# Design Space Exploration of 3D Network-on-Chip: A Sensitivity-based Optimization Approach

DONGJIN LEE, SOURAV DAS, DAE HYUN KIM, JANARDHAN RAO DOPPA, and PARTHA PRATIM PANDE, Washington State University

High-performance and energy-efficient Network-on-Chip (NoC) architecture is one of the crucial components of the manycore processing platforms. A very promising NoC architecture recently proposed in the literature is the three-dimensional small-world NoC (3D SWNoC). Due to short vertical links in 3D integration and the robustness of small-world networks, the 3D SWNoC architecture outperforms its other 3D counterparts. However, the performance of 3D SWNoC is highly dependent on the placement of the links and associated routers. In this article, we propose a sensitivity-based link placement algorithm (SEN) to optimize the performance of 3D SWNoC. The sensitivity of a link in a NoC measures the importance of the link. The SEN algorithm optimizes the performance of 3D SWNoC by calculating the sensitivities of all the links in the NoC and removing the least important link repeatedly. We compare the performance of SEN algorithm with simulated annealing- (SA) and recently proposed machine-learning-based (ML) optimization algorithm. The optimized 3D SWNoC obtained by the proposed SEN algorithm achieves, on average, 11.5% and 13.6% lower latency and 18.4% and 21.7% lower energy-delay product than those optimized by the SA and ML algorithms respectively. In addition, the SEN algorithm is 26 to 33 times faster than the SA algorithm for the optimization of 64-, 128-, and 256-core 3D SWNoC designs. The performance gain provided by the SEN-, SA-, and ML-based methods also depend on the characteristics of the benchmarks under consideration. If the traffic pattern generated by a benchmark does not have enough variation, then the ML-based method does not have adequate opportunity to optimize the network. However, we find that ML-based methodology has faster convergence time than SEN and SA for bigger systems. The ML-based optimization algorithm is almost 4 and 97 times faster than the SEN- and SA-based algorithm for a system with 256 cores.

CCS Concepts: • **Networks → Network on chip**; • **Computer systems organization → Interconnection architectures**; • **Hardware → On-chip resource management**; **Interconnect power issues**;

Additional Key Words and Phrases: 3D NoC, link placement optimization, small-world network, sensitivity

**ACM Reference format:**
Dongjin Lee, Sourav Das, Dae Hyun Kim, Janardhan Rao Doppa, and Partha Pratim Pande. 2018. Design Space Exploration of 3D Network-on-Chip: A Sensitivity-based Optimization Approach. *J. Emerg. Technol. Comput. Syst.* 14, 3, Article 32 (October 2018), 26 pages.
https://doi.org/10.1145/3197567

This work is supported by the US National Science Foundation (NSF) grants CNS-1564014 and CCF 1514269 and USA Army Research Office grant W911NF-17-1-0485.
Authors' addresses: D. Lee, S. Das, D. H. Kim, J. R. Doppa, and P. P. Pande, School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99163; emails: {dlee2, sdas, daehyun, jana, partha}@eecs.wsu.edu.

https://doi.org/10.1145/3197567

## 1 INTRODUCTION

A three-dimensional integrated circuit (3D IC) enables integration of multiple layers with smaller footprint, reduced form factor, and overall high packing density, which improves the performance of chip drastically compared to its 2D counterparts. Hence, 3D integration opens the possibility for designing highly integrated manycore chips (Pavlidis and Friedman 2009; Topol et al. 2006; Xie et al. 2006). The performance of any manycore chip largely depends on the network-on-Chip (NoC), which is the communication backbone for inter-core data exchange (Benini and Micheli 2002; Kumar et al. 2002). In this respect, 3D NoC is an emerging paradigm that achieves better performance, network latency, and lower energy dissipation compared to its 2D counterparts (Pavlidis and Friedman 2009; Feero and Pande 2008; Matsutani et al. 2008). The performance of the 3D NoC architecture also depends on the overall network connectivity and the placement of the network elements (Teuscher 2007). In this context, a small-world (SW) network-enabled 3D NoC architecture (3D SWNoC) achieves lower latency and energy compared to other state-of-the-art 3D NoCs (Das et al. 2015 and 2017). In the 3D SWNoC architecture, the vertical links act as the long-range shortcuts to induce "small-worldness" resulting in reduction of average hop count, lowering the latency and energy dissipation (Das et al. 2017).

The achievable performance benefit of a 3D SWNoC depends on the placement of both the planar and vertical links. Traditionally, simulated annealing (SA)-based optimization technique is used in various domains of VLSI and electronic design automation (EDA), including floorplanning, routing, partitioning, power-ground placement, clock-tree optimization, NoC link placements, and so on (Healy et al. 2007; Liu and Chang 2007). However, the main bottleneck of SA-based optimization methodology is that depending on the system size and type of the problem, SA may require a long time to converge to the final solution. Recently, machine-learning (ML)-based optimization algorithms have also been exploited to enable the design of low-power and high-performance 3D NoC architectures (Das et al. 2015). From the 3D NoC design perspective, the ML technique intelligently searches the design space to optimize the placement of both planar and vertical links, and converges to the same quality solution in significantly lower time when compared to the SA. Complementing the traditional SA-based and the emerging ML-inspired design optimization methodologies, in this article, we propose a sensitivity-based optimization algorithm to solve the link placement problem for 3D NoC architectures. To do this, we define the sensitivity parameter, which represents the effectiveness of each link in the 3D NoC design space. The key idea behind our algorithm is to iteratively compute a sensitivity value of each link (cost varies when the link is removed) and remove the least important link. Although the sensitivity-based algorithm is exploited only for the link placements in 3D SWNoC architecture, the algorithm is generic and is equally applicable for any other 2D or 3D NoC configurations as well.

In this article, our aim is to undertake a comparative performance evaluation for the sensitivity-based NoC link optimization with respect to the SA- and the ML-based methodologies. We consider the 3D SWNoC as the testbed for this performance evaluation. However, the outlined algorithm can be exploited for any other NoC architectures as well.

The rest of article is organized as follows. Section 2 discusses the previous works done in the field of network optimization and NoC performance improvements. The details of the design space and the NoC optimization problem with the small-world network and manycore chip design are elaborated in Section 3. In addition, the NoC optimization algorithms and their associated implementation details are discussed in the rest of Section 3. We perform an exhaustive performance evaluation in Section 4. Finally, Section 5 summarizes everything and outlines the directions for future work.

## 2 RELATED WORK

In this section, we discuss the details of previous works on the design and optimization of NoC architectures enabled by 3D integration.

To improve the performance of manycore chips, various 3D NoC architectures have been proposed in the literature. Among all of them, the MESH-based designs are the most popular and widely used (Kim et al. 2007; Loi et al. 2011; Marcon et al. 2014). A 3D MESH NoC is an extended version of the conventional 2D MESH architecture, where the cores and routers are placed in a regular grid pattern. A 3D NoC has much higher connectivity and reduced wire length when compared to its 2D counterparts. However, MESH-based NoCs suffer from high latency and energy dissipation due to their inherent multi-hop communications. To take advantage of the availability of reduced vertical distance inherent in 3D integration, a 3D Dimensionally Decomposed (DimDe) NoC router architecture (Kim et al. 2007) was developed that reduces the total energy dissipation, but latency was not minimized. A hybrid NoC-bus-based architecture was proposed in Li et al. (2006) by using central bus arbiter and Dynamic Time Division Multiple Access (dTDMA) technique for bus access in the vertical dimension to reduce the network latency. However, bus-based designs suffer from long latency in the presence of higher traffic injection applications and also dissipate higher energy due to relatively higher capacitance values when compared to MESH-based architectures (Feero and Pande 2008).

In this context, NoCs incorporating small-world connectivity can perform significantly better than locally interconnected MESH-like networks (Ogras et al. 2006), yet they require far fewer resources than a fully connected system. Hence, in this work, we consider small-world-based 3D NoC (3D SWNoC) as the testbed for performing the link placement optimization (Das et al. 2015).

To exploit the advantages of 3D integration, several works have addressed the synthesis of application-specific NoC architectures (Seiculescu et al. 2009; Zhou et al. 2012). The Sunfloor 3D was proposed for developing application-specific 3D NoCs (Seiculescu et al. 2009). The design and synthesis of application-specific 3D NoC architectures were also investigated in Wang and Dong (2009). Later, a more general-purpose 3D NoC was proposed in Xu et al. (2009) using an ILP-based algorithm to insert long-range links to develop low diameter and low radix architecture. However, the reduction in energy dissipation was found to be limited.

Recently, two irregular 3D NoC architectures, viz., *mrrm* and *rrrr*, have been proposed in the literature (Matsutani et al. 2014). In *mrrm*, there are two layers of MESH-based interconnection and two layers of random connectivity. In *rrrr*, all four layers follow random connection. In this case, both *mrrm* and *rrrr* architectures were developed following a random link placement algorithm. It was found that both architectures can outperform 3D MESH in terms of network latency and energy dissipation. We can expect that by adopting an efficient link placement algorithm along with more promising NoC architectural design space will further improve the performance of 3D NoCs when compared to existing architectures.

In this work, we focus on designing an optimized link placement strategy applicable for any kind of NoC design. As a case study, we consider the NoC design space of 3D SWNoC for achieving better performance from the architectural perspective. We propose a sensitivity-based link placement algorithm to enhance the performance of 3D SWNoC and the convergence time of the algorithm. In addition, we compare the performance of 3D SWNoCs optimized by the sensitivity-based, the simulated annealing-based, and the machine-learning-based algorithms.

## 3 DESIGN SPACE AND LINK PLACEMENT OPTIMIZATION OF 3D SWNOC

A small-world (SW) network lies in between the regularly connected MESH and completely random networks. The SW network consists of a large number of short-range links facilitating the

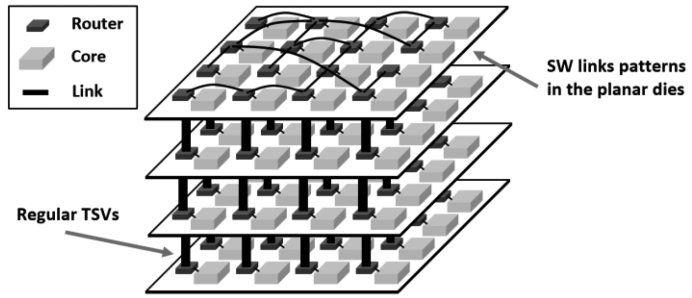


Fig. 1. Conceptual view of small-world (SW) connection-based 3D SWNoC architecture. The horizontal plane (each die) has SW-based connectivity where the regular TSVs enable long-range short-cuts.

local communications while a few long-range shortcuts establishing direct links between distant nodes helps in improving the long-distance communication. The diameter of a SW network is bounded logarithmically with its network size, i.e., diameter $\propto \log(N)$, where $N$ is the number of nodes in the network (Nguyen and Martel 2005). In addition, a SW network is also particularly resilient against any kind of link failure (Teuscher 2007). Both of these properties of the SW network make it attractive to be employed for designing efficient and robust on-chip communication backbone.

To design the small-world-enabled 3D NoC, we follow the power-law-based connectivity pattern, where the probability of connecting two nodes varies exponentially with distance between them, i.e., $p(r) \propto r^{-\alpha}$. Here, $p(r)$ is the probability of connecting two nodes separated by link length $r$, and $\alpha$ is the connectivity parameter that dictates the amount of "small-worldness" introduced in the network by following this connectivity rule. For the 3D NoC, the additional third dimension (compared to conventional 2D NoCs) allows us to place the cores and associated routers in the 3D space in such a way that the planar physically long links (of 2D network) are placed along the vertical dimension, which facilitates the small-world connectivity.

