Test-Delivery Optimization in Manycore SOCs*

Mukesh Agrawal, Student Member, IEEE, Michael Richter, Member, IEEE,
and Krishnendu Chakrabarty, Fellow, IEEE

Abstract—We present two test-data delivery optimization algorithms for system-on-chip (SOC) designs with hundreds of cores, where a network-on-chip (NOC) is used as the interconnection fabric. We first present an effective algorithm based on a subset-sum formulation to solve the test-delivery problem in NOCs with arbitrary topology that use dedicated routing. We further propose an algorithm for the important class of NOCs with grid topology and XY routing. The proposed algorithm is the first to co-optimize the number of access points, access-point locations, pin distribution to access points, and assignment of cores to access points for optimal test resource utilization of such NOCs. Test-time minimization is modeled as an NOC partitioning problem and solved with dynamic programming in polynomial time. Both the proposed methods yield high-quality results and are scalable to large SOCs with many cores. We present results on synthetic grid topology NOC-based SOCs constructed using cores from the ITC‘02 benchmark, and demonstrate the scalability of our approach for two SOCs of the future, one with nearly 1,000 cores and the other with 1,600 cores. Test scheduling under power constraints is also incorporated in the optimization framework.

I. INTRODUCTION

RECENT YEARS have seen the large-scale integration of an increasing number of embedded cores in system-on-chip (SOC) design. It has been predicted that the number of cores on a single chip will continue to increase in the future [1], [2]. These predictions are being matched by industry trends. For example, Nvidia has developed a manycore chip, called Fermi, that includes 512 CUDA cores [3]. Intel has announced a co-processor called Knights Corner that has over 85579 Neubiberg, Germany. Email: michael.z.richter@intel.com.

• We present a novel algorithm for minimizing test time for NOCs with arbitrary topology and dedicated routing that consistently yields shorter test times than previous approaches. This algorithm can also be used to minimize test times for SOCs using dedicated TAMs.
• We model test-delivery optimization problem for an NOC-based many-core SOC with grid topology as a grid partitioning problem and develop a dynamic programming solution.
• We show that optimal grid partitioning leads to significant reductions in test time. The partitioning solution is optimal in that minimum test time is obtained for rectangular partitions. The test times obtained using DP for the rectangular NOC topology are close to (no more than 10% in most cases) TAM-independent provable lower bounds.
• We demonstrate the scalability of the proposed method by deriving optimization results for realistic SOCs of today and for the future. We present results for a 400-core SOC (20x20 NOC) of today, and even larger SOCs of the future—992 cores (32x31 NOC) and 1,600 cores (40x40 NOC). The results were obtained in reasonable
time using modest computing resources.

- We show how test scheduling can be carried out under power constraints.
- The proposed dynamic-programming algorithm computes
the pareto frontier for a range of access-point counts and
total test-pin counts in a single run. Therefore, a wealth
of data can be obtained to effectively choose the number
of access points and test channels for the SOC.

The rest of the paper is organized as follows. Section II
discusses related prior work. Section III introduces the subset-
sum problem, and shows how it can be used for solving the
test-scheduling problem in SOCs. Section IV provides details
about the proposed dynamic programming algorithm. Test
scheduling under power constraints is discussed in Section V.
Results are presented in Section VI, and Section VII concludes
the paper.

II. RELATED PRIOR WORK

A typical NOC is made up of several network tiles. A
network tile, in turn, is composed of a router, a core and
a core-to-network interface [21]. Routers are responsible for
routing communication data according to a routing protocol.
Routers are interconnected using links that are composed of
multiple channels. A network interface translates between a
router’s and a core’s communication protocols.

The topology of the network describes the logical layout of
the NOC. In a grid-based layout, a router has multiple input
and output ports, e.g., one port each for five different directions
(north, south, east, west and core). A router receives data on
one of its input ports and forwards it to one of the output ports
according to the routing protocol. In the XY-routing protocol,
data is first routed along the X and then along the Y axis.
Since this scheme is not affected by on-chip network traffic,
routing decisions are deterministic.

The testing of the NOC infrastructure, fault detection, and
reconfiguration in the presence of faults have been studied in
[22], [23]. A scheduling algorithm that assigns a higher priority
to cores requiring longer test time, and finds shorter
test-delivery paths to these cores, was proposed in [15]. This
approach was extended to take into account power constraints
in [14], [16]. An optimization method to identify dedicated
routing paths and incorporate precedence constraints and
shared BIST resources was provided in [24].

A differential clocking scheme to reduce power consumption
in the cores, and utilize the capability of an NOC to run at a frequency higher than scan clock, was presented in
[17]. Multiple clock speeds can also be leveraged to reuse
common links connected to cores on a time-sharing basis [18].
Simultaneous optimization of NOC testing and core testing
was presented in [19].

A test-wrapper design for the reuse of functional intercon-
nects was introduced in [12], [13]. TAM width constraints
imposed by the NOC flit width, and the associated problem of
flit-bit under-utilization were highlighted in [25]. Other design
approaches are presented in [10], [11], but these methods do
not show how test time reduction can be achieved.

The compression of test input data for reducing the number
of test input pins was studied in [26]. Test-output compression
to reduce output pin-count was discussed in [20]. Our approach
can be extended to incorporate input and output compression
as proposed in [20], [26].

The need for partitioning the NOC to avoid jitter-less
transport of test and response data, and distributing different
bandwidths to these partitions was introduced in [9]. The TR-
Architect test scheduling algorithm [27] was extended in [13]
to minimize test time in a NOC with arbitrary topology and
dedicated routing. However, a systematic method for creating
valid partitions in an NOC with a grid topology that only
support XY-routing method was not considered.

In [20], a general test-scheduling problem was formulated
and solved using ILP. Furthermore, a heuristic approach was
also presented, where contention was avoided for routers and
links between cores that were assigned to different access
points. However, the heuristic is based on ILP that does not
scale well with the number of cores and the number of access
points. No methodology was discussed on an appropriate
selection of the total number of pins or the number of access
points. Moreover, no strategy for locating optimal positions of
access points was discussed in [20].

Two methods for delivering test data using NOC have been
studied in previous work. In packet-based scheduling [15],
every packet is scheduled independent of others; a path is
reserved for one packet and packets belonging to the same
test set can be assigned a different set of NOC resources. The
second approach, based on dedicated routing [24], schedules
all test packets of a core using the same path in the NOC.
Due to its simplicity, test scheduling using dedicated routing
has been further explored in [13] and [28].

III. TEST SCHEDULING FOR NOCS WITH DEDICATED ROUTING

A. Problem Description

A graph-theoretic formulation for test-time minimization for
NOCs with dedicated routing is presented in [13]. A graph
is first constructed using the topology of NOC with network
tiles representing nodes and edges between nodes replacing
links. The goal is to partition the graph into disjoint connected
components, and distribute pins to each component such that
the maximum test time of each component is minimized. The
test time of a component is equal to the sum of the test times of
the cores in the component. The test time of a core depends
on the number of pins allotted to the component to which
the core belongs. We use the same formulation for devising
a test-scheduling algorithm for an arbitrary topology. We further
assume a dedicated routing path for the testing of each core
in line with previous work on test scheduling [13], [24], [28].

We build on the system model presented in [9] where ATE
interface (access points) and core wrapper do all the protocol
and width conversion such that both the ATE and the circuit
under test (CUT) are unaware of the on-chip routing protocol
and NOC design. We assume a width conversion ratio of one
in this work; any other ratio can easily be accounted for.

B. Outline of the algorithm

If we exclude the initial delay introduced during setting up
of dedicated routes in NOC for test-data delivery and response
collection, the test time minimization problem for NOCs with
dedicated routing can be considered as a special case of the
bus-based TAM optimization problem [13]. This is because
any partition of the set of cores is a valid solution candidate
to the latter, while only those partitions that represent connected
components in the graphical representation of the given NOC
topology are solution candidates to the former. Hence, we first
revisit the TAM optimization problem described in [6], [27]
for a bus-based TAM architecture. By limiting the partitions
considered by our algorithm to those that form valid connected
components, the algorithm is also directly applicable to the
NOC test scheduling problem (Section III.F).

