Abstract—Energy Management for multi-mode Software Defined Radio (SDR) systems remains a daunting challenge. In this paper, we focus on the issue of task allocation for multi-processor based systems with hybrid processing resources that can be reconfigured to support various operations. With the objective of minimizing energy, we present an analytical probabilistic model that considers static, dynamic, configuration and communication energy components for multiple applications characterized by certain probabilities of execution. Furthermore, we propose a fast, energy aware static task mapping heuristic to minimize the average overall energy consumption. In addition, the heuristic proposes re-ordering for the processing units for further energy reduction. Simulation results show that the proposed mapping heuristic is capable of achieving results that are within 20% of the optimal solution while providing orders of magnitude speedup in processing time. Further saving is achievable via joint mapping and placement.

I. INTRODUCTION

With the current advances in wireless communication and the limitations of spectrum access due to economic and/or governmental policies, future wireless networks are steadily progressing to a co-operative model aiming to provide universal coverage that implicitly encapsulates nodes that can support multiple Radio Access Technologies (RATs). As an example, consider the IEEE 802.21 standard which provides a framework to support seamless mobility through networks based on different RATs without the need to restart the radio connection every time the mobile moves to a new network [1]. Another relevant standard, IEEE P1900.4, defines building blocks for enabling coordinated network-device distributed decision making which will aid in the optimization of radio resource usage, including spectrum access control, in heterogeneous wireless access networks [2]. To fully benefit from such emerging network concepts, there is a need for efficient design time and run time reconfigurable platforms. Numerous reconfigurable architectures have been proposed spanning different technologies including application specific instruction set processors (ASIPs), field programmable gate arrays (FPGAs), and digital signal processors (DSPs). Recently, multi-processor systems on chip (MPSoC) architectures have evolved rapidly in the race of high performance embedded computing [3], especially in applications that require a flexible computing structure that can be reconfigured to handle various applications. A common design metric among all platforms is reducing energy consumption that restricts both the capabilities of the device and the design choices that are available. Towards that end, numerous techniques have been developed to optimize energy consumption at different levels including algorithm, system, architecture, and circuit levels.

A main factor that impacts energy consumption in MPSoC based architectures is task allocation [4]. Therefore, several energy-aware task allocation techniques have been proposed. In [5], a survey is presented that discusses energy-efficient scheduling for real-time systems supporting dynamic voltage scaling (DVS) platforms, including uniprocessor and multiprocessor systems. In [6], an energy aware scheduling technique is presented for multiple real time tasks on homogeneous MPSoCs considering DVS. The scheduling is performed based on the probabilistic distribution of task execution times. In [7], the authors proposed an algorithm to minimize the energy consumption of streaming applications on homogeneous chip multiprocessors while satisfying the throughput and response time requirements. [8] represents an energy aware task allocation framework to minimize the dynamic energy consumption on heterogeneous multiprocessors considering DVS. In [9], partitioning of real time tasks on heterogeneous multiprocessors is considered to minimize the energy consumption. The authors in [10] considered the effect of both processing and communication energy for tasks on heterogeneous MPSoCs. They proposed a simulated annealing and timing adjustment heuristic to accelerate the optimization process. However, their model does not consider probabilistic applications. In [11], high level application and hardware models are presented, as well as power aware static and dynamic mapping algorithms of tasks to MPSoC architectures, where mapping is based on static and dynamic power consumption. However, this work does not factor in energy components due to communications and reconfiguration. In fact, the energy due to communication among processing units (PUs) can have a significant contribution to the overall energy consumption, as illustrated in [12]. Furthermore, for multi-mode systems, it is important to consider the reconfiguration energy required to switch PUs between different tasks. In case of reconfigurable fabric based PUs (e.g. FPGA fabric), there is a reconfiguration
energy cost associated with the change of configuration bits of configurable logic blocks (CLBs), and connections represented in the switching matrix [13], while in the case of multi-processor based systems, there is an associated cost in switching context and reloading programs from internal or external memories. In this paper, in addition to incorporating static and dynamic power as proposed in [11], we consider both communication and reconfiguration energy components using an analytic probabilistic model. We then introduce an energy-aware static task mapping heuristic based on the extended model. 

