, the respective task is moved to the destination bin. Otherwise, TSA attempts to find a task or a group of tasks hosted on the destination bin and swap it/them with the candidate task without violating the capacity of the bins under consideration. If such a task/group of tasks does exist with the overall benefit of movements being still positive, then the respective swap of tasks is performed.

6. Bin mapping algorithms

Bin mapping algorithms map the bins, containing the assigned tasks, onto the processing cores of NoC. Below we propose four algorithms for the bin mapping problem: (a) Extended Nearest Neighbor Mapping Algorithm (Ex-NN), (b) Single Cluster Mapping Algorithm (SC), (c) Multiple Cluster Mapping Algorithm (MC), and (d) Bin Swapping Algorithm (BS).

6.1. Extended nearest neighbor algorithm

In the nearest neighbor algorithm (NN), the first bin is chosen randomly and mapped on a random available core within the NoC. The next bin is mapped to the core which is available and has the minimal sum of Manhattan distance compared to the already mapped cores. The NN algorithm is proposed in [32], with the difference being that the authors in [32] consider tasks instead of bins. In the sequel, we propose an extended version of NN (Ex-NN). The main difference between NN and Ex-NN is that the latter takes into consideration the overall communication volume between bins, while the former does not. Particularly, the communication volume of a bin equals the total data exchanged between tasks contained in the respective bin and tasks not contained in it. Bins are sorted according to their communication volume in a descending order. The bin with the largest communication volume is placed at a chosen initial position. Iterating the sorted list, the remaining bins are mapped to cores in the same way as that of NN.

6.2. Single cluster algorithm

The single cluster algorithm (SC) places the bins onto the cores of the underlying NoC as follows. Initially, the bin with the largest communication volume is placed at the center of NoC. Next, the bins are placed around the cluster head according to the communication volume exchanged between the bin under consideration and the already mapped bins. The above is illustrated in Fig. 5 for a 3 × 3 Mesh NoC. As can be seen, the first bin (cluster head) is placed at the center of NoC. Next, the bin with the largest communication volume is mapped to the first available core having the minimum Manhattan distance from the center. To find an available location, we start from the center of NoC. If the core is not available, it proceeds with the left core, and then visits the cores in a spiral and clock-wise fashion until an available core is found.

6.3. Multiple cluster algorithm

The multiple cluster algorithm (MC) works in a similar way as SC, with the difference being that in MC there are more than one cluster heads. The size of a cluster in MC is predefined. The first cluster is placed at the top-left corner of the underlying NoC. The next cluster is placed at the right of the first cluster. The above procedure continues until we reach the right boundary of
in case of more than one beneficial swap, the swap with the biggest mean as swap that further reduces the current network overhead. In this case, a beneficial swap of bins between the cores within the NoC. Next, it investigates whether there are no beneficial swaps or unmarked cores. In case there are no unmarked cores, then all of the cores are unmarked and the whole procedure repeats itself. However, in case of no beneficial swaps the algorithm terminates.

7. Pareto optimization

In this section we present a Pareto-efficient algorithm (PEA) to optimize simultaneously the network overhead reduction and energy consumption. The algorithm starts from an initial solution (normally based on another algorithm) and then gradually builds a solution tree according to the aforementioned dimensions.

The algorithm gradually expands the solution tree in \( H \) iterations where \( H \) represents the height of solution tree. The root node of the tree is a symbolic node and does not represent any solution. An intermediate node represents a partial solution, while a leaf node represents a final solution. Each solution, either partial or final, has the associated energy consumption and network load incurred within the system by applying this solution. In the first iteration, the solutions belonging to the 2nd level of the solution tree are generated by evaluating each pair of cores. That is each intermediate solution in 2nd level includes two cores. Because the execution cost of exploring all of the solutions is prohibitive, in each iteration we only expand the \(<\text{best}>\) intermediate solutions by adding a new processing core to them. Solutions that belong to the Pareto frontier are considered \(<\text{best}>\) solutions. The process of evaluating the quality of intermediate/final solutions (to find \(<\text{best}>\) solutions) is discussed in the next paragraph in detail.