The TAM optimization problem, discussed in [6], [27], can
be stated as follows. Given a set of cores with respective
test time information for a range of pin widths, total TAM
width, and the maximum number of TAM partitions, we have
to find an assignment of cores to the TAM partitions and the
TAM width of each partition such that the overall test time is
minimized. We view this problem from a different angle; we
partition the set of cores into disjoint subsets, and find an
optimal distribution of pins or TAM wires to these subsets
to minimize test time. A systematic method of enumerating
candidate partitions is discussed.

Our algorithm consists of three parts: a subset-sum formu-
lization to effectively enumerate candidate partitions in a
reasonable time, a greedy method to optimally distribute pins
to a given partition, and a brief introduction to the subset-sum
problem and a method to solve it. We first describe the subset-
sum problem (SSP) before elaborating on how we utilize SSP
to tackle the TAM-optimization problem.

C. Subset-sum problem (SSP)

The subset sum problem (SSP) is posed in the form of a
question. Given a set of $n$ integers, $X = \{x_1, x_2, \cdots, x_n\}$, does there exist a non-empty subset, whose elements sum to zero?
This problem is known to be NP-complete [29]. An equivalent
version of this problem asks the question if there is a subset of
the given set with the sum of its elements equal to a given
integer $s$. There exists no algorithm that can solve a given
instance of SSP in polynomial time. However, a method based
on dynamic programming solves the problem optimally in
pseudo-polynomial time [29]. Figure 1 outlines this method—
we call it \texttt{SolveSubsetSum}—that has run-time complexity of $O(n(L_{SSP} - U_{SSP}))$, where $L_{SSP}$ is the least element of
the given set, and $U_{SSP}$ is the sum of all positive integers of
the set, and $L_{SSP} \leq s \leq U_{SSP}$.

\begin{verbatim}
SolveSubsetSum (X)
1: \texttt{SUM}(1, x_1) \leftarrow \text{true}
2: for \(i \leftarrow 2 \) to \( n \) do
3: \hspace{1em} \( s \leftarrow L_{SSP} \) to \( U_{SSP} \) do
4: \hspace{2em} \text{if} \( s = x_i \) or \( \text{SUM}(i - 1, s) = \text{true} \) or \( \text{SUM}(i - 1, s - x_i) = \text{true} \) then
5: \hspace{3em} \( \text{SUM}(i, s) \leftarrow \text{true} \)
6: end if
7: end for
8: end for
9: return \( \text{SUM} \)
\end{verbatim}

Fig. 1. Pseudocode for the procedure \texttt{SolveSubsetSum}.

The binary element $\text{SUM}(i, s)$ of the 2-D array $\text{SUM}$ stores
a true if it is possible to create a subset from the set $\{x_1, x_2, \cdots, x_i\}$ with sum $s$, otherwise it stores a false. From the procedure
\texttt{SolveSubsetSum}, it is evident that $\text{SUM}(i, s)$ is true only
under three conditions:

- The subset contains only of $x_i$, i.e., $s = x_i$.
- There exists a subset with sum $s$ without using the
element $x_i$, i.e., $\text{SUM}(i - 1, s)$ is true.
- The subset contains $x_i$ in addition to elements from the
set $\{x_1, x_2, \cdots, x_{i-1}\}$, i.e., $\text{SUM}(i - 1, s - x_i)$ is true.

This recursive formulation is the foundation for solving SSP in
pseudo-polynomial time and we leverage this idea for TAM
optimization. We use repeated invocations of the above method
on multiple instances of SSP to enumerate several candidate
partitions of the given set of cores in $K$ subsets, and use a
greedy approach to distribute pins to these subsets.

D. Optimal pin distribution for a partition

Suppose the given set of cores is partitioned into $K$ disjoint
subsets, and the cores in a subset are assigned to the same
TAM. Moreover, no two cores from different subsets use the
same TAM. The number of pins available for distribution is
$P$, where $P$ is the total TAM width.

We present a greedy algorithm for distributing $P$ pins and
prove that the greedy approach is optimal for a partition of
a given set of cores. Figure 2 outlines the algorithm for pin
distribution. In each iteration, the subset having the maximum
test time is assigned an additional pin. The algorithm termi-
nates after every pin is assigned.

Let $G_P = \{g_1, g_2, \cdots, g_P\}$ be the solution generated by the
greedy approach, where $g_i$ denotes the index of the subset to
which the $i^{\text{th}}$ pin is assigned. Clearly, $1 \leq g_i \leq K$, for all
$i$. Let $O_P = \{o_1, o_2, \cdots, o_P\}$ be an arbitrary sequence of pin
assignments that leads to an optimal solution.

Let $T_g(i)$ denote the SOC test time after the $i^{\text{th}}$ pin
is assigned using the greedy approach. Similarly, let $T_o(i)$
denote the SOC test time in the $i^{\text{th}}$ step for the given
arbitrary sequence. Furthermore assume that $T_g(i, b)$ is the test time for the
$b^{\text{th}}$ subset after the $i^{\text{th}}$ iteration using the greedy approach. Let
$T_o(i, b)$ denotes the same quantity for the arbitrary sequence.

Clearly, $T_g(i) = \max_b[T_g(i, b)] \Rightarrow T_g(i, b) \leq T_g(i) \forall b$, and
$T_o(i) = \max_b[T_o(i, b)] \Rightarrow T_o(i, b) \leq T_o(i) \forall b$.

Lemma 1. The test time does not increase in any iteration of
the greedy method, i.e., $T_g(i + 1) \leq T_g(i)$.

Proof. The test time of a core does not increase with the
addition of a pin. Since a pin is added at each step or iteration
to exactly one subset, the test time of that subset can only
decrease or remain the same. The test times of other subsets
remain unaffected. This is also true for the arbitrary sequence, i.e., $T_a(i + 1) \leq T_a(i)$. □

**Lemma 2.** In the partial sequence $G_k = \{g_1, g_2, \ldots, g_x\}$, suppose that the pin $k$ was added to the subset $b_m$ during the $k^{th}$ step, i.e., $g_k = b_m$ and $T_a(k) = \tau_k(k, b_m)$. If a pin is removed from a subset, say $b'$, such that $b_m \neq b'$, then the updated test time of the SOC is strictly greater than $T_a(k)$.

**Proof.** According to the greedy procedure, a pin is added to a subset only when it has the maximum test time during an iteration. Since a pin is removed from $b'$, the pin must have been added to $b'$ during an iteration, say $i < k$, such that $\tau_k(i, b') = T_a(k)$. From Lemma 1, $T_a(k) \leq T_a(i) = \tau_k(i, b')$. Therefore, the updated test time of $b'$ after removing a pin is $\tau_k(i, b')$. Note that the greedy procedure keeps on selecting the same subset until its test time is strictly lower than another subset. Since $b_m \neq b'$, $\tau_k(i, b') > T_a(k)$. Since the test time of $b'$ dominates $T_a(k)$, the SOC test time is $\tau_k(i, b') > T_a(k)$. Note that the addition of the removed pin to any other subset does not reduce the SOC test time.

If $b_m = b'$, then $\tau_k(i, b') \geq T_a(k)$ holds. □

**Theorem 1.** The greedy algorithm of Figure 2 finds an optimal distribution of pins for a given partition on the set of cores.

**Proof.** We prove the theorem by contradiction. At step $i$, if the distribution of pins is same for both the methods, $T_a(i) = T_o(i)$ holds. If the distribution of pins is not the same, then there exists at least one subset that is assigned at least one more pin in the greedy approach than what is assigned to it in the other approach. If we try to rearrange the pins in the greedy sequence to match the pin distribution obtained from the arbitrary approach, then by Lemma 2, the test time can only increase; hence $T_a(i) \leq T_o(i)$ ∀$i$.

