基于云模型进化算法的硅通孔数量受约束的 3D NoC 测试规划研究

许川佩 陈家栋* 万春霆
(桂林电子科技大学电子工程与自动化学院 桂林 541004)

摘 要：针对硅通孔(TSV)价格昂贵、占用芯片面积大等问题，该文采用基于云模型的进化算法对 TSV 数量受约束的 3 维片上网络(3D NoC)进行测试规划研究，以优化测试时间，并探讨 TSV 的分配对 3D NoC 测试的影响，进一步优化 3D NoC 在测试模式下的 TSV 数量。该方法将基于云模型的进化算法、小生境技术以及遗传算法的杂交技术结合起来，有效运用遗传、优胜劣汰以及保持群落的多样性等理念，以提高算法的寻优速度和寻优精度。研究结果表明，该算法既能有效避免陷入局部最优解，又能提高全局寻优能力和收敛速度，缩短了测试时间，并且优化了 3D NoC 的测试 TSV 数量，提高了 TSV 的利用率。

关键词：3 维片上网络；硅通孔；云模型；进化算法

中图分类号：TN407 **文献标识码**：A **文章编号**：1009-5896(2015)02-0477-07

DOI: 10.11999/JEIT140165

Research on Test Scheduling of 3D NoC under Number Constraint of TSV (Through-Silicon-Vias) Using Evolution Algorithm Based on Cloud Model

Xu Chuan-pei Chen Jia-dong Wan Chun-ting
(*School of Electronic Engineering and Automation*, *Guilin University of Electronic Technology*, *Guilin* 541004, *China*)

Abstract: As Through-Silicon-Vias (TSVs) in three-Dimensional Network-on-Chip (3D NoC) accompany some overhead such as the cost and the area, in order to optimize the number of TSVs of 3D NoC in test mode and reduce the test time, a new method using evolution algorithm based on cloud model is proposed to research on the test scheduling of 3D NoC and the impact of TSVs number and their allocation in each layer on 3D NoC test. This method combines the cloud evolution algorithm with niche technology and hybridization technique in genetic algorithm. It uses effectively the concepts of heredity, natural selection and community diversity to improve the quality of the algorithm on optimizing speed and precision. Experimental results demonstrate that the proposed method can not only effectively prevent from running into local optimization solution, but also improve the ability and speed of searching the best solution, and that TSVs number of 3D NoC can be optimized to improve the TSVs' utilization.

Key words: Three-Dimensional Network-on-Chip (3D NoC); Through-Silicon-Via (TSV); Cloud model; Evolution algorithm

1 引言

随着集成电路技术的发展，3 维片上网络(Three-Dimensional Network-on-Chip, 3D NoC)的思想逐步出现在集成电路设计中。3D NoC 通过硅通孔(Through-Silicon-Via, TSV)技术将各层 2 维结构芯片互连起来，其优点表现为：一是缩短了全局互连；二是降低了延迟，提高了系统性能；三是由于缩短了连接线的长度而降低了功耗；四是增加了封装密度，减小了芯片面积[1]。然而，由于 TSV 占用了芯片大量区域(特别是为 TSV 需要而留出的“禁用区”)，使得芯片上的 IP 核数量大大减少，另外，生产 TSV 的开销很大[1]，这些因素使得设计者在设计 3D NoC 时需要考虑 TSV 数量问题，而对于数量受约束的 3D NoC 测试，需要有针对性的测试方案，从而提高 TSV 的利用率，优化测试时间。

目前对于 3 维集成电路的测试规划研究，文献[2]提出了在各层均衡分布 IP 核，并对 IP 核进行测试调度的方法，减小了芯片的测试时间；文献[3]提出了在 TSV 数量受约束条件下，如何减少 3D SoC 中分立电路测试时间的方法，但不能保证优化全局的测试时间；文献[4]研究了在 TSV 数量一定的情

2014-01-24 收到，2014-04-24 改回
*通信作者：陈家栋 cljxdm@163.com

况下减少 3D SoCs 测试时间的方法，以及提出优化 TSV 数量的方法。国内外学者针对 NoC 的测试研究主要集中在 2D NoC，例如文献[5]探讨了 NoC 重用对系统测试的影响，并提出了抢占式和非抢占式测试调度的测试方法，但该方法缺乏快速有效的规划算法；文献[6]在文献[5]的研究基础上，将 NoC 测试规划与遗传算法结合起来，从而优化了测试时间。