In second iteration, \(<\text{best}>\) solutions consisting of core pairs are expanded to generate triplets of cores. Consider the example of building the solution tree with six cores shown in Fig. 7. As can be seen in Fig. 7, each \(<\text{best}>\) solution in first iteration has two cores. In next iteration, four intermediate solutions are generated against each \(<\text{best}>\) solution by adding a different core to each solution, generating a triplet of cores. The procedure is repeated and there is no intermediate solution that can be expanded further. The process of expanding the solution tree is depicted in Fig. 7 where the best intermediate solutions are grayed. It can be seen that the nodes at the \( i \)th level have \( i \) number of cores. To generate an intermediate solution we do the following. The task migration algorithm attempts to migrate some or even all the tasks from high energy cores to low utilization low energy cores included in the corresponding intermediate solution. The network load and energy consumption for the resultant placement is recorded for each solution node.

The quality of each intermediate solution is evaluated on the basis of two optimization criteria, i.e., network overhead and energy consumption to find \(<\text{best}>\) solutions. Each intermediate solution represents a point in a two-dimensional space. Solutions belonging to the Pareto frontier are considered as \(<\text{best}>\) solutions because they cannot be defeated by any other solution in both dimensions. For instance, consider the diagram given in Fig. 8. Assume that the points A, B, C, and D represent four solutions in the two dimensions of network load and energy consumption. Points A, B, and D belong to Pareto frontier because they cannot be defeated by any other point in both dimensions. As can be seen, point A defeats point B considering energy consumption, while point B defeats point A in terms of network overhead. On the other extreme, point C does not belong to the Pareto frontier as it is dominated by point B in both dimensions. \(<\text{best}>\) intermediate solutions are the solutions belonging to the Pareto frontier.

Moreover, to optimize the execution time for large mesh sizes, each row of the mesh can be treated as a cluster representing the cores belonging to the respective row. The particular implementation that treats each row of the NoC as a cluster is referred to as clustering-based Pareto-efficient algorithm.
8. Complexity analysis

8.1. Time complexity

8.1.1. Analysis of proposed packing algorithms

The complexity VEBAP is calculated as follows. The VBKp routine called by VEBAP is $O(n \times W)$ where $n$ is the number of items to be packed, i.e., total number of edges and $W$ represents the capacity of bin. The above routine is called as many times as the number of bins (represented by $B$) and thus the complexity is given by Eq. (10).

$$O(B \times n \times W)$$

Similarly, complexity of task swapping algorithm is governed by the number of times the algorithm Task-Swapping is executed which is defined by Eq. (11).

$$O(B^2 \times O(Task\_Swapping))$$

The input to task swapping routine is two bins containing a certain number of tasks. Let $\tau$ represent the average number of tasks in each bin. After each iteration, the number of tasks to be evaluated is reduced by at least one. Therefore, the time complexity of task swapping routine is $O(\tau \times (1 + \tau)/2)$, which is equivalent to $O(\tau^2)$. Consequently, the time complexity of task swapping algorithm is given by Eq. (12).

$$O(B^2 \times \tau^2)$$

Complexity of Greedy algorithm depends on the algorithm sorting the edges in descending order according their communication volume.

8.1.2. Analysis of proposed mapping algorithms

Complexity of Ex-NN is similar to the time complexity of the algorithm used for sorting the bins according to their communication volume. Subsequently, mapping of bins onto the NoC can be achieved in constant time and grows linearly with the increase in size of NoC. Similarly, the time complexity of SC algorithm is similar to the Ex-NN algorithm with the exception that in SC we first find the bin having the highest communication ($MCBin$) and then sort the remaining bins according to the communication volume of remaining bins with the $MCBin$. The complexity of MC algorithm is similar to the SC with the addition that same process is repeated for $\mu$ times (with $\mu$ signifying the number of clusters). The time complexity of BS algorithm is given by Eq. (13) where $B$ represents the total number of bins or cores. Note that the time complexity of sorting the bins according to their benefit is $O(B \times \log(B))$ and each bin is evaluated at least once.

$$O(B^2 \times \log(B))$$

8.1.3. Analysis of pareto-efficient algorithm

The time complexity of Pareto-efficient algorithm is calculated as follows. The number of iterations of PEA is determined by the number of bins, number of intermediate solutions selected for expansion, and the height of solution tree. The number of bins is fixed but the number of intermediate solutions and height of the solution tree depend on multiple factors, such as task packing, initial mapping of bins, and communication overhead among interdependent tasks. Therefore, it is difficult to estimate the exact time complexity of PEA. Although the worst case complexity of PEA is exponential, the average case complexity can be significantly lower. The estimated average complexity of PEA is given by Eq. (14) where $B$ represents the number of bins, $H$ is the height of solution tree, and $S$ represents the average number of intermediate solutions selected for expansion at each iteration.