□

**E. Enumeration of candidate partitions**

The TAM-optimization problem has been shown to be NP-complete using transformation from the bin-packing problem in [30]; therefore, a polynomial-time algorithm cannot be designed for solving the problem optimally. The test time of cores is mapped to integer elements of the given set in SSP. Assuming that only one pin available to each TAM, a set consisting of the test times of all cores using just a single pin is created. The test times vary from small to large values depending on the scan chain length and the number of test patterns of individual cores. If the range is very large, it may be computationally prohibitive to run dynamic programming (Figure 1), both in terms of time and space. Therefore, the test times are scaled down by a factor of the lowest test time before approximating them to nearest integer values. The procedure optimizeTAM of Figure 3 is called on this set to create $K$ subsets. The greedy method outlined in Figure 2 is executed on the resulting partition and the test time is noted.

The element $SUM(i, s)$, for every $s$, holds an answer to the question whether the set $\{x_1, x_2, \ldots, x_i\}$ has a subset, the elements of which sum to $s$. For every value of $s$, we get a unique subset, if $SUM(i, s)$ holds a true. The procedure findSubset finds such a subset; it searches through the entries of the 2-D array $SUM$ and constructs a solution. This is the backbone for enumerating candidate partitions. Note that the procedure optimizeTAM is a recursive procedure and the maximum recursive depth is equal to the total number of subsets that are needed to be created. In each recursive step, a subset $X_i$ is created with sum less than equal to $s$. Then a new set $X_{\text{filter}}$ is obtained by removing the elements in $X_i$ from $X$. While $X_i$ is appended to the list candidate_partition, the set $X_{\text{filter}}$ is fed in the next level of recursion. After returning from the recursive call, $X_i$ is removed from candidate_partition.

This methodology can potentially enumerate all partitions, but for keeping the computation time low, we do not search through all the entries of the array $SUM$. First a lower bound $L_{\text{TAM}}$ is computed as $\sum_{x \in X} \Delta$, where $x \in X$. The variable $s$ is varied from $L_{\text{TAM}} - \Delta$ to $L_{\text{TAM}} + \Delta$, where $\Delta$ is a parameter that controls the size of solution space that is to be searched. Since the size of $X$ decreases in every recursive step, we reduce the value of $\Delta$ by half in our experiments in successive recursive steps (not shown in Figure 3). Either a timeout can be set on the overall procedure, or a limit be placed on the maximum number of candidate partitions that are to be enumerated.

**F. Application to NOC**

Similar to extension made for NOCs in [13], the method described in the previous subsection is modified for solving the test-delivery problem in an NOC. Listed below are the required modifications:

- The findSubset procedure only returns a subset, whose elements form a connected graph.
- In the distributePins, the number of pins assigned to each subset is no more than the channel width.

The method described in [13] creates several connected components in its initial step and subsequently attempts to find a partition that minimizes test time; there is no control on how many connected components are created in the final solution. In contrast, our approach takes the number of access points (or connected components) as an input parameter. In Section VI, we show how restricting the number of connected components in [13] adversely impacts test time. Even if the number of connected components is kept unrestricted, for larger NOCs with a grid topology, we demonstrate that the method in [13] produces a significantly large number of access points with
test times that are worse than the test times produced by our method with much fewer access points. More access points leads to additional DFT cost and higher power consumption.

IV. Dynamic Programming Approach for Grid Topology

The above approach and the approach from [13] are not applicable to an NOC with mesh topology that only supports XY-routing protocol for on-chip communication, as they create connected components of arbitrary shapes and do not ensure that packets are delivered without any conflict when the routing mechanism is XY. Moreover, XY routing can be viewed as a special case of dedicating routing, if the paths taken by packets in the former case are encoded in packet headers in the latter; therefore, the method that we propose in this section can still be used for NOCs with grid topology that use dedicated routing. We will demonstrate that by using XY-routing with grid topologies, we obtain better solutions than reported in [13] for larger NOCs. Figure 4 highlights the main difference between the two proposed methods.

A. Preliminaries

The basic idea of our approach is to partition a two-dimensional grid-structured NOC into rectangular regions, and interface each region with the ATE such that the testing in each region can be carried out in parallel without any contention for routers and links. The cores in a region are tested sequentially, whereas the regions are tested in parallel. This concept is illustrated in Figure 7. The figure shows a partition of a 4x4 NOC into three rectangular regions. These regions are interfaced with an ATE through ATE interfaces. The ATE drives test in all three regions simultaneously.

The goal of partitioning is to minimize test time for the region with the maximum test time. This method is suitable for NOCs that implement the XY-routing protocol, which is a popular routing algorithm because of its easy hardware implementation and guaranteed deadlock-free routing.

The advantage of optimal partitioning is illustrated in Figure 5. Each sub-figure shows a different partition of the same 4x4 grid network, where each cell represents a network tile. The number in each cell is the number of clock cycles needed to test the core in that network tile. Note that the additional time needed to establish a connection to a core from an access point is neglected here because it is very small compared to the test times of the cores. For the purpose of illustration only, we assume that the test pins are evenly distributed to the access points. The quantity T refers to the total number of cycles that is required to test the chip. Since each region is tested in parallel, the maximum number of clock cycles needed to test any region is T. Figure 5(b) shows an optimal partitioning that reduces test time by 40% compared to the partitions shown in Figure 5(c). The number of regions is three in this figure, but in general, it is an input parameter to the proposed framework.

B. Problem formulation

An NOC N with a grid topology can be viewed as a rectangle in a two-dimensional Cartesian plane with the bottom-left corner of N coinciding with the origin (0, 0) of this plane. Any rectangle R can be uniquely identified with a four-tuple representation [i, j, l, w] where (i, j) is the bottom-left corner of R, and l and w are the length and width of R, respectively. This representation can be captured by a concise expression, R ≡ [i, j, l, w]. If the dimension of N is m×n, we can write: N ≡ [0, 0, m, n].

Our goal is to optimally partition N into rectangular regions such that testing can be carried out independently in each region, thereby minimizing overall test time. This particular choice of regions being rectangular is motivated by two reasons: first, the popular XY-routing algorithm [31], [32] ensures that there is no conflict in test data transportation because data will not cross the boundary of regions; second, it is more tractable to store and use results for subproblems in our dynamic programming-based approach.

We have to ensure that each region can be accessed by the ATE without requiring a dedicated TAM, hence, not all possible partitions of N are admissible. The two partitions shown in Figure 6, for example, are inadmissible; region B is enclosed from all sides and it cannot be tested independently. We say a region to be located at a boundary if it is not enclosed, i.e., if it has at least one boundary edge. An edge of such a region R is said to be a boundary edge if it is incident on one of the edges of the rectangle [0, 0, m, n]. In Figure 6(b),
region A has three boundary edges, regions C and D have two each, E has one, and B has no boundary edges.

We create a partition with the help of a sequence of separators. A separator is any line segment parallel to either of the coordinate axes, and divides a region completely into two sub-regions. We impose the constraint that a separator has to be parallel to one of the axes. This constraint ensures that only rectangular regions are formed. We do not consider a sequence of line segments as shown in Figure 6(a) to create a partition; it can be easily shown that such a sequence will always create an enclosed region. A separator is horizontal if it is parallel to the axis y = 0, and vertical if it is parallel to x = 0.

The partition shown in Figure 8(a) can be easily mapped to a binary tree with the region represented by the NOC as the root of the tree and each intermediate node representing the sub-region that is created by placing a separator in its parent region; see Figure 8(b). The orientations of separators are also captured in the tree in terms of how a non-leaf node is intersected. We call a region corresponding to a leaf of this tree a leaf region. Alternatively, a leaf region can be defined as a region with no subdivisions.