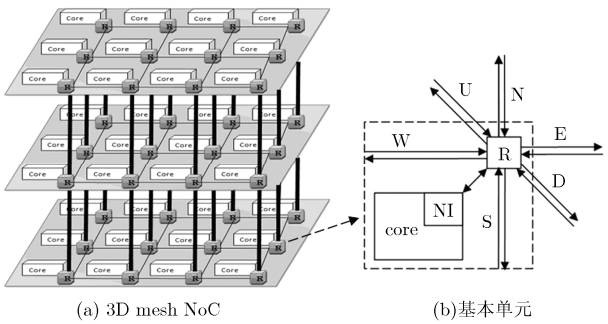
由于 3D NoC 的测试规划问题是一个 NP 难问题，本文基于高精度寻优的云模型进化算法，研究 3D NoC 测试资源(I/O 端口，TSV)分配策略，以提高资源利用率，优化测试时间，并探讨 TSV 数量及其在各层间的分配对 3D NoC 测试的影响。

2 3D NoC 结构

3D NoC 结构主要有 3D mesh 结构，蝶形胖树结构，XnoTs 结构等。目前学者主要研究的是 3D mesh 结构。典型的 3D mesh 结构 NoC 如图 1(a)所示，相邻两层对应的上、下两个路由节点均由 TSV 连接起来。

3D NoC 的一个基本单元如图 1(b)所示。其中 R 表示路由节点(Router)，每个路由节点有 7 对 I/O 端口，其中一对端口与本地芯核(core)相连，其余 6 对分别与相邻的东(E)，南(S)，西(W)，北(N)，上(U)以及下(D)6 个方向的路由节点相连，垂直方向 U 和 D 的互连线由 TSV 构成，U 和 D 方向均包含 up-to-down TSV 和 down-to-up TSV[7]，为了便于研究，文中称一个垂直方向含有 1 束 TSV。互连的两个节点之间的通信可以采用全双工的通信方式。core 表示一个资源节点，可以是处理器内核、存储器、FPGA 或 IP 核等。NI 表示网络接口，是路由节点与资源节点的通信接口。

很明显，TSV 越多，各层资源节点之间的通信就越便捷。然而，由于芯片生产开销以及芯片面积的限制，3D NoC 在设计和使用时需要考虑 TSV 数量问题。本文在 TSV 数量受约束的前提下对 3D NoC 进行测试规划研究。



(a) 3D mesh NoC (b)基本单元

图 1 3D mesh 结构 NoC 及其基本单元

3 TSV 数量受约束的 3D NoC 测试规划

3.1 NoC 测试规划

为减少硬件开销，NoC 测试采用重用片上网络的测试访问机制[6,8,9]，即复用片上网络的路由节点和通道(channel)等资源，实施并行测试以提高测试效率。在测试 core 时，测试矢量由输入端口送入网络，按一定的路由算法到达 core，生成的测试响应以同样的路由算法到达输出端，由外部的测试响应接收并分析，完成一个核的测试。

本文采用非抢占式的测试策略，给每个测试核分配一条测试路径，包括输入端口、输出端口，以及相应的通道，一旦某测试核的测试路径确定之后，该路径上的所有资源(输入/输出端口，通道)将预留给该测试核，其他核的测试矢量以及测试响应不得抢占，直到该核测试完毕为止。

TSV 数量受约束的 3D NoC 测试时，测试数据需要经过特定的 I/O 端口以及特定的 TSV 才能完成芯核的测试，因此，本文研究的测试规划问题可描述为：给定 N_{C} 个待测核 core，N_{IO} 对 I/O 端口，各层的 TSV 数量，在确定的路由算法下，如何将输入/输出端口，以及 TSV 合理有效地分配给各个 core，确定这些 core 的测试路径，并且在功耗允许的条件下，无冲突地调度各个待测核的测试顺序，使整体测试时间最小。

3.2 路由算法及路径冲突

在测试过程中，采用 XYZ 路由算法，数据包由源节点沿 X 方向传输，到达与目标节点 x 坐标相同的路由节点后，转向 Y 方向进行传输，到达与目标节点 y 坐标相同的路由节点后转向 Z 方向传输，直至目标节点。

根据已制定的测试策略，一旦给某个 core 分配了输入/输出端口以及 TSV 等资源，那么按照 XYZ 路由算法，该 core 的测试路径便是确定的。由于多个 core 同时进行测试，如果芯核测试顺序调度不合理，测试路径便有可能产生冲突，即两个 core 的测试数据在某个路由节点处向相同路由节点进行传输。

3.3 测试时间以及功耗约束

(1)测试时间　多对端口并行测试规划如图 2 所示，总测试时间取决于测试时间最长的那对 I/O 端口。

总测试时间为

$$T=\max_{0\le j\le N_{\mathrm{IO}}-1}\left(\sum_{i=0}^{N_{\mathrm{C}}-1}T_i\cdot x_{ij}+T_{\mathrm{kx}}(j)\right) \tag{1}$$

$$x_{ij}=\begin{cases}1\text{，core } i \text{ 分配到第}j\text{对I/O 端口}\\0\text{，其它}\end{cases} \tag{2}$$