$$O(B^2 \times H \times S)$$

8.2. Space complexity

8.2.1. Space complexity of packing algorithms

The space complexity of VEBAP is given by Eq. (15).

$$O(B + n + (n \times W))$$
Here, $B$ is the total number of bins, $n$ represents the number of items to be packed, i.e., total number of edges, and $W$ defines the average capacity of bins. The term $n \times W$ corresponds to the two matrices employed by VEBAP.

Similarly, the space required by task swapping and greedy algorithms is defined by Eqs. (16) and (17) respectively.

\[
O(B + k) \quad (16)
\]

\[
O(B + n) \quad (17)
\]

Note that $k$ defines the number of tasks packed in the two bins that are processed during any given iteration.

### 8.2.2. Space complexity of mapping algorithms

Space complexity of all mapping algorithms is expressed by Eq. (18).

\[
O(B^2 + (r \times c)) \quad (18)
\]

Here, $r$ and $c$ represent the number of rows and columns of NoC, respectively. Note that a matrix of size $B^2$ is used to store the communication volume of each bin with the rest bins.

### 8.2.3. Analysis of pareto-efficient algorithm

The space complexity of Pareto-efficient algorithm is calculated as follows. At each level the algorithm examines each solution, but only the solutions that belong to Pareto frontier are chosen and stored for expansion in next iteration. Let $b$ be the average branching factor of solution trees. Consequently, Pareto-efficient algorithm stores, on average, $b$ number of solutions in next iteration. Therefore, in worst-case scenario the amount of memory required to explore the solution tree of height $H$ will be $O(b^H)$. Moreover, each solution needs to store the data for $k$ number of bins. Therefore, the space complexity is given by Eq. (19).

\[
O(b^H \times k) \quad (19)
\]

It is evident from Eq. (19) that space complexity of Pareto-efficient algorithm largely depends on the height of solution tree.

### 9. Experimental evaluation

#### 9.1. Experimental setup

To evaluate the performance of the proposed algorithms, we have conducted a series of experiments. We have developed a customized simulation framework in Java using Eclipse IDE. Numerous scenarios are generated by varying the number of tasks, communication requirements, and number and size of bins (i.e., cores). The simulations have been conducted on NoC sizes ranging from $5 \times 5$ to $50 \times 50$. The number of tasks was proportional to the number of cores, with it varying from 300 to 10,000. The details related to number of tasks against each simulation scenario are presented in Table 5. In application task graphs, computational requirements of each task are uniformly distributed between 1 to 20 million instructions. Edges’ weight in application task graphs are uniformly distributed between 1 and 10. Moreover, the number of edges in the task graph ranges between 1.2 to 1.5 times number of tasks. Computational capacity of bins (cores) range from 50 to 200 based on the NoC size and number of tasks. Power consumption parameters, $P_{\text{max}}$ and $P_{\text{idle}}$ (used in Eq. (4)) are set to 55 and 36 mW, respectively (for further details see [24]).

From the above, we calculate that $P_{\text{util}} = 19$, recall that $P_{\text{util}}$ is equal to $P_{\text{max}} - P_{\text{idle}}$. Moreover, we assume a homogeneous NoC architecture where the energy to traverse any single hop (router) and link is considered same. For NoC energy consumption, value of both parameters $ER_{\text{bit}}$ and $EL_{\text{bit}}$ is set to 0.01 nJ, such that power consumption on routers and links is comparable [23]. Whereas, $EC_{\text{bit}}$ is set to 0.0001 nJ. For each simulation scenario, four different sets of workflows are generated and results are averaged over these four workflows. To show the statistical information in the results, standard error bars are included in most plots. The formula to calculate standard error of the mean is given by Eq. (20).

\[
SE = \frac{\sigma}{\sqrt{n}} \quad (20)
\]

Here $SE_i$ represents the standard error of mean, $\sigma$ standard deviation of mean, and $n$ is number of observations. Details of different simulation parameters are presented in Table 6.

