Wavefront-MCTS: Multi-objective Design Space Exploration of NoC Architectures based on Monte Carlo Tree Search

Yong Hu
yong.hu@tum.de
Chair of Electronic Design Automation, Technical University of Munich, Germany

Daniel Mueller-Gritschneder
daniel.mueller@tum.de
Chair of Electronic Design Automation, Technical University of Munich, Germany

Ulf Schlichtmann
ulf.schlichtmann@tum.de
Chair of Electronic Design Automation, Technical University of Munich, Germany

ABSTRACT

Application-specific MPSoCs profit immensely from a custom-fit Network-on-Chip (NoC) architecture in terms of network performance and power consumption. In this paper we suggest a new approach to explore application-specific NoC architectures. In contrast to other heuristics, our approach uses a set of network modifications defined with graph rewriting rules to model the design space exploration as a Markov Decision Process (MDP). The MDP can be efficiently explored using the Monte Carlo Tree Search (MCTS) heuristics. We formulate a weighted sum reward function to compute a single solution with a good trade-off between power and latency or a set of max reward functions to compute the complete Pareto front between the two objectives. The Wavefront feature adds additional efficiency when computing the Pareto front by exchanging solutions between parallel MCTS optimization processes. Comparison with other popular search heuristics demonstrates a higher efficiency of MCTS-based heuristics for several test cases. Additionally, the Wavefront-MCTS heuristics allows complete tracability and control by the designer to enable an interactive design space exploration process.

KEYWORDS

NoC, MCTS, multi-objective design space exploration

ACM Reference Format:
Yong Hu, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. 2018. Wavefront-MCTS: Multi-objective Design Space Exploration of NoC Architectures based on Monte Carlo Tree Search. In *IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD '18), November 5–8, 2018, San Diego, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3240765.3240863

1 INTRODUCTION

In modern chips, more and more processing elements (PEs) are integrated into a single Multi-Processor System-on-Chip (MPSoC). This challenges the interconnect design, which must provide a good trade-off between communication latency with power and area overheads. Here, Network-on-Chip (NoC) interconnects have established themselves as a popular solution due to their good scalability for a high number of PEs. However, application-specific MPSoCs, e.g. smartphone chips, make use of a high number of fixed-function PEs such as Video decoders or DMAs. For such systems, the network traffic is highly unbalanced. This motivates designers to develop application-specific NoC architectures, which are custom-fit for their MPSoCs.

Generally, there exist two types of NoC synthesis approaches: top-down approaches and iterative approaches. The top-down approaches start from the system specification and make decisions step by step based on heuristics. E.g if two PEs communicate with each other very heavily, they will be connected to the same router. However, top-down approaches have a common drawback. The top-level decisions are made before obtaining the complete NoC architecture, so performances and costs can only be estimated with high-level models, for which the accuracy of estimation cannot be guaranteed. Iterative design space exploration (DSE) approaches start from a given single or population of NoC architectures. They iteratively explore new NoC architectures in the design space and search for the one that has the optimal performance at an acceptable cost. The huge size of the NoC design space challenges these approaches, hence, calling for efficient exploration heuristics. Besides, designers often want to be able to trace and control the changes made during the DSE process. But existing heuristics such as Genetic Algorithms (GAs) make it hard for the designer to track the decisions applied.

This paper proposes a new iterative NoC DSE approach that supports an interactive flow while also obtaining high efficiency comparable and often better than existing heuristics. At the core of the approach, the Monte Carlo Tree Search (MCTS) heuristic is applied. For this, the DSE is modelled as a Markov Decision Process (MDP) using graph rewriting. The algorithm can search for a single optimized design by weighting the multiple objectives such as power and average latency in a single reward function. Alternatively, we also propose the Wavefront-MCTS heuristic that efficiently computes the complete Pareto front for multiple objectives. The contributions of this paper are the following:

- The formulation of the NoC DSE as Markov Decision Process (MDP) with graph rewriting. The MDP model enables a traceable and interactive flow that offers designers insight and control of the DSE process.
- The application of the Monte Carlo Tree Search (MCTS) algorithm for NoC DSE. Compared with a simulated annealing (SA) algorithm and a genetic algorithm (GA), the MCTS algorithm shows obvious improvements in exploration efficiency.

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240863

- The Wavefront-MCTS algorithm for obtaining the complete Pareto front. The algorithm uses a set of *Max* reward functions to guide parallel MCTS-based explorations towards non-dominated NoC architectures. The Wavefront modification of MCTS enables an exchange between these parallel explorations to increase efficiency. Compared with the existing popular algorithm NSGA-II, the Wavefront-MCTS heuristic covers larger parts of the Pareto front due to the systematic exploration enforced by the *Max* reward function.

The remainder of this paper is organized as follows. Sec. 2 discusses related work and the MCTS heuristic. Sec. 3 explains the MCTS-based DSE for NoCs in detail. Sec.4 introduces the Wavefront-MCTS. Sec. 5 shows experimental results that demonstrate the efficiency and tracability of the DSE. Sec. 6 concludes the paper.

2 RELATED WORK

2.1 State of the Art in NoC DSE

There has been extensive work on efficient design space exploration heuristics for application-specific NoC architectures. As was already pointed out, they can be categorized into iterative and top-down approaches.