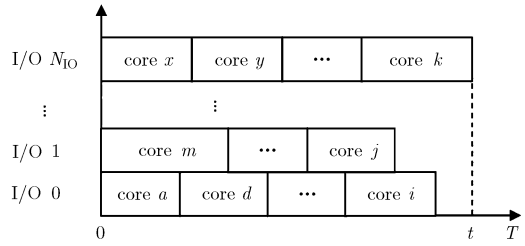


图 2 多对端口并行测试规划图

其中，T_i 代表 core i 分配到第 j 对 I/O 端口上的测试时间，N_C 代表 core 的数量，$T_{kx}(j)$ 为第 j 对 I/O 端口上测试资源冲突或功耗过大时的空闲时间，N_{IO} 为 I/O 端口对数，单个核的测试时间 T_i 包括测试数据的传输时间以及测试所消耗的时间，即 $T_i = T_{tr} + T_{te}$，而在核进行测试的同时，其余数据包已相继传入网络，因此数据传输所耗的时间主要是第 1 个测试数据包从输入端口至测试核以及最后 1 个测试响应数据包从测试核至输出端口所耗的时间。

$$T_{tr} = \mathrm{hop_i} \cdot T_p(\mathrm{first}) + \mathrm{hop_o} \cdot T_p(\mathrm{last}) \tag{3}$$

式中 $T_p(\mathrm{first})$，$T_p(\mathrm{last})$ 分别为第 1 个测试数据包和最后 1 个测试响应数据包在两个路由节点之间的传输时间，$\mathrm{hop_i}$ 和 $\mathrm{hop_o}$ 分别代表数据包输入以及输出测试核所经过的路由跳数。

$$T_p = T_h + T_r + T_d \tag{4}$$

T_h 为路由器处理数据包头所耗的时间，T_r 为确定路由的时间，T_d 为数据传输至下一路由节点所耗的时间。

通常，核的测试时间远大于数据包的传输时间，因此本文不考虑传输时间的影响。

(2)功耗约束　考虑到异构 NoC 每层的功耗不同，因此有必要将功耗约束到每一层。对于每个时隙(time slot)，第 l 层的功耗必须满足:

$$P_l = \sum_{i=0}^{N_C^l - 1} P(li) \cdot z_{li} \le P_m(l) \tag{5}$$

$$z_{li} = \begin{cases} 1, & \text{第 } l\text{层上core } i\text{处于测试状态} \\ 0, & \text{其它} \end{cases} \tag{6}$$

其中 $P(li)$ 为第 l 层上 core i 的功耗，N_C^l 为第 l 层上的 core 数量，$P_m(l)$ 为第 l 层的最大允许功耗。总功耗即为各层功耗之和，不能超过系统规定的最大测试功耗值 P_{sm}。

单个核的功耗包含传输功耗和测试功耗，即 $P = \mathrm{np} \cdot P_p + P_c$，其中 np 是该核的数据包数量，$P_c$ 是该核的测试功耗，P_p 是一个数据包通过路径的传输功耗。

$$P_p = N_r \cdot P_r + (N_{ch} - N_{tv}) \cdot P_{xy} + N_{tv} \cdot P_{tv} \tag{7}$$

其中 P_r 代表数据包在一个路由器上的传输功耗，P_{xy} 代表数据包在一条水平通道上的传输功耗，P_{tv} 代表数据包在一条 TSV 上的传输功耗，N_r，N_{ch}，N_{tv} 分别代表数据包经过的路由器数量，通道数量以及 TSV 数量，其中通道数量包含了 TSV 数量。

$$P_r = C_L \cdot \mathrm{Vdd}^2 \cdot \frac{1}{T} \cdot \left[(\sigma_{ff} + 1) \cdot \mathrm{nb_{ff}} + \sigma_{gt} \cdot \mathrm{nb_{gt}} \right] \tag{8}$$

$$P_{xy} = C_L \cdot \mathrm{Vdd}^2 \cdot \frac{1}{T} \cdot \sigma_w \cdot (\mathrm{ch_l} \cdot \mathrm{wire_w} \cdot \mathrm{ch_w}) \tag{9}$$

其中 C_L，T 和 σ_w 分别表示负载电容(技术有关常数)、时钟周期以及开关因子。变量 $\mathrm{nb_{ff}}$，$\mathrm{nb_{gt}}$，σ_{ff} 和 σ_{gt} 分别表示活性元件的数量(触发器或者逻辑门)及其通过的路由器上相应的开关因子[10]。