As explained earlier, a small-world-network-inspired 3D NoC was shown to outperform other existing 3D NoCs in terms of network latency, energy consumption, and reliability improvement (Das et al. 2017). However, the performance gain of 3D SWNoC depends critically on the placement of both horizontal and vertical links. Hence, we present the salient features of various link placement strategies investigated in this work. We principally focus on two broad optimization algorithms, viz., the sensitivity-based and the ML-inspired methodologies. We will compare and contrast the performance of these two algorithms along with the well-known simulated annealing algorithm.

### 3.1 3D SWNoC Architecture

Figure 1 shows an example of 3D SW network enabled NoC (3D SWNoC) architecture with four planar dies and 16 cores arranged in a grid-pattern on each die. Through-silicon vias (TSVs) enable the vertical communication links between the dies and act as long-range shortcuts. In addition, the interconnection patterns in the 3D SWNoC follow the power-law-based connectivity: $P_r = \gamma \cdot r^{-\alpha}$, where $r$ is the length of the link, $\alpha$ is a parameter to determine the small-worldness, and $\gamma$ is a normalization factor (Teuscher 2007; Wettin et al. 2014), and $P_r$ is the number of length-$r$ links in the network. $L$ is a class (a collection of sets) of the link distributions of all the layers, so $L = \{L_1, L_2, \ldots, L_k\}$, where $L_k$ is the link distribution in layer $k$. $L_k$ is $\{l_{k,1}, l_{k,2}, \ldots\}$ where $l_{k,r}$ is the number of length-$r$ links in layer $k$. We assume that a vertical link is inserted between two vertically adjacent routers. $l_v$ is the total number of vertical links, which count as length-1. If $P_1$ is

the number of length-1 links, then $P_1 = \sum_k l_{k,1} + l_v$, and $P_r$ is the number of length-$r$ planar links, then $P_r = \sum_k l_{k,r}$ $(r > 1)$. The planar links follow this distribution of links: $\sum_k L_k = \{l_1, l_2, l_3 \ldots\}$, where $l_{(r=1)} = (\gamma \cdot r^{-\alpha} - l_v)$ and $l_{(r>1)} = \gamma \cdot r^{-\alpha}$. For example, the 3D MESH with four layers uses 48 vertical links and 96 planar links, so our 3D SWNoC architectures also use the same number of vertical and planar links. We assume that all the layers have the same link distribution, so $L_1 = L_2 = L_3 = L_4$ in our designs. Hence, there are total 96 planar links, so $\sum_i L_{k,i} = 24$ for each layer $k$ and $l_v$ is 48. For example, if $\alpha$ is 2.4, then $L_k$ becomes $\{16, 5, 2, 1\}$ for each die. Depending on this power-law-based link distribution, we can design 3D SWNoC architecture by following the physical NoC design constraints. We refer to any instance of 3D NoC configuration as a "*state,*" which is a discrete candidate solution in the NoC design space.

### 3.2 General Problem Formulation

In this section, we formally define the NoC link placement optimization problem targeted in this work. We formulate the NoC design as a combinatorial optimization problem over a discrete design space, where the placement of the cores, routers, and the horizontal and vertical links defines the overall network configuration.

*3.2.1 Objective Function: Communication Path Length (O).* To optimize the NoC architecture, we define a unified objective function, $O$, that computes the total communication cost based on the communication path length. It is defined as the product of the hop count, communication frequency, and physical link length between two nodes summed over all possible source and destination pairs in the network, i.e.,

$$O = \sum_{i=1}^{N} \sum_{j=1}^{N} (m \cdot h_{ij} + d_{ij}) \cdot f_{ij}, \tag{1}$$

where $N$ is the system size (total number of cores), $m$ is the number of router (intra-node routing) stages, and $h_{ij}$ and $d_{ij}$ are the hop count and the communication distance between $i$th and $j$th nodes, respectively. $f_{ij}$ is the communication frequency between $i$th and $j$th nodes and captures the characteristics of each application. Optimization of the communication cost $O$ reduces the average hop count and the communication distance for any possible source-destination pairs. As a result, the optimized network improves the latency and energy consumption of the NoC for a given application.

*3.2.2 Problem Statement.* We consider a NoC design space consisting of a set of nodes, $S = \{c_1, c_2, \ldots, c_N\}$, where $N$ is the total number of cores and each core is connected to a nearby router. The locations of the routers (and associated cores) are denoted by $(x_i, y_i, z_i)$, where $(1 \leq x_i, y_i, z_i \leq 4,\ x_i, y_i, z_i \in \mathbb{N})$. For any application executed on this manycore chip, the internode communication frequencies are denoted by the set $f = \{f_{ij} | 1 \leq i, j \leq N, i \neq j\}$, where $f_{ij}$ is the communication frequency between $c_i$ and $c_j$. The NoC architecture is enabled by a small-world network following the power-law-based connectivity (Teuscher 2007; Wettin el at. 2014) and a fixed connectivity parameter $\alpha$ as described in the previous section. As a result, for a particular value of $\alpha$, the link length distribution is fixed and the set of target link distributions is denoted by $L = \{L_1, L_2, \ldots, L_T\}$, where $T$ is the total number of layers in the NoC and $L_k$ is the link distribution in layer $k$.

During the NoC design and optimization process, the communication cost of the network $O$ indicates the quality of the network for a given set of performance metrics. Our goal is to develop a link placement algorithm that explores the given SWNoC design to find the NoC with minimum communication cost. As mentioned above, we consider three different optimization algorithms, namely

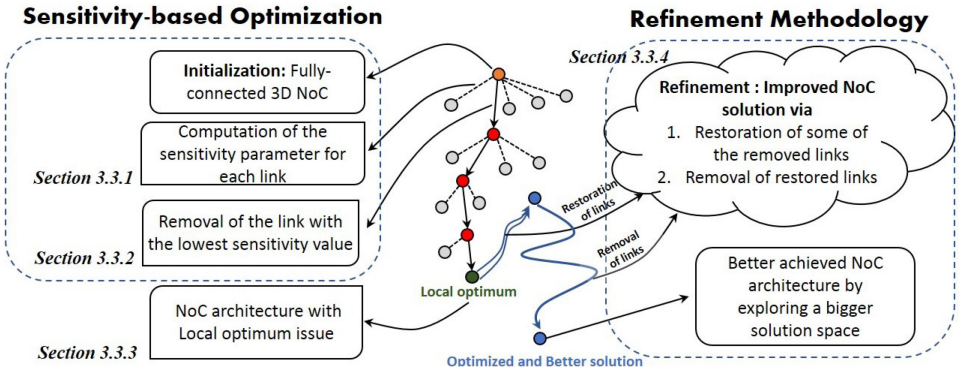


Fig. 2. Illustration of the overall design flow of the sensitivity-based NoC optimization algorithm. The SEN-based algorithm starts at the fully connected NoC configuration and defines the sensitivity parameter for each link of the 3D NoC. To explore the neighborhood NoC states, the SEN-based algorithm removes the least important link repeatedly. It is possible that during the optimization process, the algorithm may get stuck at local optimum of the cost function. To solve this local optimum problem, the network refinement methodology is employed.

the sensitivity-based algorithm, the simulated annealing, and the machine-learning approach for the placement of horizontal and vertical links in a 3D SWNoC. Our target is to compare and contrast their relative performances. The performance of all the optimization algorithms considered in this work is evaluated from two different perspectives, viz., the quality of the optimized SWNoC architecture in terms of a given set of performance metrics (network latency, and energy-delay-product (EDP)) and convergence times of the algorithms. We use the same number of vertical and horizontal links as the 3D MESH architecture for the optimized 3D SWNoC as given constraints. In addition, just like the MESH architecture, we evenly assign the same number of planar links to the four layers, so each layer has equal number of links for the same system sizes across different NoC configurations.

### 3.3 Sensitivity-based Link Placement

The sensitivity-based link placement algorithm defines the effectiveness of each link in the NoC architecture for a given link distribution and employs it to find the best network configuration. In what follows, we define the sensitivity parameter of a link, discuss the details of exploring the neighborhood states from a candidate NoC design state, and explain how the final NoC architecture is selected through a sequential decision-making process. Figure 2 shows the overall design flow of the proposed sensitivity-based NoC optimization algorithm and outlines the key processes at each design step.

*3.3.1 Definition of Sensitivity Parameter,* $\boldsymbol{s(z_{ij})}$. The sensitivity parameter, $s(z_{ij})$, of a link $z_{ij}$ connecting two nodes $c_i$ and $c_j$ is defined as the difference between the communication costs before and after removing the link. If any link in a 3D NoC architecture is not actively used for inter-node traffic exchanges (router-to-router communication), then the communication costs before and after removing the link do not change and, hence, $s(z_{ij})$ of that link is zero. If the internode communication frequency between two nodes, $i$ and $j$, $f_{ij}$, is positive (this is a general case), then the value of sensitivity parameter for the link $z_{ij}$ connecting these two nodes, $s(z_{ij})$, is greater than zero. This is because the link $z_{ij}$ is the only link that directly connects the nodes $c_i$ and $c_j$ and, hence, $z_{ij}$ is the only shortest path between these nodes. So, removal of link $z_{ij}$ increases the hop

count between $c_i$ and $c_j$, and thereby increasing the communication cost. The sensitivity $s(z_{ij})$ of link $z_{ij}$ is mathematically expressed as follows:

$$s(z_{ij}) = O_{ij,a} - O_{ij,b}, \tag{2}$$

where $O_{ij,a}$ and $O_{ij,b}$ are the communication costs after and before removing $z_{ij}$, respectively. If the sensitivity of a link is low, then the NoC is less sensitive to the removal of the link, i.e., the communication cost does not change significantly even if the link is removed. In this case, we can remove the link without degrading the performance of the NoC. If the sensitivity of a link is high, however, then the communication cost of the NoC increases significantly if the link is removed, so the link should not be removed.

*3.3.2 Outline of the Sensitivity-based NoC Optimization Algorithm.* The sensitivity-based link placement algorithm consists of three steps, initialization, exploration of neighborhood states, and identification of the terminal state with an optimized NoC architecture. In the following subsections, we discuss the details of each step and the overall sensitivity-based optimization algorithm.

*3.3.2.1 Initialization: Fully-connected 3D NoC.* The sensitivity-based optimization algorithm starts with *a fully connected 3D NoC* in which each router in a layer is connected to all the other routers in the same layer as well as the vertically adjacent routers. Since a fully connected 3D NoC guarantees the connectivity (there exists at least one path between any pair of routers), we can compute the cost of the fully connected 3D NoC using Equation (1).

*3.3.2.2 Exploration of Neighborhood States.* The initialization step generates a fully-connected NoC architecture, which has the lowest communication cost. After the initialization step, we sequentially remove some of the links in the NoC to find an optimal link placement following the power law. We use the sensitivity values to select the links to be removed. We explain the details of the link removal process below.