In terms of *iterative* approaches, the method in [12] uses Tabu search in the NoC synthesis for application-specific MPSoCs. It starts from an initial solution and iteratively explores neighborhoods. Explored designs are marked Tabu to avoid cycles in the search. The approach in [10] proposes a multi-commodity flow (MCF)-based scheme to find the optimal NoC topology that minimizes power consumption under communication latency constraints. It also provides an approximation algorithm, which they report is much faster than the commercial LP solver CPLEX. The approach in [4] applies a GA to explore the NoC design space. It models the connections in the topology as endpoint pairs and endpoints are regarded as genes. The connections are modified randomly, when the gene is mutated generating new designs. The work in [15] uses a SA algorithm to explore the NoC space. The approach filters out designs with large delay to support hard timing constraints in QoS systems. The work in [14] also applies SA in the design space exploration and it further models the memory system including caches. The work in [6] uses the STAGE machine learning algorithm to optimize the placement of planar and vertical communication links in 3D-NoCs for energy efficiency. It iteratively applies a base search and a meta search. The base search runs in a greedy way to find local optima and generate new training data. The meta-search learns from features of the training data and tries to explore a good start state for the base search. And the features consist of average hop count, products of communication hops and bandwidth and clustering coefficient.

In a *top-down* fashion, the approach in [16] uses spectral clustering to assign PEs to routers. In a subsequent step, the routing between PEs is determined by the A* algorithm. A similar approach with different clustering algorithm was presented in [17]. As was already pointed out, the assignment of PEs to routers is a key decision for application-specific NoCs. In these approaches, this decision is done based on a cost function that combines latency, bandwidths and floorplan information to find the best cluster of PEs. Yet, no simulation can be used at this level of abstraction to test the decision. Especially for top-down approaches or the cross-over steps of GAs, it is very hard for the designer to trace and understand the decisions applied by the heuristics to the NoC. Here, SA has the advantage that a sequence of changes to the NoC can be back-traced. In our experience, this is a great advantage for designers to get more insight and control into the design space exploration, in order to enhance the trust in the exploration results. The MCTS-based heuristic in this paper also allows such insights, yet, has higher efficiency than SA.

2.2 Background on MDP and MCTS

As the name implies, MDPs model decision processes. They are described as sequences of states and actions as following:

(1) State s: The decision space is represented with a finite number of states
(2) Reward function $Q(s)$: The quality of a state s
(3) Action a: An action can be applied on state s to generate an output state s'. For each state s, its available actions are represented as the set $A(s)$
(4) The transition function $f(s, a)$: Whenever the action a is applied on the state s, it gives the output state $s' = f(s, a)$

The exploration of an MDP is in principle the search for the sequence of actions that can yield the highest reward from given a specific initial state s_0.

MCTS is used as an effective method to explore MDPs [3], including in the famous AlphaGo. It structures the exploration of the MDP as a tree T. Each node v in the tree T stores one state $s(v)$. MCTS stores the initial state s_0 in the root node of the tree v_{root} with $s(v_{root}) = s_0$. When an action $a \in A(s)$ is applied on $s(v)$, a new node v' is created with a new state $s(v') = f(s, a)$ and it is added as child of node v to the MCTS tree. Besides the state, a node v also stores the reward function $Q(s(v))$ and the visiting times $N(v)$. For game engines $Q(s(v))$ is often the winning rate. For NoCs it could be a cost function combining area, power and latency. The visiting times of a node v is always initialized as one when a node v is created, that is $N(v) = 1$. Each time when a new child is added to successors of the node v, $N(v)$ will be increased by one. As shown in Fig. 1, the MCTS solves the MDP problem iteratively by exploring successor nodes of the root following the *tree policy* and *default policy*.
The tree policy includes the selection and the expansion:

(1) In the *selection* step, the most urgent node v_u is selected for expansion. There exist different formulations for urgentness. The most popular one is the Upper Confidence Bound for Trees (UCT) [13]. It calculates the urgentness as:

$$UCT(v) = Q(s(v)) + 2C_p\sqrt{\frac{2\ln(N(v_{root}))}{N(v)}} \quad (1)$$

Here C_p is a constant weight factor depending on the range of $Q(v)$. The $UCT(v)$ function may have both positive and negative values. Larger values indicate higher urgentness such that the most urgent node is selected by:

$$max_{v \in T_r} UCT(v) \rightarrow v_u \quad (2)$$

The sub-tree T_r only includes successor nodes of the current root node v_{root}.

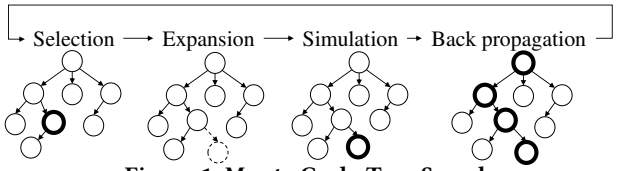


Figure 1: Monte Carlo Tree Search

(2) In the *expansion* step, an action $a \in A(s(v_u))$ is applied on the state of the most urgent node to create a new child node v' with $s(v') = f(s(v_u), a)$.

The default policy includes the simulation and the backpropagation:

(1) In the *simulation* step, the newly generated child node is evaluated by computing $Q(s(v'))$.
(2) In the *backpropagation* step, the evaluation results $Q(s(v'))$ are sent through the parent node to all predecessor nodes until the current root node v_{root}. This also triggers the predecessors to update their visiting times $N(v)$.

MCTS keeps adding and exploring new successor nodes to the current root node v_{root} based on the tree policy and default policy until a pre-defined computation budget runs out. Then MCTS selects the best direct child node of the current root node as new root node. This is equal to selecting the action a that leads from the previous root node to the new root node. The best direct child node is either selected as the one that leads to the successor node of the root node with highest Q-value or the direct child node with the highest visited times. This is a major advantage of MCTS over other heuristics such as SA. MCTS discovers late reward gains, which are not obtained in direct children of the root node but in grandchildren and even their successors. Additionally, the UCT formulation balances exploration versus exploitation very well.