P_r，P_{xy} 以及 P_{tv} 均为常数。采用文献[10]的功耗模型，P_r=10，P_{xy}=2。由于互连 TSV 很短，比水平互连通道小两个数量级，所以功耗远远小于水平通道的功耗，考虑到 TSV 的工艺与分布结构还没有统一，本文设定 P_{tv}=1。

4　基于云模型进化算法的测试规划方法

3D NoC 测试规划属于 NP 难问题[11]，而基于云模型的进化算法[12−14]是一种高精度寻优的算法，将该算法与 3D NoC 测试结合起来，可以迅速而且准确地找到最优的资源(I/O 端口，TSV)分配方案，从而提高 TSV 利用率，优化测试时间。

4.1 算法的相关设计

假定 NoC 有 L 层，将底层称为第 0 层，其上各层按顺序依次称为第 1 层，第 2 层……。核数量 $N_C = \sum_{l=0}^{L-1} N_C^l$，$N_C^l$ 为第 l 层的核数量，各层核的编号依次 $0, 1, \cdots, N_C^l - 1$。TSV 数量 $N_V = \sum_{l=0}^{L-2} N_V^l$，其中 N_V^l 为第 l 层的 TSV 数量，各层 TSV 编号依次为 0, 1,…, $N_V^l - 1$。设 I/O 端口对数为 N_{IO}，则 I/O 端口对的编号依次为 0, 1,…, $N_{IO} - 1$。

本文以 3 层 NoC 为例来进行说明。

4.1.1 染色体　对于本文所研究的问题，一条染色体就是完成一个待测核测试的一种资源(I/O 端口，TSV)分配方案。设定染色体 A_{li} =(G_0, G_1, G_2, G_3, G_4)，其中 G_0~G_4 为染色体中的基因。G_0 所含信息为 I/O 端口编号，G_1~G_2 依次为测试数据从输入端口传输至 core li 先后经过的各层 TSV 编号，G_3~G_4 依次为测试数据从 core li 传输至输出端口先后经过的各层 TSV 编号。各基因在 3D NoC 中的映射如图 3 所示。染色体所包含的信息为测试 core li 时测试数据所经过的关键通道。例如染色体 A_{26}= (0, 0, 0, 3, 3)所含信息为测试第 2 层第 6 号芯核时，测试数据先后经过的关键通道依次为输入 0→底层 TSV0→第 1 层 TSV0→core li→第 1 层 TSV3→底层 TSV3→输出 0。

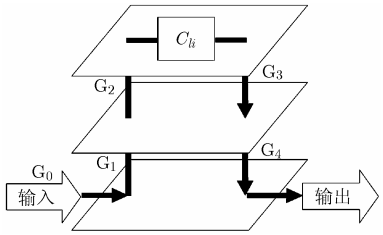


图 3 染色体结构映射图

需要注意的是，当测试底层芯核时，测试数据不需要经过任何 TSV，此时 G_1~G_4 为无效基因，用‘X’来表示，染色体标志为(G_0,X,X,X,X)；当测试第 1 层芯核时，测试数据不需要经过第 2 层以上的 TSV，此时 G_2 和 G_3 为无效基因，染色体标志为(G_0,G_1,X,X,G_4)。

4.1.2 个体 个体为完成对所有待测核进行测试的一种资源分配方案。一个个体有 N_c 条染色体，N_c 为待测核的数量。例如，A_{00}=(2,X,X,X,X)，A_{10}=(1,1,X,X,2)，$_{20}$= (0,0,0,3,3)构成一个个体，该个体有 3 条染色体，代表 NoC 有 3 个待测核，分别分布在第 0, 1, 2 层，每条染色体所含的信息代表着一个待测核的测试路径。

4.1.3 个体适应度值 文中以完成对所有 core 进行测试所需的时间作为个体的适应度值，测试时间越小，该个体越优秀。测试时间依据式(1)计算。在测试过程中，需要对分配在各 I/O 端口对上进行测试的芯核进行调度，尽量缩短测试时间。本文采用非抢占式的测试调度策略，测试调度的伪代码如表 1 所示。

表 1 测试调度的伪代码

输入：(a)含待测核资源(I/O 端口和 TSV)分配信息的个体
(b)各待测核的测试时间
输出: NoC 的测试时间 T
(1)**Begin**
(2) Sort the cores of each I/O pair in decreasing order of test time
(3) /*$T[i]$ indicates the total test time needed to test cores scheduled on I^{th} I/O pair */
(4) Initialize $T[i]$=0, for i=0,1,2 ··· number of I/O pair
(5) **While** there is any unscheduled core **do**
(6) **For** each free I/O pair with unscheduled cores **do**
(7) Find the unscheduled core with no path and power confliction, and schedule it
(8) **End**
(9) **If** there is any core has been finished tested **do**
(10) $T[i]$= $T[i]$+T_{st}, // T_{st} is the slot time
(11) **End**
(12) **End**
(13) T=max{ $T[i]$, i∈set of I/O pairs}
(14) **End**