**ALGORITHM 1:** Constraint Function

**Input**: $N$ = # cores (64, 128, 256), $T = \#layers$ (4)
**Output**: A set of non-removable links ($V$)
//$G_{p_{max}}$ is a set of links having max. connection count
**for** $i$ = 1 *to* $N$ **do** //Check connectivity
  **for** $j$ = 1 *to* $N$ **do**
    **if** $z_{ij}$ is necessary for connectivity
      Add $z_{ij}$ *to* $V$
    **end if**
  **end for**
**end for**
  **for** $r$ =1 *to* Max. link length **do**
    **for** $k$ =1 *to* $T$ **do**
      **if** $l'_{k,r} = l_{k,r}$ //Power-law distribution
        Add length-$r$ of $z_{ij}$ in layer $k$ to $V$
      **end if**
    **end for**
  **end for**
  **if** $p_{max} > s_{max}$ //Max. number of links per router
    Add all links to $V$ except $G_{p_{max}}$
  **end if**

**ALGORITHM 2:** Link Removal Function

**Input**: $N$ = # cores (64, 128, 256)
**Output**: Removing a least-critical link
Constraint()[a] //from Algorithm 1
**for** $i$ = 1 *to* $N$ **do**
  **for** $j$ = 1 *to* $N$ **do**
    **if** $z_{ij} \notin V$, **then**
      $S_{new} = s(z_{ij})$
      **if** $S_{new} < S_{min}$, **then**
        $S_{min} = S_{new}$
        $i_{min} = i$
        $j_{min} = j$
      **end if**
    **end if**
  **end for**
**end for**
remove a link $z_{i_{min}, j_{min}}$

[a]The Constraint() function has same NoC size ($N$) and number of layers ($T = 4$ in this case) as the link removal function, and hence, these input parameters are not passed in each function call.

**ALGORITHM 3:** Link Insertion Function (places one link at a time)

**Input**: $N$ = # cores (64, 128, 256)
**Output**: Adding a most-critical link

```
for i = 1 to N do
  for j = 1 to N do
    if z_ij ∉ Z, then
      S_new = | s(z_ij)|
      if S_new > S_max, then
        S_max = S_new
        i_max = i
        j_max = j
      end if
    end if
  end for
end for
add a link z_{i_max, j_max}
```

**ALGORITHM 4:** Sensitivity-based Network Optimization Algorithm

**Input**: $N$ = # cores (64, 128, 256), $T$ = # *layers* (4), $R$=3, $L$ = # links (144, 304, 640)
**Output**: Optimized 3D SWNoC

```
Initialization() //Section 3.3.2.1
while Total number of link > L do
  LinkRemoval() //Algorithm 2
  if p_max ≤ s_max
    do
      for 1 to R then
        LinkInsertion() //Algorithm 3
      end for
      for 1 to R then
        LinkRemoval() //Algorithm 2
      end for
    while !(same links inserted and removed)
  end if
end while
```

***Link Constraint*:** After the network initialization, we introduce the link-constraint step. In this link-constraint step, we find a set of non-removable links ($V$) and keep the list of possible removable links, which do not violate the pre-specified constraints in the next link removal step. There are three constraints that we should consider when we select a link to be removed from the current NoC. First, removing a link should not break the network connectivity, i.e., there should exist at least one path between any pair of routers. If there is a necessary link ($z_{ij}$) for overall connectivity, then we add $z_{ij}$ to a set of non-removable links ($V$). Second, a final 3D SWNoC architecture should satisfy the power-law-based link distribution $L_k = \{l_{k,1}, l_{k,2}, \ldots\}$. Thus, if the number of length-$r$ links in layer $k$ ($l'_{k,r}$) is equal to a given constraint $l_{k,r}$, then we add links of length-$r$ ($z_{ij}$) in layer $k$ to $V$. Third, there is also a restriction on the maximum number of connection $s_{max}$ that each router can have (Kim et al. 2016). To satisfy this constraint, we introduce the term connection count $d(z_{ij})$ of link $z_{ij}$. It is defined as the maximum of $(d(c_i), d(c_j))$ where $d(c_i)$ is the number of ports connected to node $c_i$. When we check the constraint of a link, we first compute the connection count of each link and group the links into $G_1, G_2$, and so on, where $G_p$ is a set of links whose connection count is $p$. Let the maximum connection count be $p_{max}$. If $p_{max}$ is greater than $s_{max}$, i.e., some of the remaining links violate the maximum connection count constraint, then we add all links to $V$ except $G_{p_{max}}$ for the link removal step. If any of the links in $G_{p_{max}}$ is not removable due to the connectivity or link distribution constraint, then we move on to the next set $G_{p_{max}-1}$, check the constraints, and continue in the same way. However, if $p_{max}$ is less than or equal to $s_{max}$, i.e., none of the remaining links violates the maximum connection count constraint, then we do not add links to $V$. Algorithm 1 shows this link constraint enforcement process.

***Link Removal*:** We compute the sensitivity values of all links, which are not included in a set of non-removable links ($V$). We remove a link with the lowest sensitivity value. Since removing a link increases the communication cost, we select the link having the lowest sensitivity value, because removal of this particular link causes the least performance degradation. The removal process essentially explores the neighborhood states of the current NoC configuration to find the best architecture with one fewer link than the current one. Algorithm 2 presents this link removal process.

***Updating the Current State*:** Removing a link changes the sensitivity values of other links in the NoC. Thus, we should re-compute the sensitivity values of all links whenever we remove a link. However, re-computing the sensitivities of all the remaining links is expensive, because the computation of the sensitivity value of a link involves shortest-path computation. Therefore, we update the sensitivity values of only those links that along with the removed links are part of the shortest path between any pair of nodes in the network before removing that particular link.

*3.3.2.3 Terminal State: Optimized NoC Architecture.* Iterating between the link constraint and removal steps in the NoC design generates a final NoC architecture that satisfies the target link distribution, max-connection-per-router, and the network connectivity constraints. The final NoC architecture is formed with the links having high-sensitivity values. Consequently, the communication cost of this architecture is low, and we find the optimized NoC architecture. A drawback of this algorithm is that some of the links removed in early constraint/removal steps might be critical links, which can potentially improve the communication cost if they were re-inserted into the NoC in later steps. Thus, we complement the above algorithm in the next section to improve the quality of the final NoC architecture.

*3.3.3 Optimized NoC Architecture: Local Optimum Issue.* As described earlier, the NoC link placement problem is an instance of combinatorial optimization. Thus, if the design space is not explored properly, then it can get stuck in a local minimum. For the proposed sensitivity-based optimization, starting from the initial fully-connected NoC, the algorithm repeatedly removes a link with the lowest sensitivity. However, this does not guarantee finding a global optimal solution, because the sensitivity computation is based on the current NoC architecture. The lowest-sensitivity link at a removal step is the least useful link at this step, however, it might become more useful than other links at a later removal step. Thus, it is necessary to restore some of the links removed in the earlier steps to find a better NoC architecture as the optimized solution.

*3.3.4 Methodology for Solving Local Optimum Problem.* To solve the local optima issues of the NoC optimization methodology, we propose an approach, which is based on the domain knowledge of NoC architectures and experimental observations. We refer to this approach as the network refinement procedure.

*3.3.4.1 Network Refinement.* A potential problem in the link removal step is that there is no mechanism to restore any removed links in later steps. However, restoring some of the removed links might improve the quality of the solution. The proposed network refinement approach restores a certain percentage of the links removed in earlier steps and recalculates the effects of all links in the NoC architecture. This step explores a bigger solution space by recalculating the effect of network traffic on each link, and subsequently, tries to find better solutions. However, the computational complexity of network refinement after each link constraint/removal step is high. Hence, we perform the refinement algorithm by adding only a small portion of the total number of links.

In the refinement step, we repeat adding $R$ number of removed links, and removing $R$ number of links again until no more improvement in the communication cost is achieved. To add $R$ numbers of links, we compute the sensitivity $|s(z_{ij})|$ of each non-existing (already removed) link $z_{ij}$ by computing the difference between the costs after and before adding the link into the current network. As long as $f_{ij}$ is not zero, $s(z_{ij})$ is always negative, because $z_{ij}$ is the only shortest path between the nodes $c_i$ and $c_j$, and hence, adding $z_{ij}$ decreases the hop count between them. Once we find the link with the highest sensitivity value, we add it to the network, recalculate the sensitivity values of the non-existing links, and repeat this process $R$ times to add $R$ links. Then, we remove $R$ from the existing links using the link constraint and removal process described in Section 3.3.2.2.

For the refinement methodology to be effective, we need to determine the appropriate value of $R$. Increasing $R$ helps the algorithm in exploring a larger solution space for the best solution, but spends more time to search for the solution. To determine the value of $R$, we performed exhaustive search for various benchmarks and found that $R = 3$ was the best value in terms of the performance and the runtime trade-off. Algorithm 4 shows the sensitivity-based algorithm with the refinement.

*3.3.4.2 Complexity Analysis of the Refinement Method.* The refinement algorithm backtracks some parts of the solution space by reconnecting some links and exploring best solutions to avoid the local optima issues. To evaluate the additional cost of running the refinement algorithm for an improved NoC optimization, we analyze the complexity of the algorithm as follows.

The sensitivity-based algorithm repeats (1) finding removable links, (2) removing the lowest-sensitivity link, and (3) updating the sensitivity values of all the links existing in the 3D NoC. The complexity of finding the lowest-sensitivity link is $O(k)$ if the 3D NoC has $k$ links and the sensitivity of each link is known. The complexity of removing a link is $O(1)$, because it is just a pointer manipulation in the source code. Updating the sensitivity values of all the links consists of (1) finding the set $B$ of all the core pairs affected by the link removal, (2) finding the shortest path between each core pair $b \in B$, and (3) re-computing the sensitivities of all the links. The complexity of finding all the core pairs affected by removing a link is theoretically $O(k)$, because the number of core pairs is proportional to the number of links, and we have an updated data structure that stores information of which core pair uses which link. In general, however, only a few core pairs are affected by removing a link, because many core pairs that are far away from the removed link do not use that link in their shortest paths. Thus, a constant number of core pairs are affected by removing a particular link. The complexity of finding the shortest path between two cores is also $O(k)$, because we use the Dijkstra algorithm for the shortest path finding. Therefore, the complexity of finding the shortest paths for all the core pairs in $B$ is $O(k)$.

The complexity of re-computing the sensitivity of an existing link is $O(k)$ for the following reason. To re-compute the sensitivity of a link, we remove the link and compute the total communication cost. For the same reason described above, computing $\sum\sum(m \cdot h_{ij} + d_{ij}) \cdot f_{ij}$ requires re-computing $h_{ij}$ and $d_{ij}$, which takes $O(k)$ time, only for the core pairs affected by removing the link. Thus, the complexity of re-computing the sensitivity of a link is $O(k)$. In addition, when we remove a target link, the sensitivities of many other links are not affected by removing the target link. Thus, re-computing the sensitivities of all the links actually requires re-computing the sensitivities of only a few links. Thus, the complexity of re-computing the sensitivities of all the links after removing a link is practically $O(k)$. The sensitivity-based optimization algorithm repeats the link removal process $(L - t)$ times where $L$ is the total number of links of the fully-connected 3D NoC for given core locations and $t$ is the number of edges in the 3D NoC architecture ($t \ll L$). Therefore, the overall complexity of the sensitivity-based optimization algorithm is $O(L^2)$.

If the refinement step is used, then we add and remove $R$ number of links for the refinement procedure and repeat it until it reaches a steady state in which the same links are added and removed. Adding a link requires computing the sensitivity of each non-existing (removed) link, so the complexity of adding and removing $R$ links is practically $O(R \cdot L)$. The number of repetitions of adding and removing $R$ links is constant in general. Thus, the complexity of each refinement step is $O(R \cdot L)$. From the implementation perspective, $R$ is significantly smaller than the total number of links (as mentioned above, we employed $R = 3$ for our experiments). Thus, the complexity of the overall sensitivity-based optimization algorithm with refinement is also $O(L^2)$.