The paper is organized as follows: Section II presents the extended system model for both application and hardware architecture. In section III, the problem formulation is presented. The proposed heuristic solution for the static mapping and placement is presented in section IV followed by the performance results in section V. Section VI concludes the paper.

II. SYSTEM MODEL

We assume a generic platform consisting of a heterogeneous MPSoC architecture with different types of PUs. For the sake of generality, we impose no restrictions on the types of PUs, for example, a PU can be a general purpose processor (GPP) unit that runs software instructions of a task, a DSP unit, an ASIP unit, or a reconfigurable fabric that runs a task in hardware. Each PU can run more than one task simultaneously based on its computational capability and the computational requirements of these tasks. The PUs are communicating through available means of on chip communications. Communication between two tasks is performed through a number of data transactions taking place between the corresponding PUs. When a certain application comprising a set of tasks starts to run on the platform, the tasks are mapped to different PUs based on a mapping procedure, and the selected PUs are configured to perform the corresponding tasks. Figure 1 illustrates the idea of mapping an application to a heterogeneous MPSoC architecture.

![Fig. 1. Application to architecture mapping](image)

In the next subsections, we represent the analytical models for the application and architecture similar to those originally proposed in [11]. The proposed models are augmented to further include communication and reconfiguration energy. To maintain consistency, we adopt the same parameter notations as [11].

A. Application model

In general, we assume that the system supports a different set of applications \( A = \{A_1, A_2, A_3, \ldots, A_n\} \), each application \( A_i \) supports different modes \( \mu_i = \{\mu_{i,1}, \mu_{i,2}, \mu_{i,3}, \ldots, \mu_{i,n}\} \). Different modes for different applications construct different scenarios, and one scenario \( S_m \in S \) is executed at a time, where \( S \) is the set of all possible scenarios. Each application scenario \( S_m \) is modeled as a set of tasks connected together and represented through the directed task graph \( G_m = (\varepsilon_m, V_m) \). This task graph has a set of nodes \( V_m \) representing tasks and a set of edges \( \varepsilon_m \) representing communication between tasks as shown in the task graph in Figure 1.

Each scenario has an execution probability \( \chi_m \) and a set of tasks \( T_m \subset T \), where \( T \) is the total set of tasks and \( T_m \) is the set of tasks executed in scenario \( S_m \). The variable \( \theta_{i,m} \) indicates the mapping between \( t_i \) and \( S_m \). \( \theta_{i,m} = 1 \) when \( t_i \in T_m \) and 0 otherwise. For each task \( t_i \in T_m \), there is a computational requirements denoted by \( \gamma_i \), and we assume that each scenario is executed for a known average time \( \tau_m \). Communication between the pair of tasks \( t_i \) and \( t_j \) in \( S_m \) is defined by the variable \( \varepsilon_{i,j,m} \) where \( \varepsilon_{i,j,m} = 1 \) if \( t_i \) communicates to \( t_j \) and 0 otherwise. The number of transactions between two communicating tasks \( t_i \) and \( t_j \) in \( S_m \) is defined by \( N_{\text{trans},i,m} \). For a certain task \( t_i \), the task probability \( \psi_i \) and average execution time \( \tau_i \) are calculated from the scenario probabilities and average scenario execution time as presented in (1) and (2).

\[
\psi_i = \sum_{S_m \in S} \theta_{i,m} \chi_m
\]

\[
\tau_i = \sum_{S_m \in S} \theta_{i,m} \chi_m \tau_m
\]

B. Architecture model

As mentioned previously, we assume an MPSoC based platform constructed out of different PU types. Each PU type has computational resources denoted as \( \lambda_j \) for \( j^{th} \) PU type. Each PU \( P_j \) has a number of instances. The maximum possible number of instances of each PU type is the number of tasks. This happens when all the tasks are mapped to the same PU type, and each instance hosts at most one task. The actual number of instances for each PU type is determined during the design space exploration phase.