4.1.4 云模型 本文选用 1 维正态云模型 $C(E_x, E_n, H_e)$实现进化过程。期望 E_x 代表父代的优良特性；熵 E_n 表示云模型中云滴距离期望值 E_x 的离散范围；超熵 H_e 是熵的不确定度，反映了云模型中数值的凝聚程度。正态云模型更新云滴过程：

(1)生成期望值为 E_n ，方差为 H_e 的正态随机数， $E_n^{'}=\text{RANDN}(E_n, H_e)$；

(2)以 E_x 为期望值，$(E_n^{'})^2$ 为方差生成正态随机数 x, x 称为云滴；

(3)计算不确定度 $y=e^{-(x-E_x)^2/(2(E_n^{'})^2)}$。

更新个体时，E_x 取个体的基因值，E_n 代表基因在进化过程中变异的范围，E_n=e/c_1，其中 e 为各基因取值范围。当 E_x=G_0 时，e 值为 I/O 端口对数；当 E_x=G_1 或 G_4 时，e 值为底层 TSV 数量；当 E_x=G_2 或 G_3 时，e 值为上层 TSV 数量。H_e 代表了进化过程中搜索寻优的范围，H_e=E_n/c_2。c_1, c_2 均为可控参数。在进化过程中，出现跨代精英即更优秀个体时，应调节 c_1, c_2 使 E_n 和 H_e 减小，缩小搜索范围，进行局部求精；当平凡代数(连续不出现跨代精英的代数)达到一定值时，应调节 c_1, c_2 使 E_n 和 H_e 增大，从而扩大搜索范围，进行局部求变操作。

4.1.5 交叉变异 包括杂交、自交以及变异。杂交过程采用轮盘赌的方式选取 2 个个体，越优秀的个体被选中的概率越大，个体的基因交换方式采用多点交换，同样以轮盘赌的方式选择需要交换的染色体，含有有效基因较多的染色体被选择的概率较大，基因交换如图 4 所示。自交过程是将一个个体中不同染色体的相同位置的有效基因进行交叉互换。变异则是利用随机函数更新个体中的某些有效基因。交叉变异后的个体需要检验是否合格：个体中所有 G_0 基因的值是否涵盖了所有 I/O 端口编号值，若不是，则个体不合格，需重新进行交叉变异。

4.2 算法设计

普通的云模型进化算法一般只针对一个群落，从该群落中选取优秀的种子进行个体更新，当连续 n 代没有发现精英个体才进行交叉变异操作，而且各种子个体间有血缘关系，进化代数达到一定时，各个体的基因有很大相似度，最终导致算法寻优速度慢、容易陷入局部最优等情况。为了克服这些缺点，算法中设计了普通群落、杂交群落以及精英群落 3 个群落，群落的系统结构如图 5 所示。普通群落为整个系统提供多样基因，用以维持系统的多样性，主要利用了生物学中物以类聚的小生境原理，该群落中的各种群之间没有血缘关系。根据优胜劣汰的生存原则，利用随机种群替换普通群落中的劣势种群。实验证明，杂交群落中往往比较容易产生优良品种，因此在算法中加入杂交种群以加快寻优速度。精英群落挑选优良品种进行繁殖，该群落使用云模型进化来更新个体以提高寻优精度。