We define the cost of a leaf region as the sum of the test times of the cores in that leaf region. The test time of a core is a function of the number of pins assigned to test the core. The number of pins used to test a core equals the number of pins assigned to the leaf region in which the core is present. The cost of an intermediate region R is the maximum cost over all costs of the leaf regions contained in R. The cost of R also depends on the corresponding partition size, where the size of a partition is the number of leaf regions created by that partition. Let us use πK(P) to denote the cost of N, where K is the given partition size (number of regions) and P is the number of available test pins. We can now formulate our problem as follows. Our goal is to determine an optimal sequence of separators such that:

- Each core is present in exactly one leaf region;
- Each leaf region is located at a boundary (boundary-region constraint);
- πK(P) is minimized.

For a given K and P, we use πK to refer to the cost of any region R. Our goal is also to compute the pin assignment to each leaf region (pin-assignment problem), i.e., which of the K pins are used to access different leaf regions.

C. Characterization of the optimal solution structure

Dynamic programming is an effective technique to attack optimization problems that possess an optimal substructure, i.e., for which an optimal solution can be computed from optimal solution to its smaller subproblems. Dynamic programming is typically applied when the same subproblem appears multiple times when a problem is decomposed; storing a solution to each subproblem avoids re-computation when it arises again. In this section, we show why we use dynamic programming, and how we find an optimal partition and construct the solution to the problem of test delivery.

In order to obtain an optimal solution, we have to select a sequence of separators that create a partition of size K. To simplify the following discussion, we are not considering the effect of pin width on test time, so we assign a fixed test time to each core. The extension of the algorithm to cover pin distribution is subsequently presented in section V.

Consider a m×n grid. Suppose that this grid has to be partitioned into two regions. In this case, only one separator is needed. If placed vertically, the separator can be placed at n−1 different positions. Similarly, m−1 possible positions for placing a horizontal separator accounts for a total of m+n−2 different positions. In order to minimize the test time, we compute the overall test time for each position of the separator and report the position resulting in the minimum test time. Increasing the partition size K to three requires an additional separator to be placed. If the first separator l1 is placed as shown in Figure 9(a), the second separator l2 can either be placed in Region A or Region B. There are m+a−2 ways to
place $l_2$ in $A$, and $m+n-a-2$ ways to place in $B$, resulting in a total of $2m+n-4$ ways. Thus for each choice of a position for $l_1$, there are several choices for placing $l_2$. The analysis can be continued in this way for larger values of $K$.

Suppose we are given an optimal partition for the grid and the position of $l_1$ is as shown in Fig. 9. The cost of $\mathbf{N}$ ($\pi_N$) is the maximum of the costs of the two regions, $A$ and $B$. Let us consider three cases: a) the cost of region $A$, $\pi_A$, is greater than that of region $B$ ($\pi_B$), i.e., $\pi_A > \pi_B$; b) $\pi_B > \pi_A$; c) $\pi_A = \pi_B$. In the first case, $\pi_A$ should be the optimal cost for $A$. If not, optimal cost for $A$ can replace $\pi_A$ to give a better value for $\pi_N$, which is a contradiction because we assumed $\pi_N$ to be optimal. A similar argument holds for the symmetrical second case: if $\pi_B$ is not optimal, i.e., if there were a better value for $\pi_B$, we can always substitute this value to yield a better value of $\pi_N$, which would be a contradiction. The third case can be analyzed in the same way to conclude that an optimal solution for this partitioning problem encapsulates optimal solutions for its subproblems. This property indicates optimal substructure, which is a basic requirement for applying dynamic programming. Let us recursively express this property for the above example:

$$\pi_N = \max(\pi_A, \pi_B)$$  

(1)

An optimal value of $\pi_N$ can be obtained by sweeping $l_1$ across its all possible positions. For each different position $\rho$ of $l_1$, the values for $\pi_A$ and $\pi_B$ changes. We can rewrite Equation (1) as follows:

$$\pi^* = \min_{\rho} \{\max(\pi_A, \pi_B)\}$$  

(2)

where $\pi^*$ is an optimal solution to the actual problem and $\rho$ varies over all the possible locations for placing $l_1$. Note that if the grid is to be partitioned into $K$ regions, the sum of the partition sizes of the regions $A$ and $B$ will be $K$. The partition sizes of $A$ and $B$ are not accounted for in the above equation. The partition cost is a function of partition size. Since changing the partition size of a region has an impact on overall partition cost, the value of $\pi^*$ depends on how many separators we assign to regions $A$ and $B$. For a given $P$, let us use $\tilde{\pi}(K)$ to denote the cost of $R$ when it is partitioned in $K$ regions. Factoring in the partition sizes for $A$ and $B$, Equation (2) becomes:

$$\pi^* = \tilde{\pi}_N(K) = \min_{\rho} \{\min_x \{\max(\tilde{\pi}_A(K-x), \tilde{\pi}_B(x))\}\}$$  

(3)

where $0 < x < K$.

Let us formulate a general equation for all possible subproblems. For any region $R \equiv [i, j, l, w]$, we rewrite Equation (3) by splitting it into two cases: the case when a separator is swept from top to down, and the case when it is swept from left to right. These two cases and the regions so formed are marked in Fig. 9(b) and Fig. 9(c). The final cost is the minimum cost obtained in these two sweeps.

$$\tilde{\pi}_R(k) = \min_{\omega} \min_x \{\max(\tilde{\pi}_A(k-x), \tilde{\pi}_B(x))\}, \min_{\omega} \min_x \{\max(\tilde{\pi}_A(k-x), \tilde{\pi}_B(x))\}$$  

(4)

where $0 < \omega < w$, $0 < \lambda < l$, $0 < x < k$, $R \equiv [i, j, l, w]$, $A^* \equiv [i, j, l, w]$, $B^* \equiv [i + \omega, j, l, w - \omega]$, $A^* \equiv [i, j, l, w]$, and $B^* \equiv [i, j + \lambda, l - \lambda, w]$.  

/*Enumerating all rectangles*/
for $w$ in $1 \ldots n$, $l$ in $1 \ldots m$, $i$ in $0 \ldots n_w$, $j$ in $0 \ldots m - l$ do
/*computing results for all partition sizes*/
STATE for $k$ \leftarrow 2 to $K$
computeAndStore($M, i, j, l, w, k$)
end for

Fig. 10. The top-level dynamic programming procedure for partitioning.

Fig. 11. computeAndStore Procedure.

We use Equation (4) in our algorithm. Since a subproblem is required to be solved in more than one larger problem, we use dynamic programming to solve this problem. The key idea is to store results for all subproblems and find the optimal cost $\pi^*$, which is same as $\tilde{\pi}_N(K)$. Next we discuss the algorithm and the data structures used.

D. Algorithm

The algorithm begins by enumerating all possible rectangles in $\mathbf{N}$, and for each such rectangle, computing and storing the optimal cost for all partition sizes $k$, $0 < k \leq K$. This is shown in Figure 10. The variables $l$ and $w$ are used to vary the length and width of the current rectangle, respectively. The pair $(i, j)$ denotes the bottom-left coordinate of a rectangle. A five-dimensional integer array $M$ is used for storing the optimal cost of all rectangles and for all partition sizes. The first four dimensions are used for identifying a rectangle, and the last dimension denotes the partition size. The variable $k$ iterates over all partition sizes less than equal to $K$. The trivial case of $k = 1$ is not shown in the figure and $M[i, j, l, w, 1]$ is initialized to the sum of the test times of all cores present in the rectangle $[i, j, l, w]$. The procedure computeAndStore, shown in Figure 11, describes how the optimal solution is reached.