In general, for heterogeneous PUs, each PU type supports different functional units (FUs), so some PUs may not be feasible for certain tasks. i.e., one task may require floating point operations which are not supported by certain PUs. Therefore, a feasibility indicator \( f_{i,j} \) is defined such that \( f_{i,j} = 1 \) if task \( t_i \) can run on any instance of \( P_j \) and 0
otherwise. The communication between these PUs can be bus based or network on chip (NoC)-based. For any instance of the \( j^{th} \) PU type, the static power is denoted as \( \sigma_j \) which is independent from the task running over \( \hat{p}_{j,k} \), where \( \hat{p}_{j,k} \) is the \( k^{th} \) instance of the \( j^{th} \) PU type. In addition, each \( \hat{p}_{j,k} \) has a dynamic power denoted as \( \delta_j \), and is highly dependent on the task utilization, so it is multiplied by the task utilization. Moreover, we assume that there is a reconfiguration cost associated with each task upon mapping to a certain instance \( \hat{p}_{j,k} \). In general, we refer to the reconfiguration power of reconfiguring \( i^{th} \) task on \( j^{th} \) PU type as \( n_{i,j} \) and the corresponding reconfiguration time with \( \tau_{r_{i,j}} \).

The communication between two tasks \( t_i \) and \( t_l \) mapped on \( \hat{p}_{j,k} \) and \( \hat{p}_{q,v} \) instances of any PU type is associated with a communication energy cost. This energy is based on the topology and represented as \( E_{\tau j,q,v} \). In our case, we assume the topology presented in Figure 1, which is a 2D grid connecting different PUs. The model is developed such that each instance \( \hat{p}_{j,k} \) is placed in a certain location defined by a column number \( C_{j,k} \) and a row number \( R_{j,k} \). Moreover, we assume that \( E_{\text{seg}} \) is the energy per each segment per transaction, where the segment is the connection between two successive instances in a row or a column. In general, the minimum path length between \( \hat{p}_{j,k} \) and \( \hat{p}_{q,v} \) instances is \( |C_{j,k} - C_{q,v}| + |R_{j,k} - R_{q,v}| \), and the minimum communication energy consumption is as shown in (3). Without loss of generality, the communication energy cost between any two tasks mapped onto any two instances for any given architecture will replace (3). The binary variables \( E_{\tau j,k} \) and \( L_{j,k} \) represent the placement of \( \hat{p}_{j,k} \) to the \( c^{th} \) column and \( r^{th} \) row respectively such that \( E_{\tau j,k} = 1 \) if \( \hat{p}_{j,k} \) is placed in the \( c^{th} \) \( (r^{th}) \) column \( (row) \) and 0 otherwise.

\[
N_{\text{trans}_{i,m}} = E_{\text{seg}} \cdot (|C_{j,k} - C_{q,v}| + |R_{j,k} - R_{q,v}|) \quad (3)
\]

### III. Problem Formulation

In this paper, we consider two problems: the task allocation only problem, where the PUs have fixed placement, and the joint task allocation and PU placement where re-ordering the PUs at design time is available.

For the given set of scenarios \( S \), the target is to find the mapping between different tasks and different PU instances to minimize the overall energy consumption including static, dynamic, communication and reconfiguration energy. There are two types of task mapping; namely, static mapping at design time, and dynamic mapping at run time. This paper focuses on the first type which is explored offline. The binding process is accompanied with determination of the number of instances of each PU type. A task \( t_i \) is related to the PU \( \hat{p}_{j,k} \) through task utilization \( u_{i,j} = \frac{\lambda_i}{X} \) which defines the portion of PU resources utilized by \( t_i \). In general, for the mapping of \( t_i \) on \( \hat{p}_{j,k} \) to be feasible, \( u_{i,j} \leq 1 \). The variable \( M_{i,j,k} \) defines the mapping between \( t_i \) and \( \hat{p}_{j,k} \) such that \( M_{i,j,k} = 1 \) if \( t_i \) is mapped on \( \hat{p}_{j,k} \), and 0 otherwise. In addition, the variable \( Z_{m,j,k} \) defines the mapping between scenarios and PU instances such that \( Z_{m,j,k} = 1 \) when any \( t_i \in T_m \) is mapped to \( \hat{p}_{j,k} \) and 0 otherwise. To formulate a tractable problem, we assume the following:

1) The scenarios to be run on the architecture are known a priori.
2) Each task is mapped completely on one and only one PU instance. However, each instance can host more than one task simultaneously.
3) There is at least one feasible PU type to run each task.
4) The probabilities and average execution times of scenarios are known a priori. This can be estimated based on historical measurements
5) The static and dynamic power of each PU instance is known a priori via profiling.
6) The reconfiguration time and power of any task on any feasible PU for a given task are known.
7) The average number of transactions between any two communicating tasks in any scenario is known.
8) For any topology, the communication energy per transaction between any two instances of any PUs is known.
9) There is no communication cost between any two communicating tasks running on the same PU (i.e it is negligible).
10) The transactions between any communicating tasks running on different instances of any PU types take place through the minimum path with a communication energy cost denoted as in (3).
11) Each scenario can represent single or concurrent RATs operating at any given time.