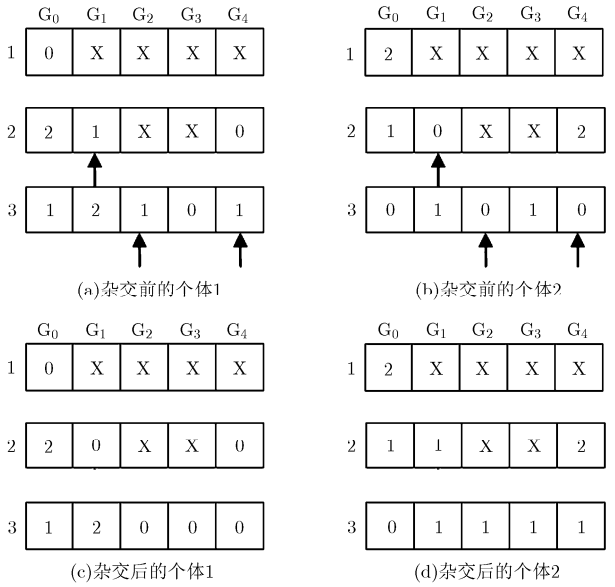


图 4　基因交换

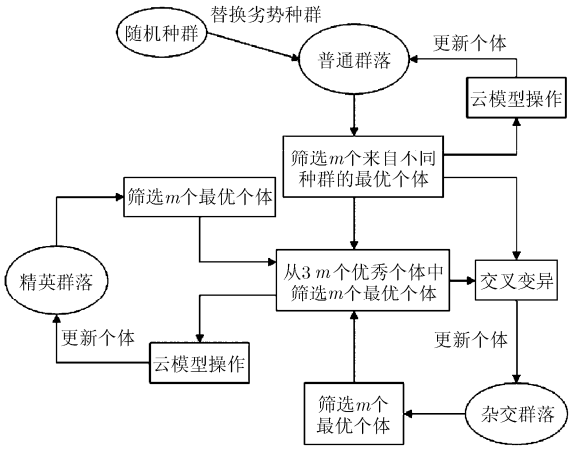


图 5　群落的系统结构框图

3 个群落之间的关系体现在种子筛选的环节。

种子筛选：从普通群落中筛选 m 个来自不同种群的最优个体，作为更新普通群落的种子(普通种子)。从所有群落中选取 m 个不同的最优个体，作为更新精英群落的种子(精英种子)。将普通种子与精英种子作为杂交群落的种子。由于普通群落中的各种群之间没有血缘关系，因此普通种子之间有较大的基因差异，可以保持系统的多样性，避免算法进入局部最优。精英种子主要利用优良种子进行繁殖，加快算法寻优速度。

整个算法流程如图 6 所示，主要步骤有：初始化群落、适应度评估、个体选择、个体更新等。

(1)初始化群落：利用随机函数产生 n 个个体，n 为群落规模，每个个体含有 N_c 个待测核的测试资源(I/O 端口，TSV)分配信息，初始化 3 个群落。

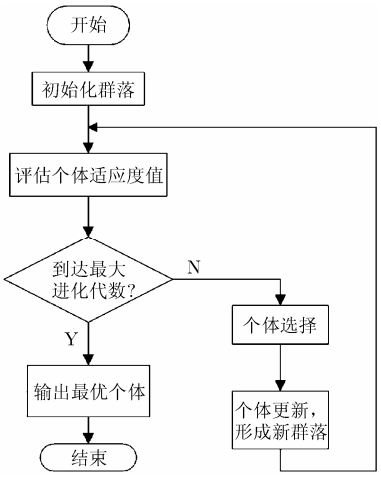


图 6　算法流程图

(2)适应度评估：计算每个个体的测试时间。

(3)个体选择：如前文的种子筛选。

(4)个体更新：该步骤是整个算法的核心部分，包括 3 个群落的个体更新，其中普通群落和精英群落利用云模型更新个体，过程相似。而杂交群落通过交叉变异来更新个体。

云模型操作：通过正向云发生器来更新每个个体中的基因，无效基因‘X’不需要更新。步骤如下：

(1)设置云模型中的 3 个特征参数 Ex, En, He，设置过程已在 4.1 节作了分析；

(2)将 3 个特征参数输入到正向云发生器中，产生一个云滴，即一个新的基因。

(3)重复步骤(1)至步骤(2)，直到产生群落所需的个体数为止。此过程中，越优秀的种子产生的种群规模越大。在普通群落中，淘汰掉最劣势的种群，用随机函数补充种群，充分利用自然界优胜劣汰的生存规则，以加快寻优速度。

交叉变异操作：设定杂交概率为 0.8，自交和变异的概率均为 0.1。在筛选种子时，为了减小近亲杂交的概率，应限定所选种子中最多只有一个精英种子。

5　实验结果与分析

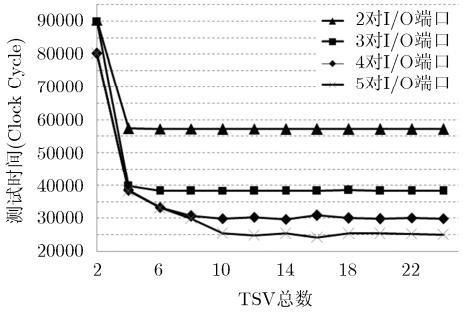
本文选择 ITC′ 02 SoC 标准电路作为实验对象，采用 Visual C++编写程序，其中正向云发生器用 MATLAB 7.1 的 MCC 方式与 VC 混编实现，在 Intel(R) core i3 3.1 GHz CPU, 3 G 内存 Windows XP 操作系统下运行。本文基于 4×3×3 mesh 结构的 3D NoC，选取 SoC 基准电路 d695×3(30 个内嵌核)和 p93791(32 个内嵌核)作为研究对象。