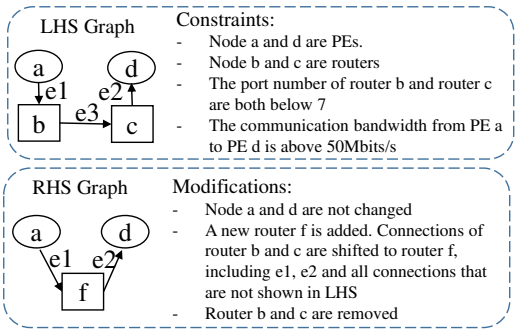
3 MCTS-BASED DSE FOR NOCS

3.1 MDP Model using Graph Rewriting

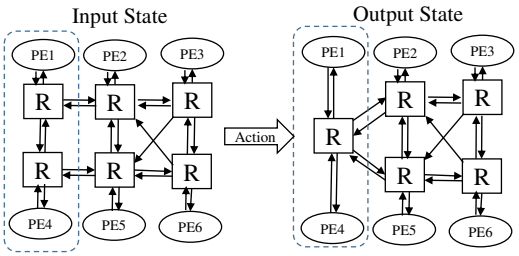
The NoC architecture can be well described with a graph: PEs and routers are represented by nodes and physical links are represented by directed edges between nodes as shown in Fig. 2(b). A modification of the NoC architecture can be formulated using the theory of *graph rewriting* [8]. Graph rewriting is based on rules, which consist of a left-hand-side (LHS) graph and a right-hand-side (RHS) graph. The LHS graph describes a region of interest (ROI), where to modify the graph. The LHS can be defined not only based on the graph structure, but also with some property constraints such as node degree, the distance between two nodes or the name or property of a node. The RHS describes how to modifiy the ROI. The modifications can also be made on both the graph structure and its properties. A rule execution on a so-called design graph is a three step process:

(1) In the *match* step, we search for the LHS in the design graph. All matches between LHS and design graph are stored. For a single rule there can be several matches.
(2) In the *selection* step, one match is chosen.
(3) In the *apply* step, the matching sub-graph in the design graph is replaced by the RHS graph of the rule.

Simply stated, we replace a sub-graph of the design graph by a new sub-graph. An example of a the graph rewriting rule for a NoC modification is illustrated in Fig. 2. The LHS includes two PEs a and



(a) Graph Rewriting Rule



(b) Execution of the Rule

Figure 2: Illustration of Graph Rewriting for NoCs

d and two routers b and c. There needs to exist a connection from a to b, b to c and c to d for the LHS to match. Besides, router b and c are constrained to have less than 7 ports and the communication from a to d should be above $50Mbits/s$. Since two PEs communicate heavily and the connected routers are still relatively small, we can merge two routers into one, in order to reduce communication latency. This modification is described in the RHS of the rule in Fig. 2(a). The rule is applied on a design graph in Fig. 2(b). During the match step, we look for the LHS in the design graph using graph isomorphism. In this example, a single match for the LHS is marked in the host graph, which consists of the PE1, PE2 and their connected routers. In the apply step, the design graph is modified according to the RHS to generate a new design graph representing a new NoC architecture.

Defining a set of graph rewriting rules allows us to model the NoC exploration as MDP. Each NoC architecture represents a state s described by a graph. Each graph rewriting rule is an action. If there exist a match between the LHS of the rule and the NoC architecture, this rule can be applied. Then the action a is defined to be available for the state s, that is $a \in A(s)$. The transfer function with $s' = f(s, a)$ is implemented by executing rule a on the initial NoC graph s to obtain a new NoC Graph representing the new state s'. We encode basic NoC modifications as rules, including adding/removing a router, adding/removing a directed link between a pair of routers and shifting a PE to a a different router. Any other kind of modification can always be achieved with a combination of these basic rules.

3.2 Weighted-Sum Reward Function

The only missing component of our MDP is the reward function $Q(s)$, which represents the quality of the state s. The reward function depends on the target of the DSE. When we are looking for a

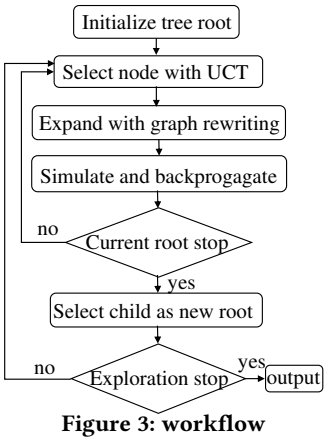


Figure 3: workflow

single good trade-off between latency and power, we can weight for example two objectives power $g_1(s)$ and average latency $g_2(s)$ to obtain a single scalar reward function $Q_{ws}(s)$:

$$Q_{ws}(s) = -1 \cdot (w_1 \cdot g_1(s) + w_2 \cdot g_2(s)), \tag{3}$$

where w_1 and w_2 are weight factors. the multiplication by -1 is necessary as the reward function must be maximized while latency and power should be minimized.

After modelling the DSE as an MDP problem, we are able to apply the MCTS algorithm. The workflow is shown in Fig. 3. It starts from an arbitrary NoC architecture and uses it as the root. The selection, expansion, simulation and backpropagation are executed iteratively to search for the best action to optimize the root. When the budget runs out, the root is replaced by the child node that is generated from the best action. With the new root, the MCTS is re-started. When the complete exploration budget runs out, the algorithm outputs the best one among the explored nodes as well as the sequence of actions that lead to this design. This allows to backtrace all modifications done by the MCTS heuristic.

4 WAVEFRONT-MCTS

It is hard to choose the weight factors that can end up with the desired trade-off between average latency and power. In order to obtain better insight into the trade-off, we can compute the complete Pareto front. The Pareto front is made up of the non-dominated points, which are defined for two competing objectives g_1 and g_2 as:

- Dominance: The objective vector $\mathbf{g_a} = [g_1(s_a), g_2(s_a)]$ is dominated by $\mathbf{g_b}$, if is $\mathbf{g_b}$ is better in one of the objectives and not worse in the other.
- Non-dominated point: The state s_a with objective vector $\mathbf{g_a} = \mathbf{g}(s_a)$ is non-dominated, if there exists no other state s_b with $\mathbf{g_b} = \mathbf{g}(s_b)$ that dominates $\mathbf{g_a}$. The objective vectors of all non-dominated states form the so-called Pareto front.