For a certain scenario \( S_m \), the energy components are defined as follows:

1) **Static energy consumption**
   For a certain scenario \( S_m \), the total static energy \( E_{s,m} \) is computed through the summation of individual static energy components of each \( \hat{p}_{j,k} \) utilized in \( S_m \) as depicted in (4):

\[
E_{s,m} = \tau_m \sum_{p_j \in P} \sum_k \sigma_j Z_{m,j,k} \quad (4)
\]

2) **Dynamic energy consumption**
   The total dynamic energy \( E_{d,m} \) associated with \( S_m \) depends on the instances used with \( S_m \) as well as the task activities on each of them and it is calculated as in (5):

\[
E_{d,m} = \tau_m \sum_{t_i \in T_m} \sum_{p_j \in P} \sum_k u_{i,j} \delta_j M_{i,j,k} \quad (5)
\]

3) **Communication energy consumption**
   The communication energy associated with a certain scenario is determined by the interconnection among the different PU instances. Generally, the communication energy \( E_{c,m} \) associated with \( S_m \) is the summation of all individual communication energy components from each transaction between each pair of communicating
tasks, which is calculated as shown in (6).

$$E_{c,m} = \sum_{t_i \in T} \sum_{P_j \in P} \sum_{l,m} \theta_{t_i,m} \varepsilon_{l,m} N_{\text{trans}}, l,m \alpha$$  \hspace{1cm} (6)

where

$$\alpha = \sum_{P_j \in P} \sum_{l,m} \sum_{k} \sum_{v} M_{i,j,k} E_{j,k,q,v} M_{i,q,v}$$

such that

$$E_{j,k,q,v} = E_{\text{seg}} \{ \sum_{c} c[|L^c_{j,k} - L^c_{q,v}|] + \sum_{r} r[|L^r_{j,k} - L^r_{q,v}|] \}$$

4) Reconfiguration energy consumption

The total reconfiguration energy $E_{r,m}$ associated with a certain scenario $S_m$ is the summation of the reconfiguration energy components for individual tasks on the corresponding PU instances, which is calculated according to (7):

$$E_{r,m} = \sum_{t_i \in T_m} \sum_{P_j \in P} \sum_{k} \theta_{i,j} \tau_{r,j} M_{i,j,k}$$  \hspace{1cm} (7)

The average total energy consumption for all scenarios is derived as in (8):

$$E_{\text{avg}} = \sum_{S_m \in S} \sum_{P_j \in P} \sum_{k} \chi_m \tau_m \sigma_j Z_{m,j,k} + \sum_{t_i \in T} \sum_{P_j \in P} \sum_{k} \{ \tau_{t_i,j} \delta_{j} + \psi_{j} \tau_{r,j} \} M_{i,j,k} + \sum_{S_m \in S} \sum_{t_i \in T} \sum_{P_j \in P} \sum_{l,m} \{ \chi_m \theta_{t_i,m} \varepsilon_{l,m} N_{\text{trans}}, l,m \cdot \}
\sum_{P_j \in P} \sum_{l,m} \sum_{k} \sum_{v} M_{i,j,k} E_{j,k,q,v} M_{i,q,v} \right\}$$  \hspace{1cm} (8)

The target is to find the optimal mapping $M_{i,j,k}$ in the case of pre-placed instances, and the optimal placement $L^c_{j,k}$, $L^r_{j,k}$ for the controlled placement as well which minimize the overall average energy consumption based on the previous energy modeling. As depicted in (8), the problem is formulated as a constrained binary quadratic programming (CBQP) for the former case, and constrained mixed integer nonlinear programming (CMINLP) for the later. The problem is formalized by the equations in (9). In these set of equations, only the equations (9a), (9b), (9c), (9f), (9g) are considered for the task allocation only problem, and the whole set of equations in is used in joint task allocation and PU placement. The constraint in equation (9b) guarantees that each task is mapped to one and only one feasible PU instance. The constraint in (9c) ensures that the utilization condition is satisfied for each instance in each scenario. The constraint (9d) states that each position defined by a row number and a column number in the grid has at most one allocated processing unit. Additionally, the constraint (9e) guarantees that each PU is allocated to one and only one position in the grid.