#### 9.2. State-of-the-art algorithms

The First Fit (FF) and Nearest Neighbor (NN) algorithms, proposed in [32], served as the yardstick for the bin mapping algorithms. FF is a simple algorithm that maps the bins in sequential order row-wise. First the bins are sequentially placed on the first row then second row and so on. On the other hand, the Simulated Annealing (Sim-A) algorithm served as the yardstick for the task mapping algorithms. To quantify the performance of the proposed algorithms we have compared the proposed algorithms with the following three state-of-the-art algorithms: (a) Simulated Annealing (Sim-A) based task mapping heuristic proposed in [17], (b) CastNet (CNet) an energy-aware application mapping heuristic proposed in [30], and (c) MPMap task and core mapping heuristic proposed in [19].

The Sim-A based algorithm [17] works initially by calculating a baseline mapping for each task rather than a random placement. In the baseline mapping, the algorithm attempts to assign each task to the preferred processing element that minimizes the energy consumption respecting the capacity of PEs. In case the preferred processor does not have the capacity to hold the task, then the task is assigned to the nearest available processing element that minimizes the network overhead among the interdependent tasks. The remaining tasks are mapped to the processing element minimizing the required resources and minimizing the energy consumption while considering the network overhead between them and the tasks that have already been mapped. After performing the baseline mapping, simulated annealing algorithm is applied. In each

---

**Table 5**

<table>
<thead>
<tr>
<th>NoC size</th>
<th>Number of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 5</td>
<td>300</td>
</tr>
<tr>
<td>6 x 6</td>
<td>400</td>
</tr>
<tr>
<td>7 x 7</td>
<td>500</td>
</tr>
<tr>
<td>8 x 8</td>
<td>600</td>
</tr>
<tr>
<td>9 x 9</td>
<td>800</td>
</tr>
<tr>
<td>10 x 10</td>
<td>1000</td>
</tr>
<tr>
<td>15 x 15</td>
<td>3000</td>
</tr>
<tr>
<td>20 x 20</td>
<td>4000</td>
</tr>
<tr>
<td>25 x 25</td>
<td>5000</td>
</tr>
<tr>
<td>30 x 30</td>
<td>7000</td>
</tr>
<tr>
<td>50 x 50</td>
<td>10000</td>
</tr>
</tbody>
</table>

**Table 6**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>[300, 10000]</td>
</tr>
<tr>
<td>Number of bins</td>
<td>[25, 2500]</td>
</tr>
<tr>
<td>Computational requirements</td>
<td>[1, 20]</td>
</tr>
<tr>
<td>Computational capacity</td>
<td>[50, 200]</td>
</tr>
<tr>
<td>Communication requirements</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>Number of edges</td>
<td>[1, 2, 1, 5] x number of tasks</td>
</tr>
</tbody>
</table>
iteration of the Sim–A, a task is randomly chosen from a processor and re-mapped to another randomly chosen processor. After remapping a task, energy consumption of the new mapping \((E')\) is calculated and compared with the energy consumption of the old mapping \((E)\). The decision for acceptance or rejection of the new mapping is made on the basis of probability function \(P(E, E', T)\) defined in [17]. If \((E')\) is less than \(E\) then the mapping is always accepted and applied. Otherwise, it is still possible that the new mapping accepted according to the \(P(E, E', T)\). The probability based acceptance function is necessary for Sim–A to avoid getting trapped to local optimum solutions. If the probability of acceptance is greater than a randomly calculated value, then the new mapping is accepted. Otherwise, it is rejected and the next mapping is calculated. The above process is repeated for a number of iterations chosen by the user.

MPMAP is a joint task and core mapping heuristic that attempts to optimize computational and communication energy consumption [19]. MPMAP works in two phases; an initial task mapping phase and an iterative remapping phase, each followed by core mapping. In the initial mapping phase, the algorithm attempts to map as many tasks as permitted on the lowest energy consumption cores depending on capacity constraints. In iterative re-mapping phase, the processing cores are sorted in a descending order based on utilization. Afterwards, each core is evaluated according to the aforementioned sorted order, with all of the tasks placed on each core being evaluated for migration to other suitable cores. The tasks are migrated to the processing core that minimizes the overall energy consumption. After the initial task mapping phase and each iteration of remapping algorithm, the core mapping heuristic is invoked. In core mapping, processing cores are mapped onto tiles of NoC considering the communication volume among the tasks.