The element $M[i, j, l, w, k]$ is initialized to a large integer value and it is updated with a smaller value found on sweeping the separator, first from left to right, and then from top to bottom. The variable $v$ stores the current position of the vertical separator. The optimal cost of the current rectangle depends on the partition cost of the two new subregions formed by the separator. The partition cost of subregions, in turn, depends on their respective partition sizes. We use a variable $x$ for varying the partition size in one of the subregions. If $x$ is the partition size of one of the subregions, then $k - x$ is the partition size for the other subregion.

In addition to producing the cost of an optimal partition, we also have to keep track of the sequence of separators used to construct this optimal solution. The procedure ConstructSolution, described in [33], can be used for this purpose.
V. ADDITIONAL CONSTRAINTS AND ENHANCEMENTS

In this section, we incorporate the boundary-region constraint, the pin-assignment strategy, and power constraints into our solution approach.

A. Boundary-region constraint

The boundary-region constraint mandates the inclusion of only boundary regions in the final solution. As described in [33], the procedure `computeAndStore` can be easily modified to enumerate only admissible partitions.

B. Pin-assignment problem

The test time of a core varies with the TAM width assigned to it. In this work, we do not consider the case when the TAM width of a core exceeds the channel width. To incorporate the effect of pin count on the partition cost, we need to add one more dimension to the search space. The overall partition cost can now be expressed as $\pi_n(K, P)$ where $P$ has to be distributed among all the $K$ regions. The optimal substructure equation can be rewritten as:

$$\pi^* = \pi_n(K, P) = \min \{\min \{\max \{\pi_n(K-x, p), \pi_n(x, P-p)\}\}\},$$

where $0 < x \leq K$, $0 < p < P$ and $\pi$ varies over all possible positions of the separator being placed. This equation can be further extended to consider the cases of horizontal and vertical separator, as in Equation (4), but is being omitted here for the purpose of brevity. For solving this problem, we evaluate the optimal solution using the same approach as in Section IV. The dimension of array $M$ is increased to six such that $M[i, j, l, w, k, p]$ stores the optimal partition cost of region $[i, j, l, w]$, if the partition size is $k$ and the number of pins assigned to this region is $p$. For each value of $k$, we add one more loop that iterates over all the pin-counts possible. Moreover, when a separator is placed, in addition to varying the size of partitions of the subregions $A$ and $B$, all possible distributions of the $p$ pins to the two subregions are evaluated. For any region $R$, $\pi_R(k, p)$ is not evaluated for $p < k$, because each region should be allotted at least one pin. We use the equality, $\pi_R(k, p) = \pi_R(k, p - 1)$, if $p$ takes values that are greater than $k$ times the flit width. The array $B$ is also a six-dimensional structure now with each element being a six-tuple structure. Two more tuples are added to specify the pins allotted to each subregion.

The run-time complexity of the proposed method is $O(mn^2K^2P^2)$ if $m \geq n$ and $O(m^2nK^2P^2)$ otherwise, as shown in [33]. The space complexity is $O(mn^2KP)$ [33].

C. Power constraints

The reduction of test application time by manipulating power profiles of test sets has been studied in [34]. Given a power constraint, the problem of minimizing the test time is presented in [34]. A similar problem in the context of NOC has been formulated and solved using ILP techniques in [35]. The scheduling problem under power constraints was shown to be computationally harder than the general test scheduling problem in [35], and the proposed method does not scale well with the number of cores. This section elaborates on ideas presented in [34] to initially treat power as a design objective to minimize power consumption, and then subsequently as a design constraint to achieve minimization of the test application time.

**Profiling of power consumption:** Accounting for power during test scheduling entails the modeling of power consumed by the individual cores. A simplistic approach is to flatten the power profile of a core to the worst-case instantaneous power consumption value, i.e., its peak value [34]. The simplicity and reliability of this model, called the global peak power approximation model (GP-PAM), is achieved at the cost of including significant false power in the model; false power is the power that is not consumed, but still being considered. This limits the degree of test concurrency that can be ideally achieved. The GP-PAM model is represented by a pair $[P_{global}, L_{global}]$, where $P_{global}$ is the global peak power consumed over a test length of $L_{global}$ cycles. Another model, proposed in [34], significantly reduces false power that enables greater test concurrency. This model, known as 2LP-PAM, creates a profile having two peaks, as opposed to a single global peak in GP-PAM. 2LP-PAM is represented by a vector pair $[P_{hi}, L_{hi}, P_{lo}, L_{lo}]$. The entire test length of a core is divided into two segments; $P_{hi}$ is the peak power for a test length of $L_{hi}$, peak power $P_{lo}$ for a test length of $L_{lo}$, and $P_{hi} \geq P_{lo}$. The splitting of the power profile provides more flexibility in scheduling cores under a power constraint because it approximates false power more accurately than the GP-PAM model.

We adapt the benefits of 2LP-PAM in our DP approach, and discuss how to construct power profiles of a region — a collective profile of all the cores present in the region.
P cores.

we first select the region with lower test time (P the list that minimizes the area under the resulting profile.

ff test lengths under di bigger rectangle under a given power constraint to minimize

power profiles of two regions to obtain power profile of the 

cores present in a leaf region

PL that the resultant profile does not exceed the power limit

The shape of the resulting profile depends on the values of the peaks of the profiles that are merged, and the power constraint. For our running example, the merged profile can look like the profile with four peaks as shown in Figure 13.

To be consistent with the two-peak power model, the number of peaks has to be reduced to two. The transformation for this case is shown in Figure 14. In 14(a), different areas are marked that have to be collapsed with the original profile to obtain two peaks. Depending on the magnitude of the areas marked, the final profile can look like Figure 14(b) or Figure 14(c). The decision on which areas to collapse depends on the area under the profile, and the profile with the least area is chosen.

We next discuss the impact of power constraints on the merging of profiles. In the case when a power violation occurs, the profile for B is shifted until no violation is caused. If the two lower peaks from the two regions exceed the power limit PL, i.e., $P_{lo}^A + P_{lo}^B > PL$, then the two power profiles cannot overlap. In such a case, we sort the four peaks in descending order and place them side-by-side to create a profile shown in Figure 15(a). Different combinations of the marked areas are grouped together to reduce the number of peaks to two. The final profile can be any one of the profiles as shown in 15(b), 15(c) and 15(d).

We have implemented a dedicated transformation procedure for all kinds of profiles that can result from the merge operation. The enumeration of these cases and the corresponding transformations are straightforward and therefore omitted. Integration with DP: The proposed method for power profile manipulation has been integrated with our DP approach. When a separator is placed in a region, rather than taking the maximum of the testing times of the two subregions as the test time for that region, we compute the new test time as the test length of the power profile obtained after merging the power profiles of the two subregions. Line 6 in Figure 11 is modified accordingly. Since any two power profiles are merged without violating power constraint, power profile of the NOC N does not violate the constraint.

The run-time complexity of the approach under power constraints remains unaffected. The procedure to merge two profiles takes a constant time for computing a merged profile. When creating a power profile for a leaf region using createPowerProfile procedure, the complexity increases by a factor of $mn$ (an efficient implementation using DP increases the complexity by a factor of $\min(m,n) \cdot \log(mn)$), but this part only constitutes the initialization step and does not dominate the runtime of the main algorithm.