实验设定不同的 TSV 数量，利用本文算法对

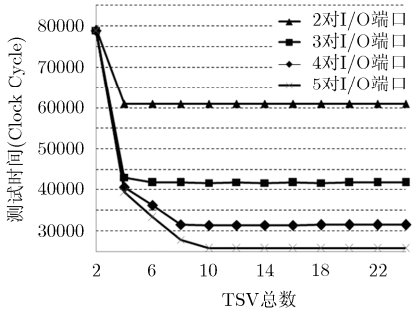
每种 TSV 数量的各种分配方案进行测试分析。实验过程中，分别设定 I/O 端口对数为 2, 3, 4 和 5；测试功耗限制为系统功耗的 50%。由于本实验测试的规模较大，为了保证能搜寻到最优测试时间，经过多次实验调整，设定每个群落的规模为 1000。

图 7 给出了各种 TSV 数量所对应的两种基准电路 IP 核的最优测试时间。由图 7 可以看出，I/O 端口对数一定时，测试时间随着 TSV 数量的增加而下降，但当 TSV 增加到一定数量时，测试时间基本上保持稳定的变化趋势。原因是 TSV 数量在一定范围内的增加使得测试数据可以在上下层之间畅通传输，但当 TSV 过多而 I/O 端口对数有限时，只会造成一些 TSV 处于饥饿状态，即造成 TSV 浪费。另外，由图 7 纵向对比可知，当 TSV 数量一定时，测试时间随着 I/O 端口对数的增加而减少，但当 TSV 数量较少时，上下层间测试数据的传输达到了饱和状态，此时即使增加 I/O 端口对数，也不能减少测试时间。图 7 数据表明，对于规模为 4×3×3 的 3D mesh 结构 NoC, I/O 端口对数与 TSV 数量有一定的匹配关系，当 I/O 端口对数为 2 时，在 TSV 数量为 4 左右处开始寻得最优测试时间；当 I/O 端口对数为 3 时，在 TSV 数量为 6 左右处开始寻得最优测试时间；当 I/O 端口对数为 4 时，在 TSV 数量为 8 左右处开始寻得最优测试时间；当 I/O 端口对数为 5 时，在 TSV 数量为 10 左右处开始寻得最优测试时间。此时 TSV 在各层的最优分配方案如表 2 第 3 列所示，括号中第 1 个数据为上层 TSV 数量，第 2 个为底层 TSV 数量。由表 2 中数据可以看出，在测试模式下，当 TSV 总数一定时，底层分配的 TSV 数量多于上层，可以提高 TSV 的利用率，优化测试时间。

为了验证改进算法(ICEA)的性能，在表 2 相同的条件下，即 I/O 端口对数、TSV 总数以及 TSV 分配方案相同，NoC 的结构不变，将其与没有采用精英群落、杂交群落以及小生境技术的算法(CEA)进行 NoC 测试规划研究比较，实验设置 CEA 的群落规模为 3000，最大进化代数设为 2000，将研究结果与 ICEA 的研究结果进行比较，结果如表 2 所示。由表 2 的测试时间和寻优代数可以看出，无论在寻优精度还是寻优速度上，算法 ICEA 都比算法 CEA 有明显的优势，特别是在 I/O 端口对数以及 TSV 总数比较大时，优势更为明显。原因是本文算法将云进化算法与小生境技术以及遗传算法中的杂交技术有效地结合起来，提高了算法的寻优精度和寻优速度。



(a) d695测试时间与TSV数量关系



(b) p97391测试时间与TSV数量关系

图 7 测试时间与 TSV 数量关系

表 2 最优 TSV 数量的分配方案以及算法比较

基准电路	I/O 端口对数	TSV 总数	TSV 分配方案	测试时间 (clock cycle) ICEA	测试时间 (clock cycle) CEA	优化率 (%)	平均寻优代数 ICEA	平均寻优代数 CEA	优化率 (%)	程序运行时间(s) ICEA	程序运行时间(s) CEA
d695	2	4	(1,3)	57257	58521	2.2	138	195	29.2	289	585
	3	6	(2,4)	38224	39753	3.8	225	322	30.1	472	988
	4	8	(3,5)	30609	32138	4.8	417	643	35.2	917	1990
	5	10	(4,6)	25454	28046	9.2	562	961	41.5	1180	3170
p93791	2	4	(1,3)	793577	807301	1.7	153	210	27.3	329	674
	3	6	(2,4)	417684	433282	3.6	266	384	30.8	577	1221
	4	8	(3,5)	341780	363209	5.9	481	756	36.4	1024	2517
	5	10	(4,6)	265740	298920	11.1	604	1018	40.7	1286	3298