One can get different points of the Pareto front by modifying the weight vectors. Yet there exist better methods to explore the Pareto front in a more systematic way as introduced in the following.

4.1 Max Reward Function

We use Max terms to search for non-dominated states, also referred in optimization theory as *MinMax* optimization. The reward

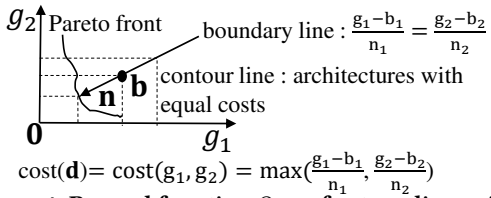
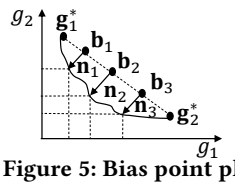


Figure 4: Reward function Q_{max} for two-dimension objectives



Bias point:
$\mathbf{b}_1 = 0.75 \times \mathbf{g}_1^* + 0.25 \times \mathbf{g}_2^*$
$\mathbf{b}_2 = 0.5 \times \mathbf{g}_1^* + 0.5 \times \mathbf{g}_2^*$
$\mathbf{b}_3 = 0.25 \times \mathbf{g}_1^* + 0.75 \times \mathbf{g}_2^*$
Normalization vector:
$\mathbf{n}_1 = \mathbf{n}_2 = \mathbf{n}_3 = \mathbf{g}_1^* + \mathbf{g}_2^*$

Figure 5: Bias point placement for two objectives

function is given as:

$$Q_{max}(s, \mathbf{b}, \mathbf{n}) = -1 \cdot \max(\frac{g_1(s) - b_1}{n_1}, \frac{g_2(s) - b_2}{n_2}) \tag{4}$$

Here, $\mathbf{b} = [b_1, b_2]^T$ is the bias vector and $\mathbf{n} = [n_1, n_2]^T$ is the normalization vector. Setting the normalization vector and bias vector, the $Q_{max}(s, \mathbf{b}, \mathbf{n})$ searches for non-dominated states with different trade-offs between the objectives. This is illustrated in Fig. 4. A boundary line separates the objective space into two parts. Below and on the boundary line, g_1 and g_2 satisfy that $\frac{g_1-b_1}{n_1} \geq \frac{g_2-b_2}{n_2}$, the reward function depends on g_1 only. So for architectures below the boundary line, if they have the same g_1, then they have same reward. This creates a set of vertical contour lines as shown in Fig. 4 where we have to lower g_1 to gain reward. Similarly, above the boundary line, there exists a set of horizontal contour lines, where we have to lower g_2 to gain reward. The highest reward is gained for the crossing between the Pareto front and the boundary line. The boundary line always includes the bias point **b** and its slope is determined with the normalization vector **n**.

In order to obtain a good spread of points on the Pareto front, the bias vectors and normalization vectors are selected according to the so-called Individual Minima IM. For each objective g_m, we set:

$$Q_{im}(s, m) = -1 \cdot g_m(s) \tag{5}$$

Running MCTS for the single objective g_m, we can obtain the state s^{*m} and the IM $\mathbf{g}^{*m} = \mathbf{g}(s^{*m})$. If the bias vector **b** is placed near the $\mathbf{g}^{*m}$, it is more probable to obtain Pareto point that is close to $\mathbf{g}^{*m}$. One can use the IM to select the bias vectors and normalization vector as follows [5]:

$$\mathbf{b}_n = \mathbf{b}(\mathbf{w}_n) = \mathbf{G}^* \cdot \mathbf{w}_x \quad , \quad \mathbf{n} = \mathbf{g}^{*1} + \mathbf{g}^{*2} \tag{6}$$

where $\mathbf{G}^* = [\mathbf{g}^{*1}, \mathbf{g}^{*2}]$ and the sum of the elements of the weight vector $\mathbf{w}_n$ is always 1. One example is shown in Fig. 5. It uses three bias vectors, which are placed equal-distantly between the two IM by choosing the weight vectors accordingly. This leads to a systematic exploration of different parts of the Pareto front. This method can be extended to support arbitrary number of objectives, e.g, if we want to see the trade-off between average latency of different use case scenarios or also want to consider area.

4.2 Wavefront Feature

For each objective g_m we run one MCTS exploration with reward function $Q_{im}(s, m)$ and for a pre-chosen set of bias points $\mathbf{b}_n$, we run an MCTS exploration using the reward function $Q_{max}(s, \mathbf{b}_n, \mathbf{n})$. To increase the efficiency of this exploration, we propose the Wavefront-MCTS algorithm. Instead of running independent MCTS explorations, Wavefront-MCTS stores all explored states in one joint MCTS tree. Explorations are parallelized at root level, which means that each exploration can select one node from the tree as its own root and grow according to its own reward function. Meanwhile, different explorations can also exchange root nodes with each other. This brings the following benefits:

- The efficiency is further improved because there can exist overlaps among the searching spaces of explorations. Once a space is explored, the information can spread to others.
- It allows to escape from local optima. When an exploration gets stuck in a local minimum, other exploration may find better states in other regions of the search tree and share the information.
- The bias point depends on the global optimum $\mathbf{g}^{*m}$ for each single objective. It is possible that the exploration for the individual minimum with $Q_{im}(s, m)$ gets stuck in a local optima. Another exploration may then find a better $\mathbf{g}^{*m}$ than the current known best for a certain objective. In this case, our algorithm can exchange the information and all explorations re-calibrate to the new $\mathbf{g}^{*m}$.