$$\min E_{\text{avg}}$$

s.t. $$\sum_{P_j \in P} \sum_{k} f_{i,j} M_{i,j,k} = 1 \hspace{1cm} \forall \ t_i \in T_m, \ S_m \in S$$  \hspace{1cm} (9a)

$$\sum_{t_i \in T} \sum_{P_j \in P} \sum_{l,m} \theta_{t_i,m} u_{i,j} M_{i,j,k} \leq Z_{m,j,k} \hspace{1cm} \forall \ S_m \in S, \ \hat{P}_{j,k} \in P$$  \hspace{1cm} (9b)

$$\sum_{P_j \in P} \sum_{k} L^c_{j,k} L^r_{j,k} \leq 1 \hspace{1cm} \forall \ c \in C, \ r \in R$$  \hspace{1cm} (9c)

$$\sum_{P_j \in P} \sum_{k} L^c_{j,k} L^r_{j,k} = 1 \hspace{1cm} \forall \ P_j \in P, \ k$$  \hspace{1cm} (9d)

$$M_{i,j,k} = \{0, 1\} \hspace{1cm} \forall \ t_i \in T, \ P_j \in P, \ k$$  \hspace{1cm} (9f)

$$Z_{m,j,k} = \{0, 1\} \hspace{1cm} \forall \ S_m \in S, \ P_j \in P, \ k$$  \hspace{1cm} (9g)

$$L^c_{j,k} = \{0, 1\} \hspace{1cm} \forall \ c \in C, \ P_j \in P, \ k$$  \hspace{1cm} (9h)

$$L^r_{j,k} = \{0, 1\} \hspace{1cm} \forall \ r \in R, \ P_j \in P, \ k$$  \hspace{1cm} (9i)

IV. PROPOSED HEURISTIC ALGORITHM

We propose an iterative heuristic solution to minimize the overall energy consumption. The proposed solution consists of two parts: An initial solution, defined in algorithm 1 and an iterative solution, defined in algorithm 2, which reallocates the tasks iteratively to enhance the overall energy consumption. Each one consists of task allocation, followed by PU placement. For the task allocation-only problem, where the placement is fixed, the placement algorithm denoted by step 31 in algorithm 1 and step 23 in algorithm 2 is not used.

A. Baseline static energy aware task allocation

Based on the fact that static power of any PU is task independent, there is no direct linear cost function that relates the mapping variable $M_{i,j,k}$ to the total static power dissipation. An initial relaxed cost function, shown in step 6, is used to estimate the average energy consumption of each task on any PU. In (8), the static energy term can be rewritten as $$\sum_{S_m \in S} \sum_{P_j \in P} \sum_{k} \chi_m \tau_m \sigma_j \sum_{i} \theta_{t_i,m} u_{i,j} M_{i,j,k}$$, and it can be relaxed in the cost function by omitting the ceiling function. This initial cost function considers only the effect of static, dynamic, and configuration energy costs. The communication cost is not considered initially because no corresponding costs are available prior to mapping. The set of tasks with the minimum cost function on $P_j$ are included in $T_j$ and all the tasks in $T_j$ are arranged based on utilization in descending order. Then a new energy cost is computed for each $t_i \in T_j$ on several PU instances. The new energy cost on each instance consists of static, dynamic, reconfiguration, and communication energy. Upon computing the static cost of each $t_i$ on $\hat{P}_{j',k'}$, the static energy consumption is included only for set of scenarios with $\theta_{t_i,m} = 1$ that has not been hosted yet on $\hat{P}_{j',k'}$ as shown in step 14. In steps 15 and 16, the dynamic and reconfiguration energy costs are calculated respectively. A preliminary communication cost is calculated for the case of task allocation-only, where the PUs have fixed allocation. In the case of joint allocation and placement, the
communication energy cost is not considered initially. Step 18 shows the preliminary communication cost upon mapping \( t_i \) on \( p_{j,k}' \). In the computation of the communication energy cost, only tasks that are communicating with \( t_i \) and already have preliminary allocation are considered based on their recent allocation. In step 24, the task \( t_i \) is mapped to \( p_{j,k} \) with the minimum total cost.