*3.3.5 Runtime Improvement of the Baseline Sensitivity-based Optimization Algorithm.* The sensitivity-based NoC optimization algorithm finds removable links and removes a least-critical link, and updates the sensitivity values of all the remaining links affected by the removal step. At some

stages of the optimization procedure, however, the changes of the sensitivity values of the links in the sensitivity update process are negligibly small. This is because the communication frequency between some pairs of nodes is low ($f_{ij} \approx 0$), so the direct links placed between these nodes carry a very small amount of traffic. Hence, they affect the quality of NoC optimization minimally. This phenomenon occurs regardless of the NoC system size and application characteristics. Therefore, we can repeat the link constraint and removal steps several times without updating the sensitivity values. This approach helps in reducing the optimization runtime significantly with negligible loss in accuracy. Especially, applying this speed-up technique at the beginning of the optimization process significantly reduces the runtime and does not degrade the quality of the final solution. Depending on applications in our simulation results, we were able to remove significant percentage of the total links at the beginning of the optimization process without updating the sensitivity values as elaborated later in Section 4.3.1.2. We also adopt the network refinement methodology in later stages of the search space exploration. As a result of the combined effect of both one-time removal of a large number of links at the beginning of the optimization procedure and the network refinement step, we can significantly reduce the runtime to find the same quality NoC architecture that is found without the speed-up technique.

### 3.4 Machine-Learning-based Optimization

In this section, we discuss one particular machine-learning approach that has already been adopted in the field of NoC optimization (Boyan and Moore 2000; Das et al. 2015). This approach adopts an algorithm called STAGE that was originally proposed to improve the speed of solving combinatorial optimization using local search methods (Boyan and Moore 2000).

To employ the STAGE approach for NoC optimization, each candidate NoC design in the design space is represented with a set of features that relate to the optimization problem. In this work, we define in total twenty features that essentially represents the average hop count ($h_{ij}$), weighted communication ($f_{ij} \cdot h_{ij}$), and clustering co-efficient ($C_c$) (Humphries and Gurney 2008) properties of the network.

The STAGE algorithm has three main components: (i) Base search, (ii) Meta search, and (iii) a learned Evaluation function. Below, we briefly describe these three components followed by a high-level description of the STAGE algorithm.

*3.4.1 Base Search.* The base search is, in general, a local search algorithm, e.g., greedy search or simulated annealing. The base search tries to optimize the main objective function of the problem (communication cost, $O$, in this case) from multiple starting solutions (candidate NoC configurations). However, the performance of local search procedures depends critically on the quality of starting solutions as the performance of search depends on the starting NoC configurations.

*3.4.2 Meta Search.* The role of the meta search procedure is to improve the performance of the base search by selecting "*good*" starting solutions. The meta search procedure explores the given NoC design space to find good starting solutions that can potentially lead the base search to (near-)optimal NoC designs. Meta search procedure is guided by a learned evaluation function that can predict the quality of the local optima obtained by performing the base search starting from a given solution.

*3.4.3 Learned Evaluation Function.* For each search trajectory of the base search procedure from a starting solution, we can generate many training examples to learn/improve the evaluation function. For each solution on this search trajectory, we generate one regression example: features of the NoC design as the input and the objective value of the local optima as the output. These regression examples are given to a regression learning algorithm (e.g., regression trees) to learn the

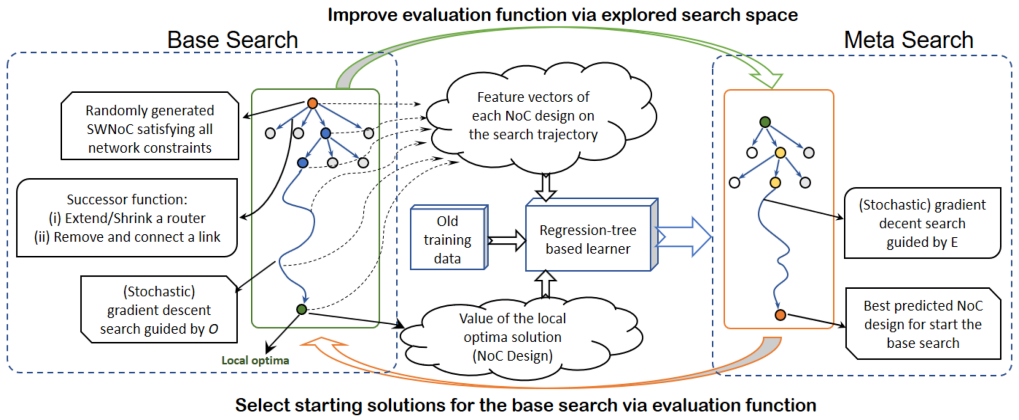


Fig. 3. Overall STAGE-based optimization algorithm for 3D SWNoC link placement. Two searches, viz. the base search and meta search, iterate in between them by seeking guidance from the evaluation function, E, to reach to the optimized NoC architecture quickly.

evaluation function $E$ (Uther and Veloso 1998). The predictive accuracy of the learned evaluation function $E$ improves with more training data.

*3.4.4 Overall Algorithm.* The high-level conceptual idea of the STAGE-based algorithm is shown in Figure 3. The overall STAGE-based algorithm iterates between the base search and the meta search procedure to improve the accuracy of the evaluation function $E$ to be able to quickly find (near-) optimal designs through base search. Initially, $E$ makes random predictions. However, as the iterations progress, we generate more training examples adaptively to correct the errors made by $E$. In each iteration, the STAGE algorithm performs the following steps: (1) meta search guided by the current evaluation function selects a starting solution for base search; (2) base search guided by the original objective function is performed from the selected starting solution; (3) if the prediction of evaluation function $E$ is wrong for the selected starting solution, then we generate regression examples for each solution on the search trajectory to improve $E$; and (4) the aggregate set of regression examples are given to a regression learning algorithm to learn the improved evaluation function.

The time complexity of STAGE algorithm is $O(P(uT_b + uT_b + uT_b + uT_b))$, where $P$ is the number of iterations, $u$ is the number of successors considered at each state, $T_b$ is the length of the search trajectory, and each $O(uT_b)$ corresponds to meta search, base search, training data generation, and regression learning respectively. The simplified complexity of STAGE-based NoC optimization is $O(4PuT_b)$. In general, the value of $P$ is small (<20) and obtained through experimental analysis. The value of $T_b$ depends on the system size ($N$), and total number of links ($L$) present in the NoC configuration.

### 3.5 Simulated Annealing (SA)-based Optimization

In addition to previously mentioned NoC optimization techniques, we also adopt the popular simulated annealing algorithm (SA) (Kirpatrick et al. 1983). In this section, we briefly describe the SA approach to optimize the small-world network-enabled NoC architecture.

Algorithm 5 shows the simulated annealing-based network optimization methodology. In step 1, we first create an initial 3D SWNoC configuration by inserting all the fixed vertical links between two vertically-adjacent routers and random planar links satisfying the NoC design constraints.

These constraints include maintaining the link distribution, and the maximum and minimum connectivity per ports as described in the earlier sections. In steps 2-3, a parameter called temperature (*Temp*) is used to control the number of iterations. $M$ is the number of moves to attempt at each temperature. In step 4, we perturb the solution by randomly choosing a planar link ($z_1$), removing it, and adding a new link ($z_2$) of the same length as the removed link in the same layer. If a solution perturbation violates any of the constraints, then we restore the original architecture by removing the new link ($z_2$) and restoring the original link ($z_1$). Then, in steps 5–8, we calculate $\Delta O$ and determine the acceptance of the new architecture by Boltzmann distribution ($Prob(\Delta O) = e^{-\Delta O/Temp}$) (Kirpatrick et al. 1983). Here, $Prob(\Delta O)$ is the acceptance probability, and $\Delta O$ is the difference between the communication path lengths before and after the link perturbation. The higher values of temperature cause more such moves to be accepted. Hence, for initial temperature, we use sufficiently large initial temperature (100) to try various SWNoC configurations. In steps 9–12, we store the current architecture as a best architecture if the communication path length of the new architecture is less than that of the best architecture. In steps 17 and 18, we reduce the temperature by $\theta$, which is the cooling rate of the temperature. The parameter $\beta$ is the decreasing rate of $M$ (Lundy and Mees 1986). To avoid rapid cooling, we choose large enough values for $\theta$ and $\beta$ to avoid local optimum. This process continues until the temperature is lower than 1. We consider the optimized SWNoC configuration to compare with other optimization techniques as described earlier.

**ALGORITHM 5:** Simulated Annealing-Based Network Optimization Algorithm

**Input**: $N$ = # cores (64, 128, 256), $T$ = #*layers* (4), $Temp$ = 100,
$L$ = # links (144, 304, 640), $M = (3 \times 10^3, 1 \times 10^4, 6 \times 10^4)$, $\theta = 0.98$, $\beta = 0.98$
**Output**: Optimized 3D SWNoC

```
1   Initialize
2   while (Temp > 1)
3     for m=1 to M do
4       Link perturbation
5       ΔO = O_new − O_old
6       r = random(0, 1)
7       if e^(−ΔO/Temp) ≥ r or ΔO < 0, then
8         O_old = O_new
9         if O_new < O_best
10          O_best = O_new
11          Best arch. = Current arch.
12        end if
13      else
14        Restore
15      end if
16    end for
17    Temp = θ × Temp
18    M = β × M
19  end while
```

## 4 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we evaluate the network latency and EDP of the 3D SWNoC designed with the optimization algorithms discussed above.

### 4.1 Experimental Setup

We use a cycle-accurate NoC simulator that can simulate any irregular 3D architecture (Wettin el at. 2014). We consider three system sizes of 64, 128, and 256 cores divided into a fixed number of layers, $T$ (=4). Hence, for these system sizes, the cores and routers are placed in $4 \times 4 \times 4$ (64 cores), $4 \times 8 \times 4$ (128 cores), and $8 \times 8 \times 4$ (256 cores) grids, respectively, to pack them as closely as possible. The cores and the network routers are equally partitioned into the four layers. The routers are synthesized from an RTL-level design using TSMC 65-nm CMOS process in Synopsys Design Vision. All router ports have a buffer depth of two flits and each router port has four virtual channels. The NoC simulator uses wormhole routing, where the data flits follow the header flits once the router establishes a path. Due to the irregular nature of our 3D SWNoC architecture, the topology-agnostic Adaptive Layered Shortest Path Routing (ALASH) is used as the routing algorithm (Lysne et al. 2006). In this experimental study, we use 7 applications with widely varying traffic patterns. We use seven MCSL benchmarks (FFT, SPEC fpppp (FPPPP), H.264 decoder, Robot control (ROBOT), Reed-Solomon encoder and decoder (RS_enc and RS_dec) and sparse matrix solver (SPARSE)) (Liu et al. 2011). These benchmarks vary in characteristics from computation intensive to communication intensive in nature and thus are of interest in this work.

### 4.2 Performance of Optimized 3D NoC

In this section, we compare the performance of the SEN-, the SA-, and the ML-based link placement algorithms. We employ network latency and energy-delay-product (EDP) as our performance metrics. EDP is defined as the product of network latency and energy consumption and unifies both of them into a single value. For comparative performance evaluation, the performance metrics are normalized with respect to those of the 3D SWNoC optimized with the SA algorithm.

*4.2.1 Network Latency.* Figure 4 shows the normalized network latency values of the 3D SWNoCs optimized with SA-, ML-, and SEN-based algorithms for 64-, 128-, and 256-core system sizes. We observe from the figure that SWNoC designed with SEN achieves the lowest network latency for all the simulation cases. Compared to SA, SEN reduces the network latency by maximum 7.1% (4.3% on average), 10.4% (5.9% on average), and 29.0% (11.5% on average) for the 64-, 128-, and 256-core systems, respectively. In addition, SEN-based 3D SWNoC reduces the network latency more effectively as the system size increases from 64 to 256.