The CastNet [30] task mapping heuristic works in the following manner. The CastNet algorithm starts by identifying the highest priority task which is the task having the highest ratio of incoming and outgoing communication traffic. After that the algorithm finds a prefixed number of initial candidate core positions within the NoC. For each initial candidate core position, the algorithm repeats the following steps. The highest priority task is placed at the initial core position. Later, a task \(t_s\) is selected among the unmapped tasks that has the highest communication volume with the already mapped tasks. Next, \(t_s\) is placed at the core that minimizes the communication energy and network overhead between \(t_s\) and already mapped tasks. A complete mapping is achieved for each initial candidate core. For each complete mapping, the algorithm records the total energy consumption. Finally, the mapping resulting in the lowest total energy consumption is applied.

9.3. Results and discussion

In this section, benefits of the proposed approach are quantified and experimental results are discussed to evaluate the quality of proposed algorithms. The performance of the algorithms is evaluated on the basis of two metrics: (a) network load and (b) energy consumption. In the subsequent paragraphs, Greedy and VBKP refer to Greedy and Knapsack based VEBAP algorithms, respectively. Similarly, Greedy-Swap and VBKP-Swap represent the results of TSA applied on top of Greedy and VEBAP algorithms, respectively. The results are grouped in two categories based on synthetic and real world workloads.

9.3.1. Synthetic workloads

Figs. 9 and 10 show the comparison of average energy consumption and network load for different combinations of each algorithm, respectively. The results have been calculated by taking the average of all the results of a given algorithm for all NoC sizes, i.e., from 5 × 5 to 50 × 50 mesh networks. It must be noted that the
simulated annealing (Sim-A), CastNet (CNet), and MPMAP heuristics were not run on top of other bin packing algorithms but are run independently. The experimental results indicated that VBKP-Swap achieved the lowest energy consumption on average when applied on top of all of the mapping algorithms. More specifically, VBKP-Swap achieved up to 55% energy savings with an average of 48% as compared to the Greedy algorithm. On the other extreme, VBKP achieved the second best results with marginally higher energy consumption when applied on top of all of the mapping algorithms, except for the case of BS where VBKP slightly outperformed VBKP-Swap regarding energy consumption. It is interesting to note that VBKP reduced the energy consumption up to 54% with an average of 46% as compared to the Greedy algorithm.

Similarly, VBKP-Swap achieved significantly lower network load in all cases as compared to the Greedy and Greedy-Swap algorithms. On the other hand, VBKP had slightly higher network load than VBKP-Swap. More specifically, VBKP and VBKP-Swap achieved up to 57% and 52% reduction in network load as compared to the Greedy algorithm, respectively. This is mainly because VBKP and VBKP-Swap attempt to jointly optimize the resource utilization and communication overhead resulting in a placement with lower energy consumption and reduced network load.

We also observed that PEA when applied on top of VBKP algorithm demonstrated the lowest energy consumption and network load. Specifically, it achieved 50% and 57% reduction in energy and network load, respectively, as compared to Greedy algorithm. The reason that when applying PEA on top of VBKP leads to better results against applying it on top of VBKP-Swap is the following. First of all, recall (as shown in Figs. 9 and 10) that VBKP-Swap is superior to VBKP in terms of both energy consumption and network load reductions. Recall also that PEA builds its solution tree based on some initial solution. Therefore, if the initial solution is tight in terms of both energy consumption and network load (as is the case with VBKP-Swap) the solution search space becomes small leaving less room for further improvements.