B. Iterative remapping solution

After the initial mapping, an iterative solution is used to remap tasks to reduce the total average energy consumption as represented in algorithm 2. The number of iterations is indicated by \( \text{no\_iter} \) variable. We assume that the iteration counter \( \text{N\_iter} \) starts from 2, as the first iteration is the initial solution presented in algorithm 1. At each iteration, this solution seeks a PU instance for each task \( t_i \) that maximizes the energy saving. A new energy cost is calculated for each task \( t_i \) on each \( p_{j,k}' \) based on static, dynamic, reconfiguration and communication cost as shown in steps 7 to 11. The communication cost is estimated based on the recent assignment of other tasks communicating with \( t_i \), and recent ordering of PU instances in case of allocation and placement heuristic. As depicted in step 13, the task \( t_i \) is assigned to the PU \( p_{j,k} \) with the minimum total energy cost and satisfies the utilization constraint for all scenarios \( S_t \) hosting \( t_i \). The resulting average energy consumption is reduced with the increase in the number of iterations and the performance approaches that of the optimal solution.

C. PU placement ordering solution

This algorithm is applicable at design time if we consider reordering the PU instances in the grid to minimize the communication energy cost. After the mapping algorithm, the instances that host tasks are defined by the set of occupied instances \( P_o \). The set of placed instances are defined by \( P_p \), and the set of unplaced instances are defined by \( P_u \). The placement ordering is determined based on the average number of transactions between different instances. The algorithm tends to place the instances with higher number of transactions as close as possible. The upper limit of the grid size \((N_i + N_p) \times (N_i + N_p)\), where \( N_i \) is the number of tasks and \( N_p \) is the number of PU types. The actual size of the grid is determined after the PU instances placement. The first instance to be placed is the one with highest average number of the 2-way transactions with other instances. It is placed in the middle of the hypothetical maximum size grid. Then, the instances are placed one by one in the descending order of the average number of transactions with the placed instances. Each instances is placed in the location that minimizes the energy consumption.

V. PERFORMANCE RESULTS

The model in section III is general in nature where application and architecture parameters can be populated via profiling.