In order to exchange root nodes between explorations, the MCTS algorithm needs to be adapted. The pseudo-code of the Wavefront-MCTS is given in Algorithm 2. The inputs consist of an initial state s_0, a set of weight vectors $\mathbf{w}_n$ and the individual minima $[\mathbf{g}^{*1}, \mathbf{g}^{*2}]$. The algorithm starts with the initial state as root nodes for all explorations and builds up the MCTS tree. The weight vectors determine N different search directions. The MCTS tree grows towards those N directions. For each exploration, it still follows the basic workflow in Fig. 3. It first tries to search the best action for the current root $v_{root,n}$ by iteratively running the following steps:

- SELECT($v_{root,n}, \mathbf{w}_n, \mathbf{G}^*$) : This function selects and returns the most urgent node v_u in the sub-tree with given root $v_{root,n}$. For each rule, if its LHS can be found in the state $s(v_u)$, it is one of the available actions. If the root has an available action that is never applied, then the root and this rule will be selected. Otherwise, the algorithm searches among all child nodes that have at least one available action and select the one with highest value in the UCT($v, \mathbf{w}, \mathbf{G}^*$) function as shown in Alg. 1.
- EXPAND(v_u) : This function expands the node v_u using graph rewriting. If node v_u has non-applied available actions, one of them will be randomly selected and applied on v_u. Otherwise, the action will be selected among all available actions. The process of applying a rule is shown in Sec. 3.1. If there exist multiple matches of the LHS, a random match will be selected and get rewritten. The newly generated state creates a new node v'. It is added as a child of node v_u and its visited time is initialized as $N(v') = 1$. The edge between the new child and v_u stores the rule and the selected match to get a complete trace of optimization process.
- SIMULATE(V') : Unlike normal MCTS, the newly generated node is not simulated immediately in Wavefront-MCTS. Instead, newly generated nodes are stored in the set V'. After expansions of all N search directions are finished, the set of newly generated nodes V' are simulated together to obtain their performances and costs including area, power and average latency. The advantage of this flow is that simulations can be parallelized and distributed to multiple threads and even multiple machines. This can significantly increase the speed of exploration.
- BACKPROPAGATE(V') : After simulation, the set of newly generated nodes, V', is then backpropagated. For each node $v \in V'$, we trace all of its predecessor nodes, including parent, grandparent until the tree root. The visited times of those predecessor nodes are increased by one.
- UPDATEMINIMA($\mathbf{G}^*$,V') : each newly generated node $v' \in V'$ will be compared with current individual minima $\mathbf{G}^* = [\mathbf{g}_1^*, \mathbf{g}_2^*]$. If a node is found better than $\mathbf{g}_1^*$ or $\mathbf{g}_2^*$, the individual minima will be replaced with the new one.

Algorithm 1 UCT Function based on Biased Max Reward Function

Input: Node v
Weight vector $\mathbf{w}_n$
Individual Minima $\mathbf{G}^* = [\mathbf{g}_1^*, \mathbf{g}_2^*]$
Output: UCT value of node v for weight vector $\mathbf{w}_n$

$\mathbf{b}_n \leftarrow \mathbf{G}^* \cdot \mathbf{w}_n \qquad \mathbf{n} \leftarrow \mathbf{g}^{*1} + \mathbf{g}^{*2}$
$UCT \leftarrow Q_{max}(s(v), \mathbf{b}_n, \mathbf{n}) + C_p\sqrt{\frac{2lnN(v_{root})}{N(v)}}$

return UCT

As shown in the workflow in Fig. 3, after a certain number of iterations, the MCTS chooses the most promising action to update its root. For Wavefront-MCTS, the most promising action is the one that created the direct child node of any root node, which leads to the node with highest Q_{max} value. This is described in Line 22 to 34 of Alg. 2. The current root nodes of different search directions are stored in the set V_{root} and their successor nodes are stored in the set V_s. The roots of various search directions are updated one by one. Each direction n searches for its best node by comparing the quality of nodes using its Q_{max} reward function together with its weight vector $\mathbf{w}_n$. One important feature is that the root update searches among the direct child nodes of all root nodes $v_{root,n}$. In this way, all search directions can share information between each other. And the best node for direction n may exist not in the sub-tree of its own root $v_{root,n}$ but in the sub-tree of another search direction, e.g., due to local minima or when one exploration discovers a very competitive state. In this case, the exploration can move to the root of another search direction.

5 EXPERIMENTAL RESULTS

Our experiments are performed with several benchmark applications. They consist of the multimedia application obtained from [9], the SoC benchmark used in the European project NaNoC [1], the industrial mobile phone application [2] and a synthetic MPSoC application with uniform distributed communication load. The multimedia application and MPSoC application each consist of 16 PEs,

Algorithm 2 Wavefront-MCTS Algorithm

Input: Initial state s_0
  Weight vectors $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_N]$
  Individual Minima $\mathbf{G}^* = [\mathbf{g}_1^*, \mathbf{g}_2^*]$
Output: Non-dominated states S^*, MCTS Tree T

$T \leftarrow$ ADDNODE($\emptyset, v_0$) with $s(v_0) = s_0$
for n = 1 to N **do**
  $v_{root,n} \leftarrow v_0$, with $s(v_0) = s_0$
end for
for l = 1 to L **do**
  while within computation budget **do**
    $V' \leftarrow \emptyset$
    for n = 1 to N **do**
      $v_u \leftarrow$ SELECT($v_{root,n}, \mathbf{w}_n, \mathbf{G}^*$)
      $v' \leftarrow$ EXPAND(v_u)
      $V' \leftarrow V' \cup \{v'\}$
      $T \leftarrow$ADDNODE(T, v')
    end for
    SIMULATE(V')
    BACKPROPAGATE(V')
    UPDATEMINIMA($\mathbf{G}^*$,V')
  end while
  $V_s \leftarrow \emptyset$, $V_{root} \leftarrow \emptyset$
  for n = 1 to N **do**
    $V_{root} \leftarrow V_{root} \cup \{v_{root,n}\}$
    $V_s \leftarrow V_s \cup \{v_u |$ v_u is on subtree with root $v_{root,n}\}$
  end for
  for n = 1 to N **do**
    $\mathbf{b}_n \leftarrow \mathbf{G}^* \cdot \mathbf{w}_n$ $\mathbf{n} \leftarrow \mathbf{g}^{*1} + \mathbf{g}^{*2}$
    $v_{best} \leftarrow \arg\max_{v \in V_s} Q_{max}(s(v), \mathbf{b}_n, \mathbf{n})$
    while the parent of $v_{best} \notin V_{root}$ **do**
      $v_{best} \leftarrow$ the parent of v_{best}
    end while
    $v_{root,n} \leftarrow v_{best}$
  end for
end for

$S^* \leftarrow$ FINDNONDOMINATEDSTATES(T)
return S^*, T

the mobile application consists of 22 PEs and the NaNoC application consists of 25 PEs. The initial NoC designs are created based on the classical mesh architecture.