We also conducted a series of experiments to analyze the trade-off between the performance, in terms of energy and network load, and the execution times of clustering and non-clustering based Pareto-efficient algorithms. Figs. 11 and 12 present the comparison between energy consumption and network load of clustering and non-clustering implementation of PEA, respectively. For example, “Greedy-Clust” and “Greedy” refer to the results of clustering and non-clustering PEA applied on top of the Greedy algorithm, respectively. The results have been normalized according to Greedy-Clust. Moreover, results have been calculated by taking the average of all the results of a given bin packing algorithm for all mesh sizes and mapping algorithms. It was observed that energy consumption and network load for the clustering-based case is comparatively higher than the non-clustering case against all of the bin packing algorithms. However, the difference in execution times between clustering and non-clustering algorithms was huge as depicted in Fig. 13. More specifically, energy consumption and network load
of the clustering-based algorithm ranged from 4% to 7% and 10% to 15% higher than the non-clustering based implementation, respectively. However, it can be seen from Fig. 13 that the execution time of the clustering-based PEA is significantly less than the non-clustering one, especially for the case of VBKP-Swap where the clustering algorithm is three orders of magnitude better than the non-clustering one. Moreover, we can see that among the results of clustering based algorithms, the execution time of Greedy-Swap and VBKP-Swap are comparatively low. The reason behind this is that PEA did not achieve significant improvements in energy consumption and network load when running on top of the above algorithms as compared to the case of Greedy and VBKP algorithms. Therefore, the solution tree for the PEA when running on top of swapping algorithms did not expand to as many levels as that of running on top of non-swapping algorithms.

Fig. 14 shows the comparison of execution time of bin packing algorithms averaged over all mesh sizes. The execution time of MPMAP, CNet, and Sim-A are significantly higher than all the proposed algorithms. The reason is that MPMAP evaluates each task for migration against each available PE to reduce the energy consumption. Evaluating each task against each PE, iteratively, leads to a significantly increased run time. Similarly, CNet calculates a complete mapping solution for each candidate core and then selects the best solution among the available solutions. The number of candidate cores and solutions increase considerably with the increase in the size of NoC, leading to a substantial increase in algorithm’s execution time. We also performed analysis of the execution time of bin mapping algorithms depicted in Fig. 15. Among the bin mapping algorithms, execution times of BS and MC were significantly higher than the remaining mapping algorithms. The above is justified by the fact that BS and MC performed additional processing to consider the dependencies among the communicating tasks. However, it must be noted that the execution time of mapping algorithms is very low as compared to the bin packing algorithms.

Next, we explored the behavior of proposed bin packing and placement algorithms over varying mesh sizes. Therefore, we drew a parallel among FF, Sim-A, PEA, CNet, MPMAP, as well as the mapping algorithms BS and MC. It must be noted that BS and MC were chosen to be included in the comparison because they demonstrated the best performance among all of the mapping algorithms presented in Section 6. The Pareto-efficient, BS, MC, and FF algorithms were run on top of the Greedy-Swap (Figs. 16 and 17) and VBKP (Figs. 18 and 19) algorithms.

It can be seen in Fig. 16 that MPMAP outperformed all of the algorithms in terms of energy consumption in almost all cases except for the case of $7 \times 7$ and $30 \times 30$ mesh where MPMAP had slightly higher energy consumption than CNet. The next best results were achieved by CNet with Sim-A following closely. In case of network load, BS exhibited the best performance. On the other extreme, BS achieved the lowest network overhead for all mesh sizes. Similarly, MC exhibited the next best results on average, with CNet following closely. Sim-A and MPMAP exhibited irregular performance in terms of network load. Specifically, MPMAP’s network load increases significantly as the size of mesh NoC increases. The reason behind this is that MPMAP focused on
reducing the computational energy consumption without taking explicitly into consideration the communication overhead among the interdependent tasks during the initial task mapping phase. It is also reported that Sim-A resulted in the second highest network load, after MPMAP, for large mesh sizes, i.e., $15 \times 15$ onwards as shown in Fig. 17. The reason that Sim-A exhibits an increased network load is that when Sim-A performs task swaps, it considers only the energy consumption. Therefore, a swap that results in reduced energy consumption may lead to an increase in network load. The BS, MC, and CNet algorithms achieved lower network load compared to MPMAP, Sim-A, and PEA. The above is because BS, MC, and CNet explicitly attempt to reduce the communication overhead by placing heavily communicating tasks or bins in close proximity. The experimental results also reveal that PEA, when run on top of Greedy-Swap, exhibited comparatively higher energy consumption than MPMAP, CNet, and Sim-A for the majority of NoC sizes. However, PEA achieved lower network load compared to MPMAP and Sim-A for the majority of mesh sizes (particularly for large mesh NoCs). Moreover, network load of PEA reaches close to that of MC and CNet. Specifically, PEA achieved lower network load than CNet for larger mesh NoCs, i.e., $15 \times 15$ onward except in the case of $30 \times 30$ mesh.