<table>
<thead>
<tr>
<th>Algorithm 1 Baseline task allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: ( T_j \leftarrow \emptyset ; \forall ; P_j \in P )</td>
</tr>
<tr>
<td>2: ( W_{m,j,k} = 0 ; \forall ; S_m \in S, ; P_j \in P )</td>
</tr>
<tr>
<td>3: ( U_{m,j,k} = 0 ; \forall ; S_m \in S, ; P_j \in P )</td>
</tr>
<tr>
<td>4: for each ( t_i \in T ) do</td>
</tr>
<tr>
<td>5: Determine ( P_j ) type which minimizes the expected energy consumption based on static, dynamic, and configuration energy costs only such that ( f_{t,i} = 1 ) and ( u_{t,i} \leq 1 )</td>
</tr>
<tr>
<td>6: ( P_j \leftarrow \min { \tau_{t,i} u_{t,i} (\sigma_j + \delta_j) + \psi_{i,j} \tau_{r_{i,j}} } )</td>
</tr>
<tr>
<td>7: ( T_j \leftarrow T_j \cup { t_i } )</td>
</tr>
<tr>
<td>8: end for</td>
</tr>
<tr>
<td>9: for each ( P_j \in P ) do</td>
</tr>
<tr>
<td>10: Sort all tasks in ( T_j ) in descending order according to the utilization ( u_{t,i} )</td>
</tr>
<tr>
<td>11: for each ( t_i \in T_j ) do</td>
</tr>
<tr>
<td>12: for each ( p_{j,k} \in P ) do</td>
</tr>
<tr>
<td>13: Let ( S_{t,j} ) be the set of scenarios which host task ( t_i )</td>
</tr>
<tr>
<td>14: ( S_{\text{cost}i,j,k'} = \sum_{S_m \in S_t} \chi_m \tau_{m,j,k'} (1 - W_{m,j,k'}) \sigma_j )</td>
</tr>
<tr>
<td>15: ( D_{\text{cost}i,j,k'} = \tau_{t,i} u_{t,i} \delta_j )</td>
</tr>
<tr>
<td>16: ( R_{\text{cost}i,j,k'} = \psi_{i,j} \tau_{r_{i,j}} )</td>
</tr>
<tr>
<td>17: if { Allocation only } then</td>
</tr>
<tr>
<td>18: ( C_{\text{cost}i,j,k'} = \sum_{t_j \in T_n} \sum_{S_m} { \chi_m \theta_{t,i,s} \theta_{t,i,m} } E_{\text{seg}} N_{\text{trans}i,s,m} { \left</td>
</tr>
<tr>
<td>19: ( \text{end if} )</td>
</tr>
<tr>
<td>20: ( C_{\text{cost}i,j,k'} = 0 )</td>
</tr>
<tr>
<td>21: ( \text{end if} )</td>
</tr>
<tr>
<td>22: Assign ( t_i ) to ( p_{j,k} ) such that: ( p_{j,k} \leftarrow \arg \min { \text{Est_Cost}_{i,j,k'} } )</td>
</tr>
<tr>
<td>23: Assign ( t_i ) to ( p_{j,k} ) such that: ( p_{j,k} \leftarrow \arg \min { \text{Est_Cost}_{i,j,k'} } )</td>
</tr>
<tr>
<td>24: Assign ( t_i ) to ( p_{j,k} ) such that: ( p_{j,k} \leftarrow \arg \min { \text{Est_Cost}_{i,j,k'} } )</td>
</tr>
<tr>
<td>25: ( U_{m,j,k} + u_{t,i} \leq 1 ; \forall ; S_m \in S_t ) and ( f_{i,j} = 1 )</td>
</tr>
<tr>
<td>26: ( W_{m,j,k} = 1 ; \forall ; S_m \in S_t )</td>
</tr>
<tr>
<td>27: ( M_{j,k} = 1 )</td>
</tr>
<tr>
<td>28: end for</td>
</tr>
<tr>
<td>29: end for</td>
</tr>
<tr>
<td>30: if { Allocation and placement } then</td>
</tr>
<tr>
<td>31: Placement algorithm() ;</td>
</tr>
<tr>
<td>32: end if</td>
</tr>
</tbody>
</table>
To demonstrate the performance of the heuristic algorithm, we assume an environment with 3 scenarios with a total of 8 tasks, where we generate random task graphs of each scenario with random costs. The performance is estimated for different number of PU types and compared with a reference optimal solution. The optimal solution is obtained through CPLEX [14] optimizer solver. The problem can be formulated as a mixed-integer quadratic programming (MIQP) with linear constraints. By setting all the variables to integer values, and limiting the lower and upper bounds to 0 and 1 respectively, the optimal solution for the initial CBQP is obtained. To reduce the simulation time of the solver, we limit the number of nodes to 800000.

The simulation parameters are shown in Table I. We used the same parameters as in [11] in addition to other parameters related to reconfiguration and communication energy. A uniform distribution is assumed for all scenarios.

Figure 2 shows the average normalized energy consumption for different number of iterations of the heuristic with respect to the optimal solution. It is shown that average energy consumption resulting from the heuristic solution is about 1.2 times that of the optimal one. The figure also highlights the energy reductions with multiple iterations. After a few iterations, the solution saturates such that further iterations results in no perceptible improvement. It is shown in Figure 2 that the solution after 3 iterations coincides with the one after 8 iterations.

Figure 3 provides the average normalized execution time with each $\hat{p}_j,k$ such that:

$$\hat{p}_j,k \leftarrow \arg \min_{\hat{p}_j,k'} \left\{ \text{Est}\_\text{Cost}_{i,j,k'} \right\},$$

$$U_{m,j,k} + u_{i,j} \leq 1 \forall S_m \in S_t \text{ and } f_{i,j} = 1.$$
communication energy. The heuristic provides solutions that are close to the optimal binary quadratic programming solution (within 20%) while achieving a speedup in the execution time up to three orders of magnitude compared to the optimal CBQP. Moreover, the heuristic provides ordering of PUs to further reduce the impact of the communication energy cost. This joint allocation and placement achieves additional energy saving up to 30% in the simulated test case.

**REFERENCES**