However, the ML-based 3D SWNoC has 0.5%, 1.1%, and 2.1% higher network latency on average for the 64-, 128-, and 256-core system sizes, respectively, than the SA-based counterpart. For FFT, FPPPP, H.264, and SPARSE, the ML approach consistently achieves lower network latency than SA-based solution for all the system sizes. The maximum reduction in the network latency is achieved for the FFT benchmark for all the system sizes. For ROBOT, RS_dec, and RS_enc, however, ML-optimized 3D SWNoC shows 6.5%, 7.9%, and 12.1% higher network latency on average, respectively, than SA-based design for the 64-, 128-, and 256-core system sizes. For these three benchmarks, a large number of cores do not have any inter-node communications, so the variation of the feature vectors is minimal (zero variation is observed in most of the cases). Consequently, the ML-based algorithm fails to distinguish different candidate solutions from each other and cannot capture the network characteristics efficiently. We analyze the performance of the 3D SWNoCs optimized by the ML-based algorithm and the dependency of the performance of the feature functions in more detail in Section 4.3.2.

The SEN-optimized 3D SWNoCs exhibit the lowest network latency for all the benchmarks and the three system sizes. To explain this, the normalized path lengths of the 3D SWNoCs optimized with the SA-, ML-, and SEN-based algorithms for 64-, 128-, and 256-core system sizes are shown in Figure 5. As the figure shows, the SEN-based 3D SWNoC achieves 5.8%, 7.9%, and 12.2% lower

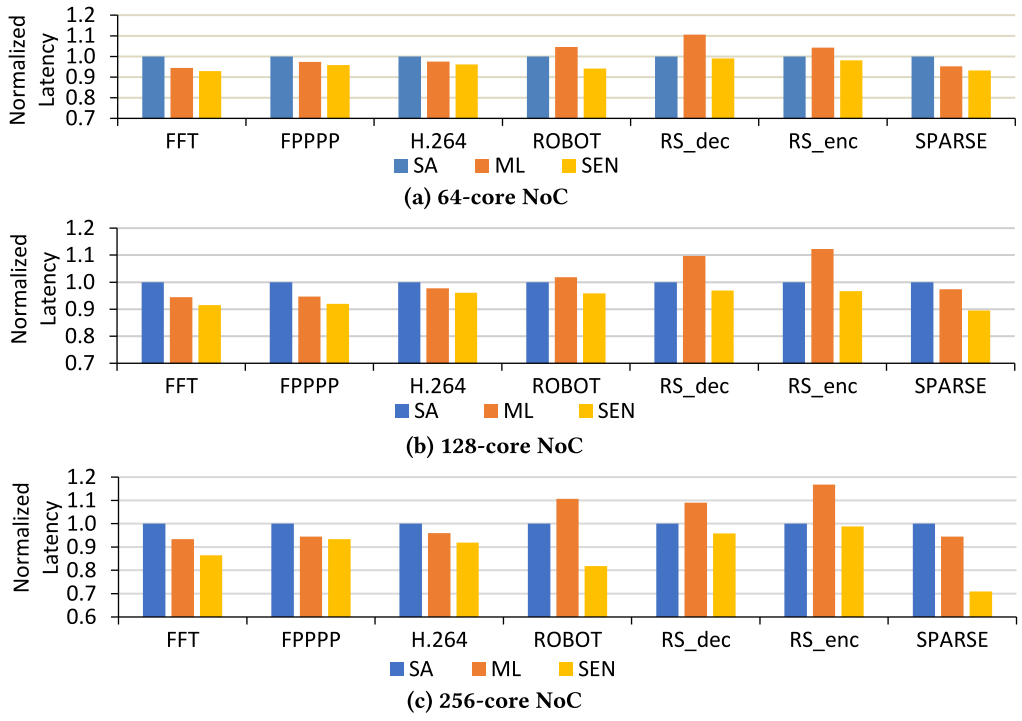


Fig. 4. Normalized network latency of the 3D SWNoC architectures optimized by SA-, ML- and SEN-based optimization algorithms for 64-, 128-, and 256-core NoCs.

path lengths on average than the SA-based design for the 64-, 128-, and 256-core systems sizes, respectively. However, the ML-based 3D SWNoC shows 3.2%, 2.8%, and 2.9% higher path lengths on average than the SA-based solution for the 64-, 128-, and 256-core system sizes, respectively. Similar to the network latency, ML-based design has much higher path lengths than SA for ROBOT, RS_dec, and RS_enc, which leads to the higher average path lengths.

*4.2.2 Energy-Delay-Product (EDP).* Figure 6 shows the EDP values of the 3D SWNoCs optimized by SA, ML, and SEN for 64-, 128-, and 256-core system sizes. As Figures 4 and 6 show, the EDP result has the same trend as the network latency result explained in Section 4.2.1. The average EDP values of the 3D SWNoCs optimized by SEN for the 64-, 128-, and 256-core systems are lower than those of the 3D SWNoCs optimized by SA by 8.3%, 10.7%, and 18.4%, respectively. In addition, the SEN-based algorithm achieves the lowest EDP for all the benchmarks and system sizes as the figure shows. However, the average EDP values of the 3D SWNoCs optimized by ML for the 64-, 128-, and 256-core systems are higher than those optimized by SA by 0.2%, 1.6%, and 3.3%, respectively. The reason that SEN achieves the lowest EDP is that each message travels shorter planar and vertical link lengths in the SEN-optimized 3D SWNoCs than in the SA- and ML-optimized 3D SWNoCs. Since shorter travel lengths lead to lower network energy consumption, the SEN-optimized 3D SWNoCs achieve the lowest EDP.

### 4.3 Effects of the Optimization Algorithms: SEN and ML

In this section, we analyze the SEN- and ML-based optimization algorithms from NoC optimization perspective. In addition, we also explain how the quality of the solution and the optimization

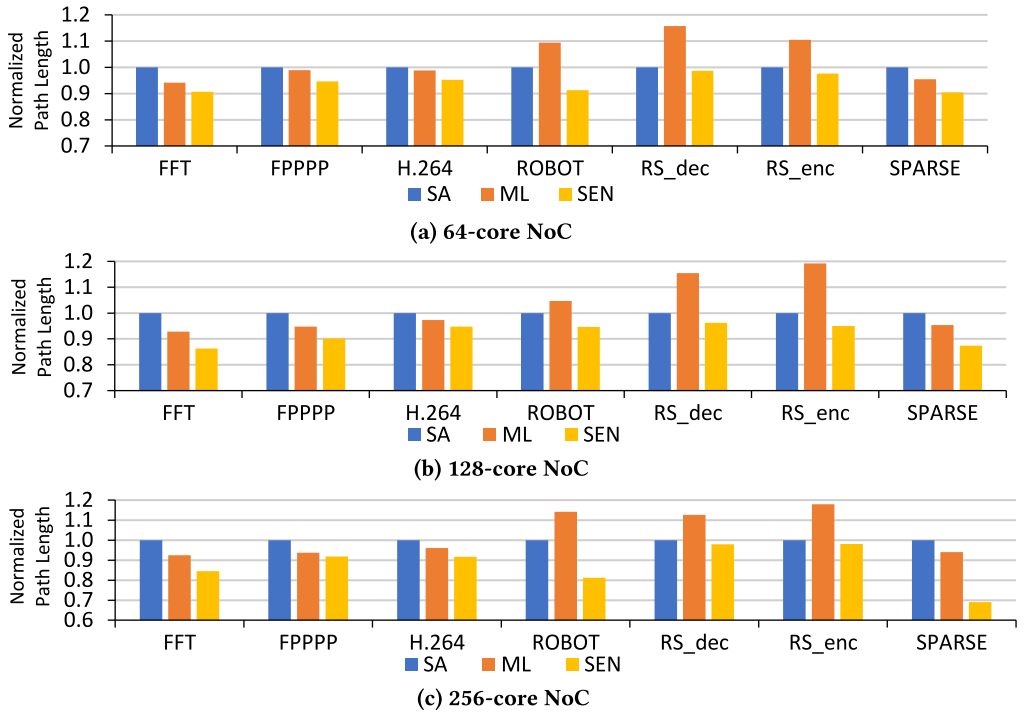


Fig. 5. Normalized path lengths of the 3D SWNoC architectures optimized by the SA-, ML-, and SEN-based optimization algorithms for 64-, 128-, and 256-core NoCs.

runtime vary with the characteristic of the benchmarks and system sizes for the two optimization algorithms.

*4.3.1 Analysis of SEN-based Optimized Network.* The proposed SEN-based algorithm has potential to reduce the network latency of the 3D SWNoCs and the optimization runtime with additional optimization steps, viz., refinement (for improving the quality of candidate NoC solutions) and removing a large number of links in a single-step (for runtime improvement). In this section, we discuss the details of employing these two additional steps during the optimization process and their effects on the quality of the optimized 3D SWNoC architecture and the amount of runtime reduction. We use the 64-core NoC for the analysis of the SEN-based algorithm.

*4.3.1.1 Determination of R Parameter for Efficient Refinement.* As discussed in Section 3.3.4.1, we adopt the refinement approach for the SEN-based optimization to overcome the problem of stalling at local optimal points. The refinement step repeats adding and removing $R$ links where $R$ decides the quality of an optimized 3D SWNoC and the optimization runtime. If we increase $R$, then we spend more time to explore a larger solution space, which will likely lead to a lower network latency value, at the cost of increased optimization runtime. Since there is a trade-off between the quality of an optimized 3D SWNoC and the optimization runtime, finding an optimal value of $R$ is crucial. Thus, we experimentally determine the optimal value of $R$ to improve the efficiency of the SEN-based algorithm.

Figure 7 shows the average EDP and runtime values of the seven benchmarks of the 64-core 3D SWNoCs optimized by the SEN-based algorithm for different $R$ values. The EDP values are normalized with respect to the EDP of the 3D SWNoC optimized with $R = 0$, which does not

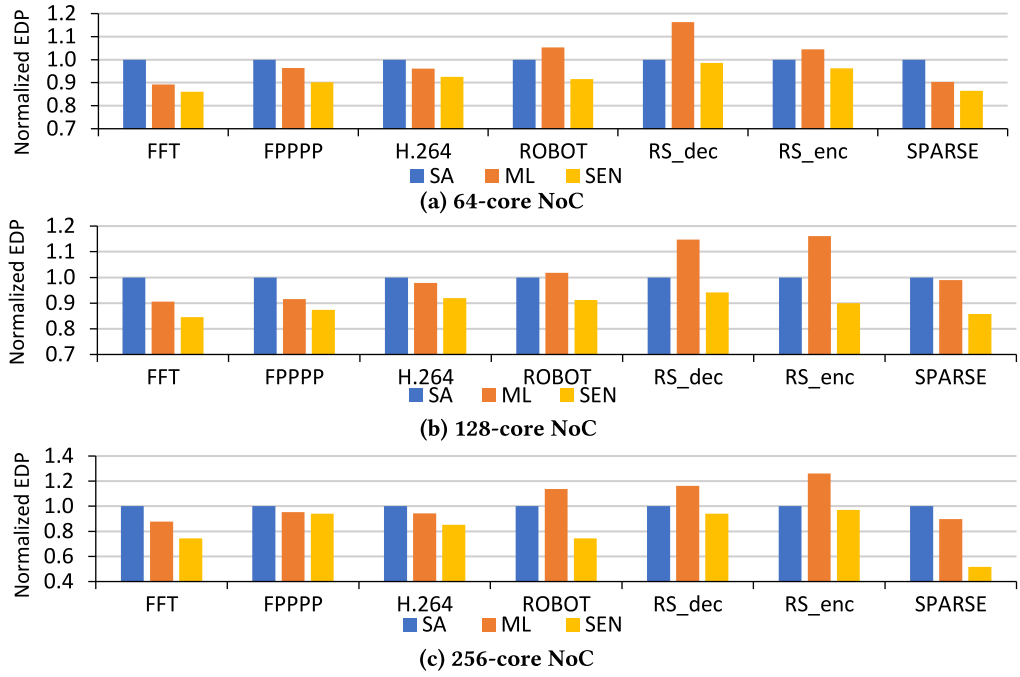


Fig. 6. Normalized EDP values of the 3D SWNoC architectures optimized by the SA-, ML-, and SEN-based optimization algorithms for 64-, 128-, and 256-core NoCs.