D. Reducing the run-time complexity by a factor of $P$

The run-time complexity of the proposed approach was found to be a function of $P^2$. The number of pins $P$ is much larger in magnitude when compared to other parameters such as $m,n$, or $k$. We next show how the factor of $P^2$ can be reduced to $P$. The approach presented in the previous section distributes pins on both sides of the separator such that the sum of the pins on both sides is equal to a given pin count $p$, and selects the distribution that minimizes test time. All possible combinations are tried making the search exhaustive. This is repeated for all possible values of $p$; therefore, a factor of $P^2$ is seen in the computational complexity. Exhaustive search can be reduced to selective search by observing that given a
**computeAndStore**($M, i, j, l, w, k$)

1: **for** $v$ in $i + 1 \cdots i + w - 1$, $x$ in $1 \cdots k - 1$
   
   /* This is the optimal distribution for the base case ($p = k$). $p_1$ pins to one subregion and $p_2$ to the other. */
2:    $p_1 \leftarrow x$; $p_2 \leftarrow k - x$;
3: **for** $p \leftarrow k$ to $P$
4:    $t_1 \leftarrow M[i, j, l, v - i, x, p_1]$
5:    $t_2 \leftarrow M[v, j, l, i + w - v, k - x, p_2]$
6:    $t \leftarrow \max(t_1, t_2)$
7: **if** $M[i, j, l, w, k, p] > t$
8:    $M[i, j, l, w, k, p] \leftarrow t$.
9: $B[i, j, l, w, k, p] \leftarrow [v, v', x, k - x, p_1, p_2]$

Assigning the next pin to the subregion having greater test time.

10: **if** $t_1 \geq t_2$ **then** $p_1++$ *else* $p_2++$

Fig. 16. Modified computeAndStore for reducing the run-time complexity.

This is the optimal distribution for the base case ($p = k$). $p_1$ pins to one subregion and $p_2$ to the other. For the base case when the pin-count $p$ equals $k$, the number of pins assigned to the subregions equals their respective partition sizes, i.e., $x$ and $k - x$. This is the only possible (and hence optimal) distribution because each leaf region has to be driven by at least one pin. The arrays $M$ and $B$ are updated whenever a better result is found. A similar procedure is used for sweeping horizontal separators.

### VI. Experimental Results

In this section, we first describe how we created our test cases. The rest of the section includes the following:

- For NOCs with arbitrary (irregular) topologies, we compare the results obtained from our approach based on subset-sum with an implementation of [13].
- For a mesh-based NOC, we compare the results obtained using DP to that obtained with ILP [20]. Since the comparison is based on the problem instances that can be solved with ILP in a reasonable time, we take small problem instances first. These test cases provide a good baseline with which we compare the quality of solutions produced by our approach.
- We compare the CPU times required for the proposed method with two baseline methods.
- We demonstrate diminishing returns on increasing the partition size (number of regions) for a given pin count.
- Results showing the effect of power constraints on the total test time is reported and compared with [35].
- Improvements in the computation time due to the speedup technique discussed in Section V.

#### A. Test scheduling without power constraints

We used six SOCs from the ITC’02 SOC Test Benchmarks [36], namely d695, g1023, p34392, p22810, t512505 and p93791, with 10, 14, 19, 28, 31 and 32 cores, respectively. Since we are interested in evaluating performance for a large number of cores, we created additional SOCs by taking cores from the last two SOCs and replicating them. The test times for all TAM widths (constrained by floor width) for each core were obtained using the design wrapper algorithm [6].

For mesh-based NOCs, we adopted the same approach as outlined in [14], [15], [18]–[20] for generating the topology of the NOC. We assumed XY-routing, a switching delay of three clock cycles per router for access point-to-core or core-to-access point path establishment, and one cycle each for transmitting header and tail flits. A flit width of 32 bits was assumed. For each leaf region, we placed an access point at the middle of its longest boundary edge, and computed the cumulative routing delay of the region as the sum of routing delays for all the cores located in that region. It can be shown that, by taking the derivative of the routing delay and equating it to zero, the routing delay is minimized when the access point is placed at this specific location of the region. The routing delay for a core during path establishment is three times its Manhattan distance from the access point assigned to that core. We executed the ILP model using the FICO XPress-MP Solver [37] that was also used in [20].

Each region contains exactly one access point, hence the number of regions (partition size) in our work and the number of access points, as reported in [20], are synonymous. All results were obtained on a 16-core Intel(R) Xeon(R) machine with a processor speed of 2.53 GHz, 64 GB memory and a cache size of 12288 KB.

In the first experiment, we compare our method for an NOC with an arbitrary topology with the method presented in [13]. Since the latter is based on TR-Architect [27], it does not restrict the number of access points (ATE interfaces) that are used; it reports the number of access points that are required to minimize the test time. We modify ModifiedCreateStartSolution procedure to initialize a solution with a fixed number of access points ($K$). Table I shows the results for the two methods for two NOC test cases. The column $K^*$ does not restrict the number of access points for the method of [13]. The column *$K$ ’ reports the number of access points required ($K^*$) along with the minimized test time. We use $K^*$ to derive the test time for the proposed method, which is shown in the last column. The NOC examples (irreg1 and irreg2) are taken from [13] and they both consist of nine routers. Each router is assigned a core taken from the SOC d695. Since there are nine routers and d695 consists of 10 cores, we ignore one core, namely Module5, for this experiment. Note that the greedy procedure of pin distribution is optimal at every iteration; hence, solutions are obtained for a range of pin counts in a single run. The CPU time was found to be negligible in all cases. Our method consistently reports better solutions than [13].

Next we present results for a mesh-based NOC. We took a 6x6 NOC obtained from the SOC t512505 (31 cores) and compared the results obtained using the two approaches. The ILP proposed in [20] does not address the problem of optimal access point placement. Therefore, we show two different sets of results for the ILP model; one obtained by a random
TABLE I

<table>
<thead>
<tr>
<th>Design</th>
<th>K = 2</th>
<th>K = 3</th>
<th>K*</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>33859</td>
<td>31588</td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td>28226</td>
<td>21412</td>
<td>T</td>
</tr>
<tr>
<td>32</td>
<td>20516</td>
<td>19372</td>
<td>T</td>
</tr>
<tr>
<td>40</td>
<td>18940</td>
<td>17704</td>
<td>T</td>
</tr>
<tr>
<td>48</td>
<td>18793</td>
<td>17116</td>
<td>T</td>
</tr>
<tr>
<td>56</td>
<td>18793</td>
<td>15773</td>
<td>T</td>
</tr>
<tr>
<td>64</td>
<td>18793</td>
<td>15760</td>
<td>T</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Design</th>
<th>K = 2</th>
<th>K = 3</th>
<th>K*</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>33859</td>
<td>31588</td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td>28226</td>
<td>21412</td>
<td>T</td>
</tr>
<tr>
<td>32</td>
<td>20516</td>
<td>19372</td>
<td>T</td>
</tr>
<tr>
<td>40</td>
<td>18940</td>
<td>17704</td>
<td>T</td>
</tr>
<tr>
<td>48</td>
<td>18793</td>
<td>17116</td>
<td>T</td>
</tr>
<tr>
<td>56</td>
<td>18793</td>
<td>15773</td>
<td>T</td>
</tr>
<tr>
<td>64</td>
<td>18793</td>
<td>15760</td>
<td>T</td>
</tr>
</tbody>
</table>

TABLE III

<table>
<thead>
<tr>
<th>K</th>
<th>P</th>
<th>Test time (clock cycles), T</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>663388</td>
<td>1.24 s</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>615042</td>
<td>11.29 s</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>473990</td>
<td>4.6 s</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>605224</td>
<td>2 m</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>461053</td>
<td>12.22 s</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>458132</td>
<td>1 m</td>
</tr>
</tbody>
</table>

ILP*: ILP with random placement of access points.
ILP**: ILP with a placement of access points guided by the proposed technique.
ILP †: ILP with placement of access points.

The ILP solver was allowed to execute until it found an optimal solution. Note that an “optimal solution” for ILP cannot always be achieved due to bottleneck cores.

The CPU time for DP is less than 1 s.

The results for different values of K and P are shown in Table II. The column \( \Delta^* \) (\( \Delta^{**} \)) records the relative difference in test time reported by dynamic programming (DP) over that obtained from ILP (ILP*). If the test time obtained by ILP is \( T_{ILP} \) and that obtained from DP is \( T_{DP} \), then \( \Delta^* = \frac{T_{DP} - T_{ILP}}{T_{ILP}} \times 100\% \), and \( \Delta^{**} = \frac{T_{DP} - T_{ILP}}{T_{ILP}} \times 100\% \).