We use ORION 3.0 [11] to provide a good estimation on the NoC power and area. Cycle-accurate SystemC simulations are applied to obtain the communication latency. They are executed in parallel to obtain a fast DSE. The LISNoC [18] is used as baseline to generated VHDL codes at RTL level to calibrate the ORION and the SystemC models. In order to obtain a working NoC, a routing must be set as well. Here we use a custom deterministic routing generator. The routing paths for each flow are found by solving an Integer Linear Program (ILP) that minimizes the routing path length and conflicts at the router output ports.

Table 1: Comparison between Optimization Heuristics

Testbench	Method	Area (um^2)	Power (mW)	Latency (Cycles)	$Q_{ws}(s)$
Multimedia	Initial	324207	66.7	43.28	-183.3
	SA	62349	13.01	15.13	-46.9
	GA	225782	47.2	28.98	-126.97
	MCTS	39183	8.12	13.89	-36.68
Mobile	Initial	563529	115.28	16.2	-306.1
	SA	303679	62.15	12.72	-204.89
	GA	302679	62.16	14.79	-225.6
	MCTS	114253	23.52	13.56	-165
NaNoC	Initial	571403	116.97	15.2	-249.47
	SA	278813	57.01	12.42	-165.88
	GA	305318	63.21	14.14	-194.08
	MCTS	96456	19.87	13.3	-149.56
MPSoC	Initial	324207	66.31	20.47	-287.59
	SA	188438	38.75	15.57	-204.14
	GA	204266	41.53	18.33	-235.21
	MCTS	138016	28.86	16.19	-197.97

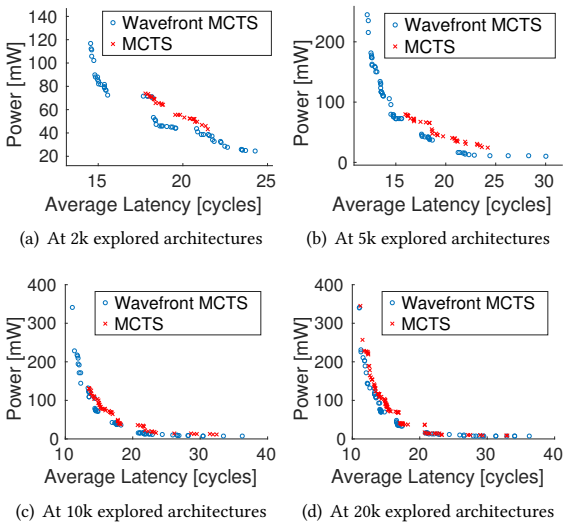
5.1 Comparison of MCTS with SA and GA

Our experiments compare the MCTS optimization heuristic with other popular methods: Simulated Annealing (SA) and a Genetic Algorithm (GA). All three methods start from the same initial designs and are allocated the same computation budget. They also share the same overall weighted sum reward function $Q_{ws}(s)$ for fair comparison. The area is not used in the reward function because it is correlated to the power as can be seen from the experimental results in Table 1. In application-specific NoC synthesis, other parameters and goals may be of importance: Some flows may be more latency-sensitive than others, several use cases may exist and must be efficiently executed on the same platform and so on. This can all be included in the weighted reward function as long as weights are supplied to set the preference of each goal.

Compared with initial designs, all three methods obtain significant improvements. Clearly, all benchmark systems do benefit from a custom-fit NoC architecture. For the multimedia application, the SA improves the reward function by 74.21%, GA by 30.73% and MCTS by 79.99%. Also in all other benchmark systems, MCTS achieves the highest reward gain. So, as can be seen, MCTS is a very competitive heuristic. Of course, each heuristic can be further optimized to perform better on the NoC DSE problem. We believe the comparison is fair, as a rather standard implementation of each heuristic is used. MCTS performs very well and better than the other heuristics because it is designed to balance exploration vs. exploitation very well.

5.2 Wavefront Feature

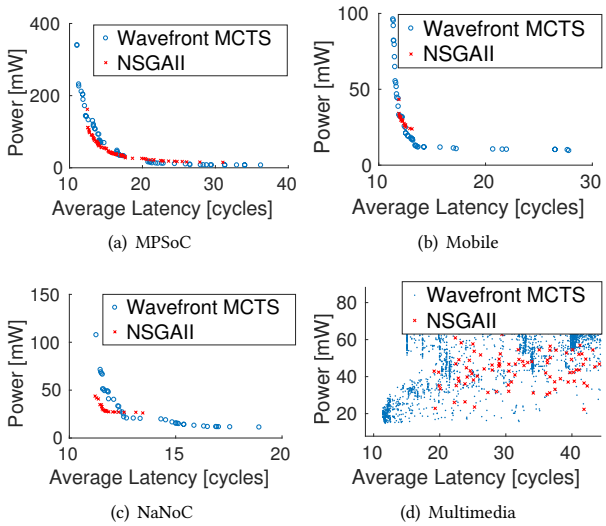
The Wavefront feature was introduced to improve the efficiency of standard MCTS when we want to compute the complete Pareto Front. To show its efficiency, we compute the Pareto front between power and average latency for the MPSoC benchmark. Both use 9 search directions with the same set of weight vectors using the Q_{max} reward function. Their exploration budgets are both set to