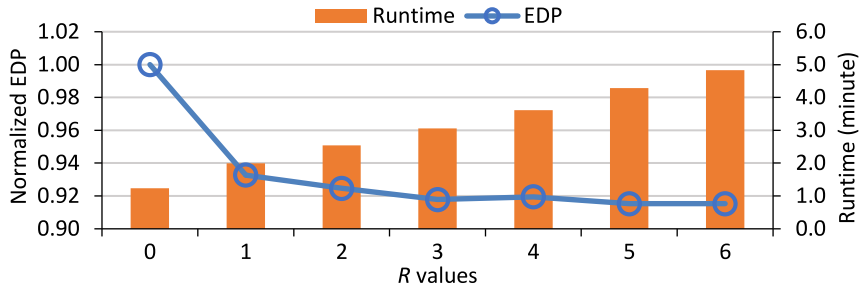


Fig. 7. Normalized average EDP and runtime values of the seven benchmarks of the 64-core 3D SWNoCs optimized by the SEN-based algorithm for different R values.

perform the refinement during network optimization. We observe from the figure that the EDP decreases and the runtime increases monotonically as $R$ increases from 0 to 6. The EDP reduces noticeably up to $R$ value of 3, and then remains almost constant even if $R$ increases further. For this reason, we used $R = 3$ for the SEN-based network optimization in Section 3.3.4.1.

*4.3.1.2 Runtime Improvement.* As discussed in Section 3.3.5, the joint effect of the one-time removal of a large number of links at the beginning of the network optimization and the refinement procedure is that we can significantly reduce the optimization runtime without degrading the quality of the NoC architecture. Before we start the three-step SEN-based optimization procedure (Section 3.3.2.2), we can remove some low-communication links ($f_{ij} \approx 0$), which are non-critical from the NoC performance perspective. If restoring some of the removed links can improve the

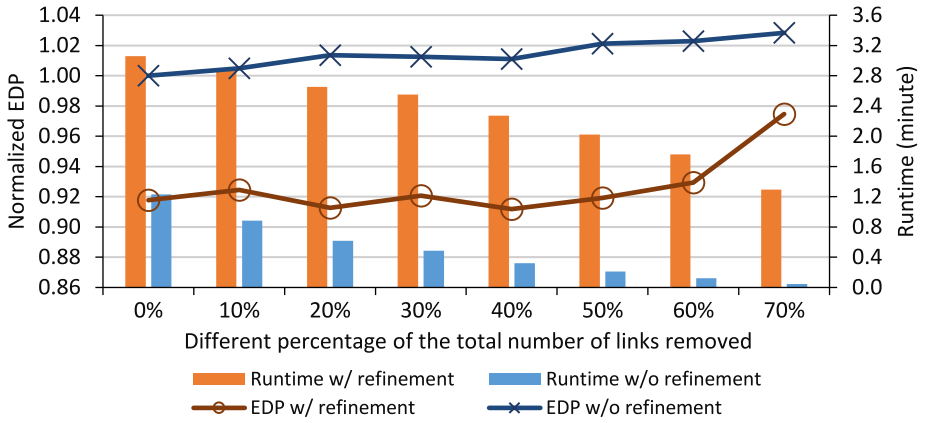


Fig. 8. Normalized average EDP values of the seven benchmarks of the 64-core 3D SWNoCs optimized by the SEN-based algorithm with and without the network refinement step and their optimization runtimes for different numbers of links removed at the beginning of the optimization.

performance of the NoC architecture, then the network refinement step will re-examine the effects of the links on the performance of the NoC architecture and restore them. Consequently, we can save the unnecessary computation time associated with non-critical link search and removal during the optimization process.

To find an optimal number of links, we can remove without sensitivity update at the beginning of the optimization, we removed $K_{init}$% of the total links and started the normal network optimization process (link constraint, removal, updating the current state and refinement). Figure 8 shows the average EDP values of the seven benchmarks of the 64-core 3D SWNoCs optimized by SEN with and without the network refinement step and their optimization runtime values for different $K_{init}$ varying from 0% to 70% with a step size of 10. The reason we do not explore removing more than 70% is as follows. Since we keep removing links during the optimization process, we cannot have 144 links at the end of the optimization if the 3D NoC architecture has less than 144 links after the one-time removal of links. Although the refinement step temporarily adds some links, the total number of links before and after a refinement step does not change. Since the initial 3D NoC architecture has total 528 links (120 horizontal links forming a complete graph in each layer and 16 vertical links between two adjacent layers), removing 70% and 80% links leave 158 links and 106 in the architecture, respectively. Thus, we experimented with removing up to 70% links. In Figure 8, the EDP values of the 3D SWNoCs are normalized with respect to that of the 3D SWNoC optimized with no one-time link removal and without the refinement step.

We first observe in the figure that the EDP values increase as $K_{init}$% increases from 0% to 70% if the network refinement step is not used. This means that some of the links removed at the one-time removal step become critical for EDP reduction during the main optimization process. Without the network refinement step, however, those links cannot be restored in the main optimization process. Thus, removing more links at the one-time removal step results in the increased EDP. However, the EDP values do not monotonically increase as $K_{init}$% increases if the network refinement step is used. This is because some of the links that are removed at the one-time removal step, but become critical for EDP reduction during the main optimization process are restored by the network refinement step. In Figure 8, the EDP values of the cases of removing 20% and 40% of the links in the one-time removal step are lower than the EDP value of the case of skipping the one-time removal step. Moreover, the EDP of the case of removing even 50% of the total links at

the one-time removal step is also similar to the case of $K_{init} = 0\%$ while the runtime of the former is 50% shorter than that of the latter.

If more than 50% of the links are removed at the one-time link removal step, however, then the EDP value starts increasing, even if the network refinement step is used. This is because the network refinement step cannot restore some of the critical links removed earlier. A way to resolve this problem would be to increase $R$ so that the refinement step can search a larger design space. However, increasing $R$ will also increase the optimization runtime. From all these observations, we find that combining the one-time link removal step and the network refinement step with proper $K_{init}$ and $R$ values can significantly reduce the optimization runtime without degrading the EDP of the 3D SWNoC architecture.

*4.3.2 Analysis of ML-based Optimized Network.* In this section, we analyze the performance of the ML-based NoC optimization algorithm. Especially, we discuss how the ML-based algorithm explores the NoC design space, successfully learns the solution space, and converges to the final network configuration. In addition, we explain why the performance of the ML-based algorithm is limited for some of the benchmarks considered here. We consider the change in feature vectors of the 3D SWNoC and the communication characteristics of different benchmarks to analyze the performance of the ML-based approach.

*4.3.2.1 Change in Feature Functions with ML.* In the machine-learning-based optimizations, the set of candidate solutions in the design space is represented as a function of the feature vectors. Depending on the effectiveness of the feature vectors and type of the learning function, the optimization algorithm explores and gathers knowledge about the design space. In this work, we defined in total 20 feature vectors to represent any NoC configuration in the design space. The differences in respective values of the feature vectors are utilized to separate those NoC configurations from each other. In general, larger variation separates different candidates in the solution space more efficiently while no difference in feature vectors represents identical NoC configurations.

To explain the effectiveness of STAGE-based NoC optimization approach, Figures 9(a) and 9(b) show the difference in the values of all the feature vectors between a randomly generated SWNoC and the optimized architecture (initial and terminal state of the base search as outlined in Section 3.4 and Figure 3). For the ease of understanding, the values of all feature vectors are normalized with respect to the values of the initial solution. Here, as an example, we consider 64-core NoC configuration for two benchmarks with opposite characteristics (from communication perspective) viz. FFT and RS_dec. The other two system sizes (128- and 256-core) also show similar characteristics. In addition, other benchmarks (FPPPP, H.264, RS_enc, ROBOT, and SPARSE) also show similar feature vector characteristics like either FFT or RS_dec, and hence, we have not shown the traffic characteristics plots for them.

From the figures, it is seen that for FFT benchmark, there are consistent variations between the initial and final optimized architectures across all the feature vectors. This states that the NoC solution space of FFT benchmark has a large variation across the candidates and they are distinguishable from each other. As a result, the ML-based approach can learn this space easily and exploit these variations to explore better quality solutions during the optimization process. Consequently, for FFT benchmark, the ML performs similarly compared to SA or SEN algorithms (Figures 4 and 6 shown before).

However, for RS_dec benchmark, there are variations for only some of the features while others have only marginal or no variations at all. This refers to a solution space where NoC candidate solutions have identical characteristics and inseparable from each other. As a result, the trajectory of the base search (as shown in Figure 3) is short, number of data points in the training set is small,

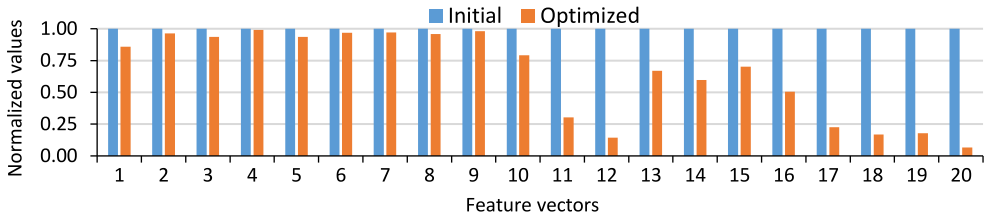


Fig. 9(a). Variation in feature vectors for the initial (random) and optimized 64-core 3D SWNoC with STAGE-based optimization for FFT benchmark.

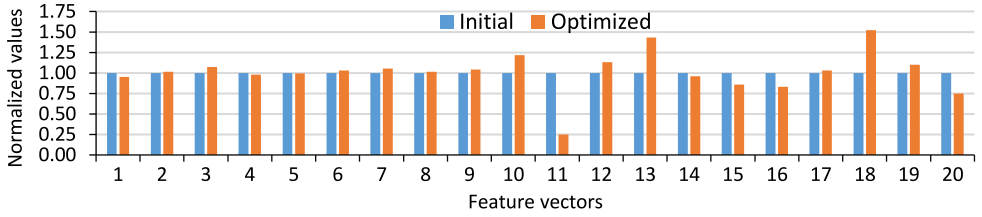


Fig. 9(b). Variation in feature vectors for the initial (random) and optimized 64-core 3D SWNoC with STAGE-based optimization for RS_dec benchmark.