Figs. 18 and 19 depict the comparison of energy consumption and network load over varying mesh sizes for VBKP algorithm, respectively. PEA achieved the best results in terms of energy consumption and network load. Similarly, BS and MC had the
second and third best performance, respectively, in terms of energy consumption and network load. The results of Pareto-efficient, BS, and MC algorithms remained consistent over all of the mesh sizes except for 50 × 50 mesh. Specifically, in case of 50 × 50 mesh, the energy consumption and network load of BS is comparatively higher than MC and quite close to CNet.

It is noteworthy to point out that energy consumption and network load for CNet decreases gradually and becomes closer to proposed algorithms when increasing the size of NoC. Particularly, from 20 × 20 mesh onward the energy consumption and network load of CNet is quite close to MC and BS, but still significantly higher than PEA. The reason behind the decrease in energy and network load is that CNet has more options for initial candidate cores when increasing mesh size, resulting in that way in an increased number of tentative solutions. However, an increase in the number of tentative solutions is translated to an increase in execution time as illustrated in Fig. 14.

On average, BS achieved the lowest network load, as compared to the rest algorithms. More specifically, BS reduced the network load by 30% and 40% on average as compared to FF, when the former was applied on top of Greedy-Swap and VBKP, respectively. Whereas, MC achieved, on average, the second lowest network load. A more general conclusion can be drawn from the above comparison of Greedy-Swap and VBKP. That is, the network load and energy consumption of all the proposed mapping algorithms tend to increase for higher mesh sizes with Greedy-Swap algorithm. Whereas, in case of VBKP, the energy consumption and network load of proposed mapping algorithms remains relatively consistent.

Similar experiments were also conducted for the aforementioned algorithms applied on top of Greedy and VBKP-Swap. However, figures for Greedy and VBKP-Swap were not included because they exhibited similar trends, with the BS and MC having the best performance against the rest of the task mapping algorithms. Moreover, Sim-A achieved the lowest energy consumption and network load as compared to the rest of the mapping algorithms on top of Greedy. On the other hand, in case of VBKP-Swap, Pareto-efficient outran all of the algorithms, with BS and MC resulting in the second best results. Furthermore, examples of Pareto frontier are depicted in Figs. 20 and 21 to illustrate the tradeoff between energy consumption and network load. Figs. 20 and 21 show the results of PEA applied on top of greedy algorithm. It is worth mentioning that due to space constraints, it is not feasible to include the Pareto frontier results for all simulation cases because PEA is applied on top of each of the task packing algorithm and simulation scenario against all mesh sizes ranging from 5 × 5 to 50 × 50 NoCs.

9.3.2 Real world workflows
We have also conducted experiments with task graphs that are based on real world scientific application workflow traces. These workflows include: (a) CyberShake, (b) LIGO, (c) Montage, and (d) SIPHT [21]. Five different workflow traces are employed to generate task graphs (each graph having 100 tasks on average) for each of the above four applications. Experiments for the aforementioned task graphs have been conducted over 5 × 5 mesh NoC.

The experimental results are presented in Figs. 22–29. These results clearly show that the overall trend for both energy consumption and network load is similar with the results of synthetic task graphs presented in Fig. 9 and Fig. 10, respectively. However, it is interesting to note that VBKP and VBKP-Swap algorithm perform better over CyberShake and Montage compared to LIGO and SIPHT. For instance, VBKP-Swap achieved on average 51% and 48% lower energy consumption compared to Greedy when run over CyberShake and Montage, respectively. Whereas, VBKP-Swap exhibited on average 28% and 17% lower energy compared to Greedy when executed over LIGO and SIPHT task graphs, respectively. In general, similar trend is observed with VBKP algorithm for both energy consumption and network load. The reason behind such behavior is that CyberShake and Montage are more computationally and communication intensive against LIGO and SIPHT. The above indicates that VBKP and VBKP-Swap algorithms show superior performance with workloads having higher computational and communication requirements. Even though the performance gains of VBKP and VBKP-Swap decrease when computational and communication requirements are low, they still outperform the rest algorithms in terms of both energy consumption and network load.