(a) At 2k explored architectures (b) At 5k explored architectures

(c) At 10k explored architectures (d) At 20k explored architectures

Figure 6: Comparison of Wavefront-MCTS vs. MCTS without Wavefront Feature



(a) MPSoC (b) Mobile

(c) NaNoC (d) Multimedia

Figure 7: Wavefront-MCTS vs.NSGA-II Comparison

20k NoC simulations. Fig. 6 shows different stages of the exploration. At an early stage, when Wavefront-MCTS and MCTS explored 2k archtiectures each, the Wavefront-MCTS is already significantly ahead of MCTS. Similar behavior can be observed in Fig. 6(b) when each of them explored 5k architectures. In Fig. 6(c), the non-dominated states of MCTS are also not anymore dominated by Wavefront-MCTS, but the Wavefront-MCTS covers a much larger range of the Pareto front. Wavefront-MCTS already finds a stable front after exploring 10k architectures, while this is observed for standard MCTS for 20k architectures. Hence, the Wavefront feature improves efficiency significantly without adding any significant computational overhead.

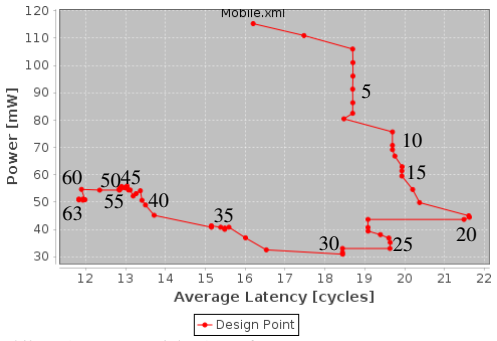
5.3 Comparison: Wavefront-MCTS vs. NSGA-II

Our experiments also compare the Wavefront-MCTS algorithm with one of the most popular multi-objective optimization heuristics, the NSGA-II algorithm [7]. The results are shown in Fig. 7. For the MPSoC benchmark in Fig. 7(a), the non-dominated architectures of Wavefront-MCTS range from (11.04 , 340) to (36.23 , 7.195). In comparison, the ones of NSGAII range from (12.45 , 162) to (31.03 , 15.22). The minimum power found by Wavefront-MCTS is 52.73% lesser than the minimum power found by NSGA-II and the minimum latency is 11.33% lesser. For the mobile benchmark in Fig. 7(b), the Wavefront-MCTS also covers a wider range of the Pareto front. The discovered non-dominated architectures of both methods do not dominate one or the other. For the NaNoC benchmark in Fig. 7(c), the Wavefront-MCTS covers a much wider range of the Pareto front. Yet, the solution discovered by NSGA-II dominate the ones from MCTS in some parts of the front. Here Wavefront-MCTS did not yet stabilize and discover the complete front. For the multimedia benchmark in Fig. 7(d), all explored architectures are shown. As becomes obvious, there exist architectures that can both optimize power and latency very well for the system. These were discovered by the Wavefront-MCTS algorithm, while they were not found by NSGA-II. So overall ,we can conclude, that Wavefront-MCTS is very competitive when we target to explore the complete Pareto front. Especially the systematic search with the Q_{max} reward function allows to discover the complete range of the Pareto front very efficiently.

5.4 Interactive DSE

One important feature of the MCTS-based approach is that it can be traced very well and allows for an interactive DSE. This is demonstrated with the mobile benchmark. The initial architecture is a 5x5 mesh as shown in Fig. 9(a). The trace of an exploration path is shown in Fig. 8. This is the path taken by the actions to move from root node to next root node. It shows the performance and power gain for each action on the optimization path together with each action which are shown in the lower part of the Figure. Architectures can also be traced along the optimization path. The intermediate architecture after applying 30 actions is shown in Fig. 9(b). Compared with the initial architecture, many links were removed as well as some redundant routers. As a result, the power and area are reduced, but meanwhile due to less routing resources, some flows have to choose longer and more crowded paths. This is confirmed in Fig. 8 that architecture with index 30 has a low power but high latency. The final architecture is shown in Fig. 9(c). Compared with the intermediate architecture in Fig. 9(b), PEs were re-grouped. They are those that communicate with a high bandwidth between each other. This is beneficial to reduce latency. But for power, the influence is not straightforward. On one hand, actions removed more redundant links and routers reducing power. On the other hand, actions created some high-radix routers increasing power. It is difficult to predict which effect is stronger. But this is shown in the performance trace in Fig. 8. Compared with the intermediate architecture of index 30, the last output architecture has a much lower latency and a slightly higher power. In general, designers can also control the exploration interactively. One may only run MCTS for a limited number of actions, then apply actions manually, or re-start the exploration from a given design, possibly, modifying the preferences for the goals.

The trace also shows how MCTS balances exploration vs. exploitation. From short term view, the first two actions seem to be a



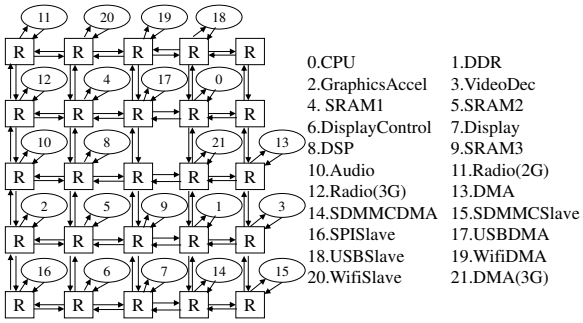
Mobile -> 1 : Remove Link R1 to R6
1 -> 2 : Remove Link R16 to R21
2 -> 3 : Remove Link R12 to R13
3 -> 4 : Remove Link R10 to R11
4 -> 5 : Remove Link R14 to R13
5 -> 6 : Remove Link R22 to R17
6 -> 7 : Remove Link R19 to R14
7 -> 8 : Shift DMA from R2 to R18
… …
30 -> 31 : Shift SRAM1 from R18 to R16
31 -> 32 : Shift CPU from R8 to R2
32 -> 33 : Add Link R24 to R4 … …