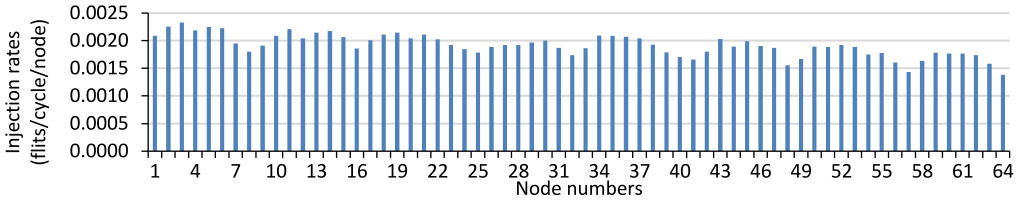


Fig. 10(a). Internode traffic injection rates for each node for 64-core system with FFT.

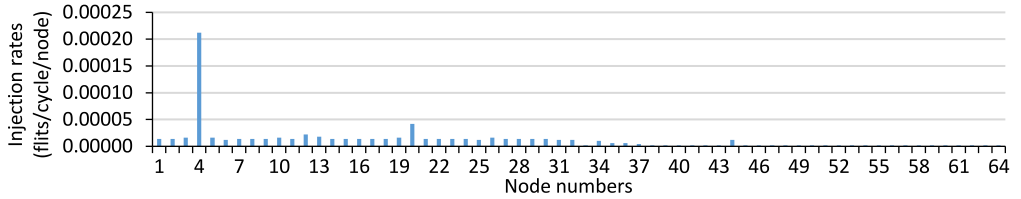


Fig. 10(b). Internode traffic injection rates for each node for 64-core system with RS_dec.

and data points are also identical in nature, and consequently the search space is difficult to learn for the evaluation function, $E$. Hence, the quality of the NoC solutions explored by the STAGE in the considered NoC design space does not improve significantly as seen from Figures 4 and 6 before. This type of variation in feature vectors is observed for RS_enc and Robot benchmark as well and, hence, they also show similar quality solutions during the optimization.

*4.3.2.2 Analysis of Benchmark Characteristics.* To analyze the performance of STAGE-based optimization for the benchmarks further, we consider the traffic injection rates of each node in the system and their mean and standard deviation. In this case, we also consider the 64-core NoC for illustration purpose. Figures 10(a) and 10(b) show the injection rates in flits/cycle/node for

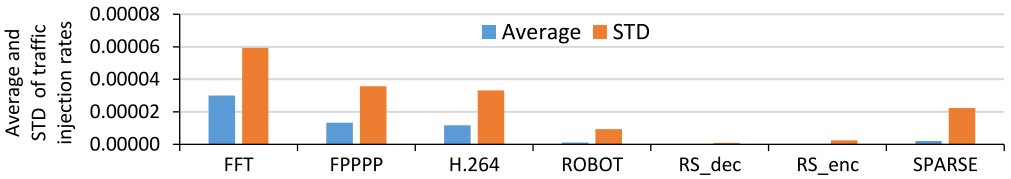


Fig. 11. Average and standard deviation (STD) in injection rates among the nodes of a 64-core system.

each node for FFT and RS_dec benchmarks, respectively. While the traffic injection rates for FFT benchmark (Figure 10(a)) has non-zero values across all the nodes, however, large number of nodes have really low rates (almost zero) compared to some other benchmark like RS_dec (Figure 10(b)). As a result, for FFT, the internode communications vary widely, and for RS_dec benchmark, large number of nodes do not interact with each other. To elaborate this, Figure 11 shows the average and standard deviation (STD) in traffic injection rates of 64-core NoC for all the benchmarks.

As seen from this figure, both the average and standard deviation values of the injection rates vary widely among the benchmarks. In addition, the FFT benchmark has higher values for both the average and standard deviation compared to the RS_dec benchmark. As a result, the communication characteristics of individual nodes in RS_dec benchmark are indistinguishable from each other. Consequently, for RS_dec, different candidate solutions in the NoC design space have similar values for respective feature vectors, and are not distinguishable. As a result, the performance of STAGE-based optimization for RS_dec benchmark is limited compared to the other algorithms considered in this work.

For FFT, due to a large variation in communication characteristics, the NoC candidates are easily distinguishable from each other, and hence, STAGE learns the NoC design space efficiently to explore high-quality solutions. Finally, following similar arguments, the performance of STAGE for other benchmarks can be explained and found to be consistent with the performance shown in Figures 4 and 6.

*4.3.3 Performance Analysis of SA-based Optimized SWNoC.* In this section, we analyze the performance of the SA-, ML-, and SEN-based NoC optimization algorithms in detail. Especially, we discuss how the SEN-based algorithm achieves better latency, EDP, and runtime than the SA-based algorithm. Figure 12 shows the difference in the communication path lengths of the 64-core SWNoCs optimized by the SA-, ML-, and SEN-based algorithms for the FFT benchmark as an example. The considered NoCs have total 4032 ($P(64,\ 2) = 64!/62!$) $f_{ij}$ values. However, the total communication path length is generally dominated by frequently-communicating node pairs, which have high $f_{ij}$ values. Thus, we pick the top 300 (64% of the total traffic) among the 4032 $f_{ij}$ values and analyze them. In Figure 12, the $x$-axis represents the 300 communication node pairs that are sorted in the descending order of the $f_{ij}$ values. The $y$-axis shows the differences in the communication path lengths obtained through two optimization algorithms ($O_{ij,SA} - O_{ij,SEN}$ and $O_{ij,ML} - O_{ij,SEN}$ in Figures 12(a) and (b), respectively). If SEN achieves lower path length than SA, then $O_{ij,SA} - O_{ij,SEN}$ is positive in Figure 12(a). If SA achieves lower path length than SEN, then $O_{ij,SA} - O_{ij,SEN}$ is negative.

Figure 12(a) shows the difference in communication path length of SWNoC optimized by the SA- and SEN-based algorithms. We observe in the figure that SEN achieves lower communication path lengths than SA for most of the frequently communicating nodes. SA uses random perturbations for the neighborhood search and the acceptance of the perturbations follows the Boltzmann distribution. Thus, SA finds a suboptimal solution with limited time and generally requires numerous iterations to reach a desired result. The number of solutions of the 64-core NoC

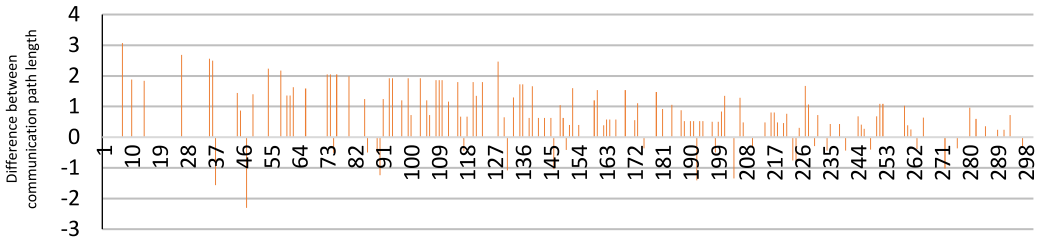


Fig. 12(a). The difference in communication path length of the 64-node SWNoC optimized by the SA- and SEN-based algorithm for FFT benchmark.

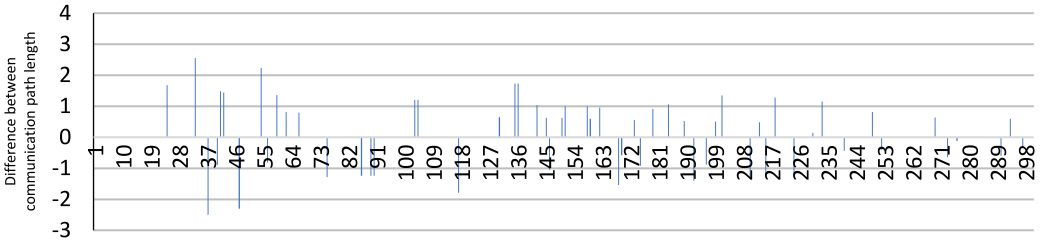


Fig. 12(b). The difference in communication path length of the 64-node SWNoC optimized by the ML- and SEN-based algorithm for FFT benchmark.

optimization problem we are solving in this article is approximately $1.04 \times 10^{62}$. (Each layer has 16 cores on a $4 \times 4$ grid and 24 links. Assuming the link distribution of each layer is $\{16,\ 5,\ 2,\ 1\}$, the total number of core pairs ($n_r$) separated by the Euclidean distance $r$ in each layer as follows: $n_1 = 24,\ n_2 = 34, n_3 = 40, n_4 = 20$. Thus, the total number of combinations of the link placement in each layer is $\binom{24}{16} \times \binom{34}{5} \times \binom{40}{2} \times \binom{20}{1} \approx 3.19 \times 10^{15}$. Since there are four layers, the total number of combinations of the link placement for the 64-core 3D NoC is $(3.19 \times 10^{15})^4 \approx 1.04 \times 10^{62}$.) Hence, the runtime of SA-based optimization is significantly longer than that of SEN. SA might be able to find a better solution than SEN, but it will require very long optimization time. However, the SEN-based optimization algorithm removes least-critical links and adds most-critical links repeatedly in the link removal and refinement steps. This methodology is similar to applying the gradient descent method (removing the least-critical link) with hill climbing (adding the most-critical link), so it is fast and effective at the same time.

As shown in Figure 12(b), ML-based NoC achieves comparable path length with SEN. The ML technique intelligently searches the design space to optimize the placement of links, and converges to the same quality solution with significantly lower runtime when compared to the SA. Therefore, as shown in Figures 4 and 6, the latency and EDP differences between ML and SEN are only 1.5% and 3.1%, respectively.

### 4.4 Convergence Time

In this section, we analyze the convergence time of all the optimization algorithms used in this work. Figure 13 shows the runtimes of the SA, ML, and SEN algorithms for 64-, 128-, and 256-core systems. It is seen from Figure 13 that the runtimes of the ML-based optimization for the 64-, 128-, and 256-core NoCs are on average 12.2, 23.4, and 96.8 times faster than the SA-based approach, respectively. Similarly, the runtimes of SEN-based optimization for the 64-, 128-, and 256-core NoCs are on average 32.7, 27.6, and 25.5 times faster for the same NoC sizes than SA. For the 64- and

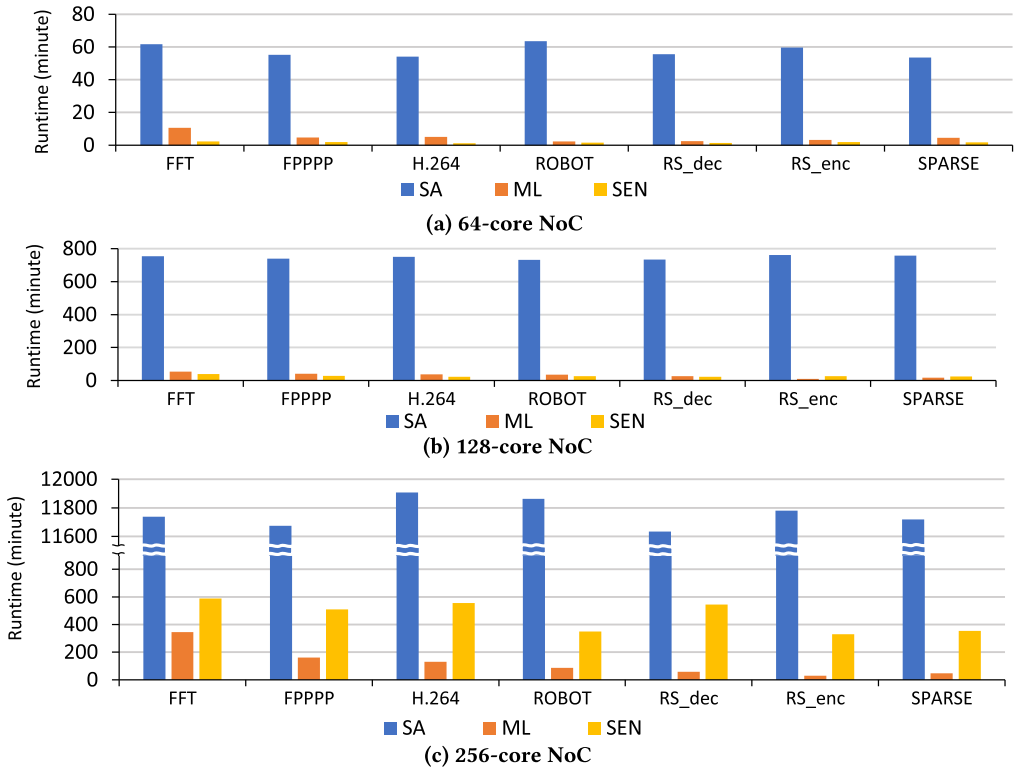


Fig. 13. Runtimes of the SA-, ML-, and SEN-based optimization algorithms.