Keeping in view the aforementioned results, we can safely conclude that the proposed VBKP algorithm coupled with PEA achieved the best results both in terms of energy consumption and network load. Moreover, BS and MC achieved a performance close to that of PEA when the latter is applied in conjunction with VBKP.

10. Conclusions
This paper studied the problem of task and core mapping to minimize energy consumption and communication overhead for NoC based MPSoC platforms. We presented a novel Knapsack based bin packing algorithm (VBKP) as well as a task swapping algorithm to jointly optimize the energy consumption and network load. Moreover, we presented several bin packing algorithms that were run on top of the bin packing algorithms. In addition, we presented a Pareto-efficient algorithm exploring the solution space in two dimensions (i.e., energy consumption and network load). Extensive simulations were conducted with both synthetic and real world scientific application graphs to evaluate the performance of proposed algorithms. The experimental results indicated that the proposed VBKP algorithm in conjunction with PEA can effectively reduce the energy consumption and network load. Moreover, the proposed clustering-based implementation of PEA can significantly reduce the execution time, making it suitable for exploring large-scale problems.
Fig. 22. Energy consumption over CyberShake.

Fig. 23. Network load over CyberShake.

Fig. 24. Energy consumption over Montage.

Fig. 25. Network load over Montage.
Fig. 26. Energy consumption over LIGO.

Fig. 27. Network load over LIGO.

Fig. 28. Energy consumption over SIPHT.

Fig. 29. Network load over SIPHT.
Below we list some future extensions to the proposed work. First, the congestion in the links of the underlying network is an interesting problem to be studied. Second, how the proposed approach can be extended to support other architectures, such as fat tree, torus, and butterfly could be explored. Third, analyzing the impact of proposed technique regarding the lifetime and reliability of the MPSoC architecture is also an attractive area to explore.

Acknowledgments

Samee U. Khan’s work was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Nikos Tziritas’s work was supported by NSFC and PIFI International Scholarship under the grants 61550110250 and 2017VCT0001, respectively.

References


[27] Tahir Maqsood received his M.S. in Computer Networks from Northumbria University, UK, in 2007. He is currently a Ph.D. candidate in the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan. His research interests include application mapping, energy efficient systems, and network performance evaluation.

[28] Nikos Tziritas received his B.Sc. degree from the Technological Educational Institute of Serres, Greece, in 2004, and a M.Sc. and a Ph.D. degree from the University of Thessaly, Greece, in 2006 and 2011, respectively. He is currently affiliated with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research has been on scheduling, load-balancing and replication in CDN, as well as energy optimization and resource management in WSNs and cloud computing systems. He is the recipient of the Award for Excellence for Early Career Researchers in Scalable Computing from IEEE Technical Committee in Scalable Computing.
Sajjad A. Madani works at COMSATS Institute of Information technology as Associate Professor. He did M.S. in Computer Sciences from Lahore University of Management Sciences and Ph.D. from Vienna University of Technology. His areas of interest include low power wireless sensor network and green computing. He has published more than 40 papers in peer reviewed international conferences and journals.

Samee U. Khan is Associate Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan’s research interests include optimization, robustness, and security of: cloud, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in over 300 publications. He is a Fellow of the Institution of Engineering and Technology (IET, formerly IEE), and a Fellow of the British Computer Society (BCS). He is an ACM Distinguished Lecturer, a member of the ACM, and a Senior Member of the IEEE.

Cheng-Zhong Xu received the Ph.D. degree in computer science from the University of Hong Kong in 1993. He is currently a professor in the Department of Electrical and Computer Engineering of Wayne State University, and the director of Cloud and Internet Computing Laboratory (CIC) and Sun’s Center of Excellence in Open Source Computing and Applications (OSCA). His research interest is mainly in scalable distributed and parallel systems and wireless embedded computing devices. He has published two books and more than 160 articles in peer-reviewed journals and conferences in these areas. He is a senior member of the IEEE.

Albert Y. Zomaya is the Chair Professor of High Performance Computing & Networking in School of Information Technologies, University of Sydney. Prof. Zomaya is the author/co-author of seven books, more than 400 papers, and the editor of 12 books and 15 conference proceedings. He was the Editor in Chief of the IEEE Transactions on Computers and servers as an associate editor for 19 leading journals. He is a Fellow of the IET and AAAS.