Negative values of \( \Delta^* \) indicate that the solutions obtained from ILP are worse than that from DP. The same experiment is repeated for another 6x6 NOC obtained from the SOC p93791. The corresponding results are tabulated in Table III. The CPU time for DP is less than 1 s.

The ILP solver was allowed to execute until it found an optimal solution. Note that an “optimal solution” for ILP cannot always be achieved due to bottleneck cores.

The CPU time taken by ILP is clearly impractical, and the test time obtained from ILP in the test time reported by dynamic programming (DP) over that obtained from ILP (ILP*). Since the lower-bound expression in [27] is not tied to any particular TAM design and utilizes only the volume of test data that must be transported, these bounds are also applicable to the test-time minimization problem in NOCs.

We show similar results for a larger 14x14 NOC obtained by replicating cores from the SOC benchmark p93791; see Table V. We set a time limit of three hours for ILP for each value of K and P, and report the best intermediate results obtained within that time limit. It was observed that with an increase in the number of regions K, ILP took longer time to report the first intermediate solution. We set three hours as the limit because no noticeable improvement was seen in the ILP intermediate solutions after this duration. The CPU time taken by DP is only 4 minutes and 8 seconds, which is negligible in comparison to the cumulative execution time of 1 day and 12 hours taken by ILP for all the 12 cases shown in Table V. Our approach also reported better results for some larger values of K. Note that for instances for which ILP yields lower test times than DP, a combination of the two methods can be used. An effective partition can be first identified using DP, and then the test-pin assignment problem can be solved using ILP, as in [20]. However, for larger problem instances, ILP is not feasible due to high computation requirements.

We next show results for a 20x20 NOC obtained by replicating cores from the two SOC benchmarks; see Table VI. Considering the size of NOC, a CPU time limit of 6 hours was set for ILP for each of the 12 cases shown in Table VI. No intermediate solutions were reported for larger values of K. While the cumulative CPU time taken by ILP for all the cases was found to be 2 days and 7 hours, ILP** took 3 days. The CPU time taken by ILP is clearly impractical, and the ILP approach does not scale with the number of cores and the size of the partition. In contrast, the CPU time for DP is only 25 minutes and 10 seconds.
To further demonstrate the scalability and benefits of the proposed approach, we evaluated the DP method for an SOC of the future that has nearly 1,000 cores (a 32x31 NOC). The DP procedure completes in 4 hours of CPU time. In order to evaluate the quality of the solution (test time obtained), we developed a simple baseline heuristic of generating a partition. Among all intermediate regions available for partitioning, the region having the largest number of network tiles was selected and a separator was placed randomly. Pins were distributed in the ratio of the dimension of the leaf regions. We generated 100 such partitions for each different values of \( K \) and a value of \( P = 150 \), and report the mean, minimum, and maximum test time for each case, as shown in Table VII. It can be seen from the table that DP provides consistently superior results—two orders of magnitude reduction in test time compared to the mean test time for the baseline case. Compared to the minimum test time for the baseline case, the test time reduction is in the range of 13% to 48%.

We also considered an SOC with 1,600 cores (40x40 NOC). The DP solution (for \( 2 \leq K \leq 5 \)) was obtained in 3 hours of CPU time for \( P = 150 \). The test times obtained from DP was consistently lower than that for the randomized baseline method for all values of \( K \). For example, for \( K = 5 \), we obtained 19.80%, 44%, and 69.33% reduction in test time compared to the minimum, average and maximum test times, respectively, obtained from the baseline method.

We also examined the scalability of the subset-sum-based method for large SOCs. We ran the procedure optimize_TAM on the 32x31 NOC for \( K = 4 \). Figure 17 shows the percentage reduction in test time reported by the procedure over the DP-based method for varying values of \( \Delta \) and \( P \). The figure also shows the CPU time needed by the procedure for each value of \( \Delta \). It can be seen that when \( \Delta \) is high, the subset-sum-based method is capable of producing better results than the DP, but takes more CPU time. The reduction in test time was found to be as high as 3.3% when \( \Delta \) was set to 20000. The test time reported by the procedure optimize_TAM was 1.4% more than that for DP for \( \Delta = 5000 \). The CPU time varied from 7.2 hours to 1.5 hours as \( \Delta \) was swept from high to low.

### Table IV

Comparison of test times (in cycles) for p93791 obtained from different methods with the TAM-independent lower bound reported in [27].

<table>
<thead>
<tr>
<th>( K )</th>
<th>( P )</th>
<th>( DP )</th>
<th>ILP/ILP*</th>
<th>Lower bound [27]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>48</td>
<td>601357</td>
<td>604344</td>
<td>580744</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>479515</td>
<td>457362</td>
<td>435561</td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td>415483</td>
<td>400795</td>
<td>387167</td>
</tr>
<tr>
<td>5</td>
<td>96</td>
<td>329886</td>
<td>310467</td>
<td>309378</td>
</tr>
</tbody>
</table>

### Table V

Test times for 14x14 NOC for various values of \( K \) and \( P \).

<table>
<thead>
<tr>
<th>( K )</th>
<th>( P )</th>
<th>Test time (clock cycles), ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>ILP*</td>
<td>DP</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>3715943</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>2857587</td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td>2491517</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>1917893</td>
</tr>
<tr>
<td>7</td>
<td>108</td>
<td>1662490</td>
</tr>
<tr>
<td>8</td>
<td>120</td>
<td>1541735</td>
</tr>
<tr>
<td>9</td>
<td>108</td>
<td>1766538</td>
</tr>
<tr>
<td>10</td>
<td>120</td>
<td>1679271</td>
</tr>
<tr>
<td>11</td>
<td>108</td>
<td>1672190</td>
</tr>
<tr>
<td>12</td>
<td>140</td>
<td>1371750</td>
</tr>
<tr>
<td>13</td>
<td>140</td>
<td>1768660</td>
</tr>
<tr>
<td>14</td>
<td>152</td>
<td>1391710</td>
</tr>
</tbody>
</table>

ILP*: ILP with random placement of access points.
ILP**: ILP with placement of access points guided by the proposed technique.

**Table VI**

Test times for 20x20 NOC for various values of \( K \) and \( P \).

<table>
<thead>
<tr>
<th>( K )</th>
<th>( P )</th>
<th>Test time (clock cycles), ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>ILP*</td>
<td>DP</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>3912011</td>
</tr>
<tr>
<td>4</td>
<td>56</td>
<td>34327589</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>3299309</td>
</tr>
<tr>
<td>6</td>
<td>72</td>
<td>28623706</td>
</tr>
<tr>
<td>7</td>
<td>84</td>
<td>25376412</td>
</tr>
<tr>
<td>8</td>
<td>96</td>
<td>23671502</td>
</tr>
<tr>
<td>9</td>
<td>108</td>
<td>22563287</td>
</tr>
<tr>
<td>10</td>
<td>120</td>
<td>19147600</td>
</tr>
<tr>
<td>11</td>
<td>120</td>
<td>12901854</td>
</tr>
</tbody>
</table>

ILP*: ILP with random placement of access points.
ILP**: ILP with placement of access points guided by the proposed technique.

**Table VII**

Test time of dynamic programming vs. randomized baseline approach for 32x31 NOC, \( P = 150 \).