128-core 3D NoCs, SEN is faster than ML. However, the runtime of the ML-based optimization for the 256-core system is shorter than the SEN-based optimization. As discussed in Section 3.3.4.2, the complexity of the SEN-based optimization is $O(L^2)$ where $L$ is the number of links of the fully connected 3D NoC for given core locations. However, the time complexity of the STAGE algorithm is $O(4PuT_b)$ where $P$, $u$, and $T_b$ refer to the number of iterations required to converge, the number of successors, and the length of the search trajectory, respectively (Section 3.4.4). Table 1 shows the experimental values of the parameters affecting the convergence time and the number of steps performed for the SEN-, ML-, and SA-based algorithms for the FFT benchmark as a case study. As seen from the table, the SA-based algorithm requires a larger number of steps than the SEN- and ML-based algorithms. This is because SA uses random perturbations for the neighborhood search and the new state is accepted with probability $Prob(\Delta O) = e^{-\Delta O/Temperature}$ (Kirpatrick et al. 1983). Thus, SA generally requires numerous iterations to reach a desired result. The SEN-based algorithm requires a smaller number of steps than the ML-based algorithm for 64- and 128-core systems. However, the SEN-based algorithm requires a larger number of steps than the ML-based algorithm for the 256-core system. In the ML-based approach, a significant amount of time is spent in generating training data. In contrast, the SEN algorithm removes links and updates the cost function. As a result, the ML-based algorithm has higher convergence time for the 64-core system even when the number of steps is orders of magnitude lower than the SEN algorithm. As the system size increases, however, the number of steps in the SEN algorithm increases more rapidly than the ML algorithm. Hence, the runtime of the SEN algorithm increases much faster than the ML algorithm as the system size goes up. As Figure 13 shows, the SEN algorithm has much shorter

Table 1. Different Parameters for NoC-optimization Convergence Time Analysis

| SEN-based NoC optimization (estimated approximately) | | | | | |
|---|---|---|---|---|---|
| System size, $N$ | Number of effective links, $L$ | Number of initial links removed, $K_{init}$ | Refinement depth, $R$ | | Number of SEN-steps |
| 64 | 163 | 60% | 3 | | $7.97 \times 10^4$ |
| 128 | 736 | 60% | 3 | | $1.63 \times 10^6$ |
| 256 | 3110 | 60% | 3 | | $2.90 \times 10^7$ |
| ML-based NoC optimization (estimated approximately) | | | | | |
| System size, $N$ | Number of links, $L$ | Number of iterations required for convergence, $P$ | Number of successors, $u$ | Search Trajectory, $T_b$ | Number of ML-steps |
| 64 | 144 | 6 | 500 | 60 | $0.72 \times 10^6$ |
| 128 | 304 | 9 | 500 | 108 | $1.94 \times 10^6$ |
| 256 | 640 | 13 | 500 | 156 | $4.06 \times 10^6$ |
| SA-based NoC optimization (estimated approximately) | | | | | |
| System size, $N$ | Number of links, $L$ | Initial and Min. Temp. | $\theta$ and $\beta$ | # of moves to attempt, $M$ | Number of SA-steps |
| 64 | 144 | 100 / 1 | 0.98 / 0.98 | $3 \times 10^3$ | $9.23 \times 10^6$ |
| 128 | 304 | | | $1 \times 10^4$ | $6.28 \times 10^7$ |
| 256 | 640 | | | $6 \times 10^4$ | $7.59 \times 10^8$ |

runtime than the ML algorithm for the 64-core system, but we observe the opposite trend for the 256-core system. Hence, we conclude that the SEN-based NoC optimization is more efficient for smaller system size whereas the ML algorithm is much more efficient for bigger systems.

## 5 CONCLUSION

In this article, we have proposed a sensitivity-based 3D SWNoC optimization algorithm, which is significantly faster than the traditional SA-based algorithm. The proposed algorithm repeatedly explores the impact of each link on the network performance and removes less important links to optimize the link placement. Also, we proposed a link refinement algorithm by which some of the previously removed links are re-inserted into the network to improve the performance of the 3D SWNoC. We compared the performance of 3D SWNoCs optimized by the SEN-based, the SA-based, and the ML-based algorithms. The comparison results show that the 3D SWNoCs optimized by the SEN algorithm has lower latency than those optimized by the SA and ML algorithms. The SEN algorithm also has the shortest runtime for optimization of the 64- and 128-core systems, but it is slower than the ML and still much faster than the SA for the 256-core system.

## REFERENCES

L. Benini and G. De Micheli. 2002. Networks on chips: A new SoC paradigm. In *Computer* 35, 1 (2002), 70–78.

J. A. Boyan and A. W. Moore. 2000. Learning evaluation functions to improve optimization by local search. *J. Mach. Learn. Res.* 1, 77–112.

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 *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'15)*. IEEE Press, 705–712.

S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty. 2017. Design-space exploration and optimization of an energy-efficient and reliable 3-D small-world network-on-chip. In *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 36, 5 (2017), 719–732.

B. S. Feero and P. P. Pande. 2008. Networks-on-chip in a three-dimensional environment: A performance evaluation. In *IEEE Trans. Comput.* 53, 32–45.

M. Healy, M. Vittes, M. Ekpanyapong, C. S. Ballapuram, S. K. Lim, H. S. Lee, and G. H. Loh. 2007. Multiobjective microarchitectural floorplanning for 2-D and 3-D ICs. In *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 26, 1 (2007), 38–52.

M. D. Humphries and K. Gurney. 2008. Network "small-world-ness": A quantitative method for determining canonical network equivalence. *J. PLOS One* 3, 167–256.

R. G. Kim, W. Choi, G. Liu, E. Mohandesi, P. P. Pande, D. Marculescu, and R. Marculescu. 2016. Wireless NoC for VFI-enabled multicore chip design: Performance evaluation and design trade-offs. In *IEEE Trans. Comput.* 65, 4 (2016), 1323–1336.

J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das. 2007. A novel dimensionally decomposed router for on-chip communication in 3D architectures. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*. ACM, New York, NY, 138–149.

S. Kirpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 671–680.

S. Kumar, J. Axel, J.-P. Soininen, F. Martti, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. 2002. A network on chip architecture and design methodology. In *Proceedings of the IEEE Computer Society Annual Symposium on Very Large Scale Integration (VLSI'02)*. 105–112.

F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir. 2006. Design and management of 3D chip multiprocessors using network-in-memory. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, 130–141.

C. W. Liu and Y. W. Chang. 2007. Power/ground network and floorplan cosynthesis for fast design convergence. In *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 26, 4 (2007), 693–704.

W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang. 2011. A noc traffic suite based on real applications. In *Proceedings of the 2011 IEEE Computer Society Annual Symposium on Very Large Scale Integration (VLSI'11)*. IEEE Comp. Society, Washington, DC, 66–71.

I. Loi, F. Angiolini, S. Mitra, and L. Benini. 2011. Characterization and implementation of fault-tolerant vertical links for 3-d networks-on-chip. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 30, 124–134.

M. Lundy and A. Mees. 1986. Convergence of an annealing algorithm. *Math. Program.* 34, 1 (Jan. 1986), 111–124.

O. Lysne, T. Skeie, S. A. Reinemo, and I. Theiss. 2006. Layered routing in irregular networks. In *IEEE Trans. Parall. Distrib. Syst.* 17, 51–65.

C. Marcon, R. Fernandes, R. Cataldo, F. Grando, T. Webber, A. Benso, and L. B. Poehls. 2014. Tiny NoC: A 3D mesh topology with router channel optimization for area and latency minimization. In *Proceedings of the International Conference on Very Large Scale Integration Design and 2014 13th International Conference on Embedded Systems*. 228–233.

H. Matsutani, M. Koibuchi, I. Fujiwara, T. Kagami, Y. Take, T. Kuroda, P. Bogdan, R. Marculescu, and H. Amano. 2014. Low-latency wireless 3D NoCs via randomized shortcut chips. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE'14)*.

H. Matsutani, M. Koibuchi, D. F. Hsu, and H. Amano. 2008. Three-dimensional layout of on-chip tree-based networks. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'08)*. 281–288.

V. Nguyen and C. Martel. 2005. Analyzing and characterizing small-world graphs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 311–320.

U. Y. Ogras and R. Marculescu. 2006. "It's a small world after all": NoC performance optimization via long-range link insertion. In *IEEE Trans. Very Large Scale Integr. Syst.* 14, 7, 693–706.

V. F. Pavlidis and E. G. Friedman. 2009. *Three-dimensional Integrated Circuit Design*. Morgan Kaufmann.

C. Seiculescu, S. Murali, L. Benini, and G. De Micheli. 2009. SunFloor 3D: A tool for networks on chip topology synthesis for 3D systems on chips. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'09)*. European Design and Automation Association, Belgium, 9–14.

C. Teuscher. 2007. Nature-inspired interconnects for self-assembled large-scale network-on-chip designs. *Chaos* 17, 2.

A. W. Topol, D. C. La Tulipe, Jr., L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Ieong. 2006. Three-dimensional integrated circuits. *IBM J. Res. Dev.* 50, 4/5 (July 2006), 491–506.

W. T. B. Uther and M. M. Veloso. 1998. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the 15th National Conference on Artificial Intelligence*. 769–774.

K. Wang and S. Dong. 2009. Power optimization for application-specific 3D network-on-chip with multiple supply voltages. In *Proceedings of the Asia and South Pacific Design Automation Conferecne (ASPDAC'09)*. 362–367.

P. Wettin, R. Kim, J. Murray, X. Yu, P. P. Pande, A. Ganguly, and D. Heo. 2014. Design space exploration for wireless NoCs incorporating irregular network routing. In *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 33, 11 (2014), 1732–1745.

Y. Xie, G. H. Loh, B. Black, and K. Bernstein. 2006. Design space exploration for 3D architectures. *J. Emerg. Technol. Comput. Syst.* 2, 2 (Apr. 2006), 65–103.

Y. Xu, Y. Du, B. Zhao, X. Zhou, Y. Zhang, and J. Yang. 2009. A low-radix and low-diameter 3D interconnection network design. In *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*. 30–42.

P. Zhou, P. Yuh, and S. S. Sapatnekar. 2012. Optimized 3D network-on-chip design using simulated allocation. *ACM Trans. Des. Auto. Electron. Syst.* 17, 2, Article 12 (Apr. 2012).

Received August 2017; revised January 2018; accepted March 2018