Figure 8: Exploration Trace for Mobile Benchmark

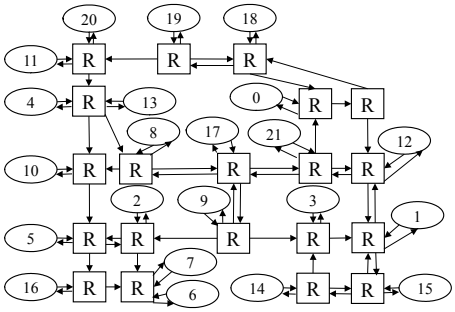
bad decision, because they obtain only a slight power reduction but sacrifice a lot of latency. However, from long term view, they are beneficial because, in the following actions, the power is reduced while latency remains almost constant. To select the best action for the current node, MCTS not only explores the direct children, but also explores successor nodes that are generated after multiple actions based on the UCT function.

6 CONCLUSION

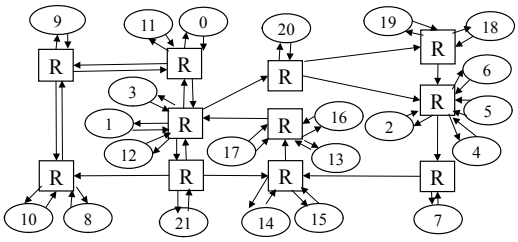
This paper formulates the NoC DSE as MDP and uses graph rewriting to define MDP actions and the transition function. This approach supports an interactive design and traceability by the designers. The MCTS heuristic is applied and shows high efficiency compared to the classical GA and SA algorithm. Further, Wavefront-MCTS is proposed. Compared with the popular NSGA-II algorithm, it can cover a larger range of the Pareto front while not sacrificing quality of non-dominated architectures.



(a) Start architecture (After 0 Actions)



(b) Intermediate architecture (After 30 Actions)



(c) Final architecture (After 63 Actions)

Figure 9: Explored Architectures for Mobile Benchmark

REFERENCES

[1] [n. d.]. ([n. d.]). http://www.nanoc-project.eu/

[2] [n. d.]. ([n. d.]). https://www.eda.ei.tum.de/forschung/electronic-system-level/

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (March 2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[4] J. W. d. Mesquita, M. O. d. Cruz, M. M. Pereira, and M. E. Kreutz. 2016. Design Space Exploration Using UTNoCs and Genetic Algorithm. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. 198–202. https://doi.org/10.1109/SBESC.2016.038

[5] Indraneel Das and J. E. Dennis. 1998. Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal on Optimization* 8, 3 (1998), 631–657. https://doi.org/10.1137/S1052623496307510 arXiv:https://doi.org/10.1137/S1052623496307510

[6] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty. 2015. Optimizing 3D NoC design for energy efficiency: A machine learning approach. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 705–712. https://doi.org/10.1109/ICCAD.2015.7372639

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (Apr 2002), 182–197. https://doi.org/10.1109/4235.996017

[8] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg (Eds.). 1999. *Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages, and Tools*. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[9] Jingcao Hu and R. Marculescu. 2005. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 4 (April 2005), 551–562. https://doi.org/10.1109/TCAD.2005.844106

[10] Yuanfang Hu, Yi Zhu, Hongyu Chen, R. Graham, and Chung-Kuan Cheng. 2006. Communication latency aware low power NoC synthesis. In *2006 43rd ACM/IEEE Design Automation Conference*. 574–579. https://doi.org/10.1109/DAC.2006.229293

[11] A. B. Kahng, B. Lin, and S. Nath. 2012. Explicit modeling of control and data for improved NoC router estimation. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. 392–397. https://doi.org/10.1145/2228360.2228430

[12] G. N. Khan and A. Tino. 2012. Synthesis of NoC Interconnects for Custom MPSoC Architectures. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. 75–82. https://doi.org/10.1109/NOCS.2012.16

[13] Levente Kocsis and Csaba SzepesvÂąri. 2006. Bandit based Monte-Carlo Planning. In *In: ECML-06. Number 4212 in LNCS*. Springer, 282–293.

[14] N. Nikitin, J. de San Pedro, and J. Cortadella. 2013. Architectural Exploration of Large-Scale Hierarchical Chip Multiprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 10 (Oct 2013), 1569–1582. https://doi.org/10.1109/TCAD.2013.2272539

[15] M. Tagel, P. Ellervee, and G. Jervan. 2010. Design space exploration and optimisation for NoC-based timing sensitive systems. In *2010 12th Biennial Baltic Electronics Conference*. 177–180. https://doi.org/10.1109/BEC.2010.5631145

[16] Vladimir Todorov, Daniel Mueller-Gritschneder, Helmut Reinig, and Ulf Schlichtmann. 2014. Deterministic Synthesis of Hybrid Application-Specific Network-on-Chip Topologies. *Transaction on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 33, 10 (Oct 2014), 1503–1516.

[17] Fatemeh Vardi, Ahmad Khadem-Zadeh, and Midia Reshadi. 2017. A heuristic clustering approach to use case-aware application-specific network-on-chip synthesis. *The Journal of Supercomputing* 73, 5 (01 May 2017), 2098–2129. https://doi.org/10.1007/s11227-016-1905-6

[18] S. Wallentowitz, A. Lankes, A. Zaib, T. Wild, and A. Herkersdorf. 2012. A framework for Open Tiled Manycore System-On-Chip. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. 535–538. https://doi.org/10.1109/FPL.2012.6339273