<table>
<thead>
<tr>
<th>( K )</th>
<th>Test time obtained by DP, ( T_1 )</th>
<th>Test time from Baseline Approach, ( T_2 )</th>
<th>( (T_1 - T_2) / T_2 \times 100 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>48696503</td>
<td>93538264</td>
<td>56438372</td>
</tr>
<tr>
<td>5</td>
<td>39058309</td>
<td>78704175</td>
<td>53480028</td>
</tr>
<tr>
<td>6</td>
<td>32955187</td>
<td>73151835</td>
<td>47784917</td>
</tr>
<tr>
<td>7</td>
<td>29912627</td>
<td>78841592</td>
<td>48190525</td>
</tr>
<tr>
<td>8</td>
<td>27623723</td>
<td>8255284</td>
<td>52717461</td>
</tr>
</tbody>
</table>

**Table VIII**

Table showing diminishing improvement in the reduction in test time as \( K \) increases for a 14x14 NOC.
For test parallelism. We report lower test times than \[13\] using to the associated problem of power consumption because of practice. Moreover, a large number of access points can lead to be used. It can be seen that \[13\] reports an extremely large points. Since our approach only creates rectangular partitions, 37.6% when \(K\) improvement achieved by our method over \[13\] is as high as 160. However, the number of available pins on the ATE is limited, hence it is natural to ask what is a suitable choice for the partition size and the pin-count that should be used, and how can we calculate these values. These questions will be addressed in future work.

We next examine our experimental results for further analysis. Table VIII reports the test times for a 14x14 SOC (196 cores) for partition sizes \(K\) varying from 3 to 8, and for three different values of \(P\). The table also shows the results produced by [13] for all these cases. The TTR (test-time reduction) column shows the relative reduction of test time obtained by adding an additional access point (using the DP approach). It can be seen that the magnitude of reduction in test time gradually decreases. This is because the test time of a core depends on the number of pins assigned to it and as the partition size increases, the number of pins available per region decreases. By increasing the pin count, we observe that the effect of sudden decrease in TTR can be moderated. For example, for \(P = 80\), the TTR rapidly dipped to 0%, but we were able to moderate the sudden decline by allotting 120 pins, and get further benefits by increasing the pin count to 160. However, the number of available pins on the ATE is limited, hence it is natural to ask what is a suitable choice for the partition size and the pin-count that should be used, and how can we calculate these values. These questions will be addressed in future work.

For test parallelism. We report lower test times than \[13\] using to the associated problem of power consumption because of practice. Moreover, a large number of access points can lead to be used. It can be seen that \[13\] reports an extremely large points. Since our approach only creates rectangular partitions, 37.6% when \(K\) improvement achieved by our method over \[13\] is as high as 160. However, the number of available pins on the ATE is limited, hence it is natural to ask what is a suitable choice for the partition size and the pin-count that should be used, and how can we calculate these values. These questions will be addressed in future work.

The row \(K^*\) in Table VIII shows the result produced by [13] when no restriction is placed on the number of access points to be used. It can be seen that [13] reports an extremely large number of access points, which can be harder to implement in practice. Moreover, a large number of access points can lead to the associated problem of power consumption because of test parallelism. We report lower test times than [13] using fewer access points. For \(P = 160\) (not shown in the table), the improvement achieved by our method over [13] is as high as 37.6% when \(K\) is restricted to 8. When \(K\) is not restricted, [13] resulted in a test time (using 23 access points) that is worse than the test time reported by DP with only 7 access points. Since our approach only creates rectangular partitions, a simple post-processing step, such as that implemented in the procedure ModifiedReshuffle of [13], can further reduce test time by moving cores from one region to another. We also report lower bound values for the two values of \(P\) in the last row of Table VIII. The test times that we obtained are only 9.4% (12.5%) larger than the provable lower bounds for \(P = 80\) \((P = 120)\).

### B. Power-constrained test scheduling

To assess the impact of power constraint on test scheduling, we ran our approach on two NOCs: a 6x6 NOC and an NOC with 100 cores (10x10), both constructed out of cores from the benchmark circuit p93791. Due to the lack of information on power consumption of these cores, we assumed that the power consumption in a core is directly proportional to the sum of the number of core’s inputs, outputs, bidirectional pins and memory elements — the same approach as adopted in [35]. All values for power consumption used were relative with respect to the total sum of the power consumption of all cores, which is referred to as “system” power consumption in [35]. We therefore refer to power in terms of a normalized value relative to the total system power.

Table IX compares the test length obtained by our approach with that obtained using the ILP model from [35]. All power constraints are defined as a fraction of the system power. Scheduling with the 1.0 power constraint is equivalent to scheduling without power constraints, as no schedule can exceed the total system power. Since our approach approximates the power consumption for a set of cores using manipulation on power profiles to create an approximate profile, the performance of the approach depends on how tightly the approximation scheme bounds the actual power profile from above. The test lengths were found to match closely with the results obtained using [35] for all values of the power limit.

Because our approach is necessitated by the intractability of problem instances involving large NOCs, we present the results for a 10x10 NOC in Table X for different power constraints and partition sizes. Since, as in this case, each core contributes very little to the system power consumption, the power constraint was set to 25% of the system power consumption at first, and then subsequently the power budget was reduced by 5% at each step. Increasing the power constraint always increases the test length. Under strict power constraints, additional access points will only increase test time compared to relaxed power constraints. The run-time complexity remains the same as before, and no appreciable difference in runtime was found for the reported cases.

### TABLE IX

<table>
<thead>
<tr>
<th>Power Test Length</th>
<th>Test Length from [35]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>64 pins</td>
</tr>
<tr>
<td>1.00</td>
<td>484201</td>
</tr>
<tr>
<td>0.45</td>
<td>485869</td>
</tr>
<tr>
<td>0.35</td>
<td>487541</td>
</tr>
<tr>
<td>0.30</td>
<td>568107</td>
</tr>
</tbody>
</table>

1 Power constraint relative to total power consumption in SOC.
TABLE X

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Original DP</th>
<th>Improved DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 x 10 (K = 8, P = 152)</td>
<td>2 min 5 s</td>
<td>9 s</td>
</tr>
<tr>
<td>14 x 14 (K = 8, P = 152)</td>
<td>4 min 8 s</td>
<td>15 s</td>
</tr>
<tr>
<td>20 x 20 (K = 8, P = 152)</td>
<td>25 min 10 s</td>
<td>1 min 16 s</td>
</tr>
<tr>
<td>32 x 31 (K = 8, P = 150)</td>
<td>4 h</td>
<td>11 min 50 s</td>
</tr>
</tbody>
</table>

TABLE XI

<table>
<thead>
<tr>
<th>PC</th>
<th>Number of access points (K)</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>993225</td>
<td>772398</td>
<td>750797</td>
<td>748576</td>
<td>749113</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>993225</td>
<td>772398</td>
<td>750797</td>
<td>748576</td>
<td>752042</td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td>1005859</td>
<td>869850</td>
<td>799056</td>
<td>795940</td>
<td>848257</td>
<td></td>
</tr>
<tr>
<td>0.15</td>
<td>1223566</td>
<td>961602</td>
<td>980472</td>
<td>1036158</td>
<td>1036158</td>
<td></td>
</tr>
</tbody>
</table>

1 Power constraint relative to total power consumption in SOC.

C. Speedup technique

We next show the effect of the speedup technique, discussed in Section V, on the computation time for DP. In Table XI, the third column corresponds to the approach taken for reducing the run-time complexity by a factor of P. The speedup is clearly evident for larger NOCs.

VII. CONCLUSION

We have developed a scalable solution to the problem of optimizing test-data delivery in an NOC-based manycore SOC. A formulation based on the subset-sum problem has been proposed for NOCs with dedicated routing and arbitrary topologies. For grid topologies supporting XY routing, test-time minimization has been solved using DP, which computes optimal solutions for rectangular partitions. Results for NOC-based manycore SOCs constructed from ITC 2002 benchmarks have shown that the proposed method yields high-quality results, and scale to large SOCs with many cores. Test scheduling under power constraints and a speedup technique have been incorporated. Since dynamic programming solutions are recursively constructed from solutions to underlying subproblems, the proposed method can inherently facilitate design-space exploration for effective test planning.

REFERENCES