6 总结

本文采用改进后的进化算法对 TSV 数量受约束的 3D NoC 进行测试规划研究，并探讨 TSV 数量优化问题。选取 ITC′ 02 测试标准电路进行仿真实验，实验结果表明，当 I/O 端口对数一定时，适度选取 TSV 数量可以降低测试成本；当 TSV 数量一定时，底层分配的 TSV 数量多于上层，可以提高 TSV 的利用率。实验结果证明，加入小生境技术以及杂交技术的云模型进化算法收敛速度快，利用其对 3D NoC 进行测试规划研究，可以快速而且精确地找到最优测试方案。

本文提出的测试规划方法针对 3D NoC 的测试资源分配进行研究，算法将测试数据经过的路径定义为染色体，不依赖于 NoC 结构，因此适用于不同拓扑结构的 3D NoC。不同拓扑结构的 3D NoC 需要选取合适的路由算法，但不同的路由算法主要影响数据的传输时间，不会影响本方法的应用。

参 考 文 献

[1] Hwang Yong-joong, Lee Jae-hun, and Han Tae-hee. 3D Network-on-Chip system communication using mini-mum number of TSVs[C]. ICTC 2011, Seoul, Korea, 2011: 517–522.

[2] 欧阳一鸣, 刘蓓, 梁华国. 一种三维 SoCs 绑定前的测试时间优化方法[J]. 电子测量与仪器学报, 2011, 25(2): 164–169.
Ouyang Yi-ming, Liu Bei, and Liang Hua-guo. Optimizing method for pre-bond test time on three-dimensional SoCs[J]. *Journal of Electronic and Instrument*, 2011, 25(2): 164–169.

[3] Noia B, Chakrabarty K, and Xie Y. Test-wrapper optimization for embedded cores in TSV-based three-Dimensional SoCs[C]. IEEE International Conference on Computer Design, Lake Tahoe, CA, USA, 2009: 70–77.

[4] Roy S K, Giri C, Ghosh S, *et al*.. Optimizing test wrapper for embedded cores using TSV based 3D SoCs[C]. IEEE Computer Society Annual Symposium on VLSI, Chennai, India, 2011: 31–36.

[5] Cota Érika and Liu Chun-sheng. Constraint-driven test scheduling for NoC-based systems[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2006, 25(11): 2465–2478.

[6] Mali G, Das S, Rahaman H, *et al*.. Non-preemptive test scheduling for Network-on-Chip(NoC) based systems by reusing NoC as TAM[C]. IEEE Asia Pacific Conference Circuits and Systems, Kuala Lumpur, Malaysia, 2010: 268–271.

[7] Liu C, Zhang L, Han Y H, *et al*.. Vertical interconnects squeezing in symmetric 3D mesh Network-on-Chip[C]. Asia and South Pacific Design Automation Conference(ASP-DAC), Yokohama, Japan, 2011: 357–362.

[8] Sbiai T and Namba K. NoC Dynamically reconfigurable as TAM[C]. 2012 IEEE 21st Asian Test Symposium(ATS), Niigata, Japan, 2012: 326–331.

[9] Amory A, *et al*.. Determining the test sources/sinks for NoC TAMs[C]. 2013 IEEE Computer Society Annual Symposium, Natal, Brazil, 2013: 8–13.

[10] Cota E, Zeferino C, Carro L, *et al*.. Reusing an on-chip network for the test of core-based systems[J]. *ACM Transactions on Design Automation of Electronic Systems*, 2004, 9(4): 471–499.

[11] Chakrabarty K. Test scheduling for core-based systems using mixed-integer linear programming[J]. *IEEE Transactions on ComPuter-Aided Design of Integrated Cireuits and Systems*, 2000, 19(10): 1163–1174.

[12] 张光卫, 何锐, 刘禹, 等. 基于云模型的进化算法[J]. 计算机学报, 2008, 31(7): 1082–1091.
Zhang Guang-wei, He Rui, Liu Yu, *et al*.. An evolutionary algorithm based on cloud model[J]. *Chinese Journal of Computers*, 2008, 31(7): 1082–1091.

[13] 许川佩, 姚芬, 胡聪. 基于云进化算法的 NoC 资源节点优化测试研究[J]. 电子测量与仪器学报, 2012, 26(3): 192–196.
Xu Chuan-pei, Yao Fen, and Hu Cong. Optimal test of NoC resource nodes based on cloud evolution algori-thm[J]. *Journal of Electronic Measurement and Instrument*, 2012, 26(3): 192–196.

[14] 李国柱. 基于云模型的实数编码量子进化算法[J].计算机应用, 2013, 33(9): 2550–2552, 2569.
Li Guo-zhu. Real-coded quantum evolutionary algorithm based on cloud model[J]. *Journal of Computer Applications*, 2013, 33(9): 2550–2552, 2569.

许川佩：　女，1968 年生，教授，硕士生导师，主要研究方向为自动测试总线与系统、集成电路测试技术.

陈家栋：　男，1986 年生，研究生，研究方向为集成电路测试.