DrPlace: 基于深度学习的可布线性驱动布局算法

郝睿[1)], 蔡懿慈[1)*], 周强[1)], 王锐[2)]

[1)] (清华大学计算机科学与技术系 北京 100084)
[2)] (北京航空航天大学计算机学院 北京 100191)
(caiyc@tsinghua.edu.cn)

摘 要: 在集成电路物理设计的布局阶段, 针对基于深度学习的布局算法结果可布线性较差的问题, 在开源的 DREAMPlace 算法的基础上提出并实现了一种基于深度学习的可布线性驱动布局算法 DrPlace. 算法模型在总体上设计并实现了布局器的整体框架, 集成了基于深度学习的可布线性驱动总体布局、可布线性驱动的合法化和详细布局. 总体布局过程中, 在目标函数中加入了引脚密度函数, 并实现了基于 GPU 的引脚密度的关键内核. 在 ISPD 2011 和 DAC 2012 布局实例上的实验结果表明, 该算法与 DREAMPlace 相比在可布线性上获得了提升, 且在运行时间、线长和可布线性方面均优于传统的可布线性驱动布局算法.

关键词: 深度学习; 布局; 可布线性驱动
中图法分类号: TP302 **DOI:** 10.3724/SP.J.1089.2021.18566

DrPlace: A Deep Learning Based Routability-Driven VLSI Placement Algorithm

Hao Rui[1)], Cai Yici[1)*], Zhou Qiang[1)], and Wang Rui[2)]

[1)] (*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2)] (*School of Computer Science and Engineering, Beihang University, Beijing* 100191)

Abstract: In order to improve the routability of deep learning based placement algorithms in the VLSI physical design stage, on the basis of open source placement algorithm DREAMPlace, a deep learning based routability-driven VLSI placement algorithm named DrPlace is proposed. The framework of the algorithm model is composed of deep learning based routability-driven global placement, routability-driven legalization, and routability-driven detailed placement. The algorithm adds the pin density function into the global placement model and proposes an efficient GPU implementation of pin density key kernel. According to the experiment results on ISPD 2011 and DAC 2012 benchmark suites, the algorithm improves the routability of the DREAMPlace and is more effective than traditional routability-driven placement algorithms.

Key words: deep learning; placement; routability-driven

随着集成电路的快速发展, 电子设计自动化(electronic design automation, EDA)技术所面临的问题愈加复杂, 需要处理的电路规模和计算的数据量不断增大. EDA 技术的发展速度能否跟上设计和制造工艺的飞速进步成为了关键问题.

在集成电路物理设计阶段, 布局是一个重要

收稿日期: 2020-08-11; 修回日期: 2020-09-28. **基金项目**: 国家自然科学基金(61774091); 国家重点研发计划(2019YFB2205001). **郝睿**(1998—), 男, 博士研究生, CCF 学生会员, 主要研究方向为大规模集成电路物理设计算法; **蔡懿慈**(1960—), 女, 博士, 教授, 博士生导师, CCF 高级会员, 论文通讯作者, 主要研究方向为电子设计自动化; **周强**(1961—), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究方向为电子设计自动化; **王锐**(1978—), 男, 博士, 讲师, 硕士生导师, CCF 会员, 主要研究方向为计算机系统与体系结构、专用处理器设计、嵌入式系统.

且耗时的步骤. 首先, 布局过程中涉及大量的迭代和优化, 所需时间会显著地影响集成电路设计周期. 其次, 在集成电路的物理设计中, 各个步骤间存在着密切联系, 布局结果的可布线性会影响布线过程的运行时间、拥挤度和布通率等参数.

近年来, 除了以线长和时延为驱动外, 以可布线性为驱动的布局算法也受到关注. 尽管在过去的几十年中布局算法有了显著的进步, 快速且高效的布局仍然是一个具有挑战性的问题.

同时, 深度学习领域也在快速发展. 通用图形处理器(graphics processing unit, GPU)等芯片的应用, 使训练神经网络时能够使用的计算能力以指数级增长. PyTorch[1]等深度学习框架的出现, 也使训练神经网络模型变得更加简单、快速和高效. 这一系列突破, 为 EDA 技术的发展开辟了新的方向.

为加快布局速度, 文献[2]提出了一种基于深度学习、支持 GPU 加速的布局框架, 与最先进的布局算法相比, 虽然其大幅减少了总体布局时间且布局结果的总线长(wire length, WL)基本相同, 但结果可布线性较差; 而在迭代中加入快速总体布线来提高结果可布线性, 又会使总体布局时间大幅回升. 另外, 传统的可布线性驱动布局算法[3-6]受限于当时的思想与计算能力, 只能使用多线程加速, 要解决十万量级标准单元布局问题所需要的时间甚至要以小时计算. 综上所述, 如何快速地生成可布线性较好的布局结果, 是一个亟待解决的重要问题.

为解决上述问题, 本文提出了一种基于深度学习的可布线性驱动布局算法, 它能够在较短的时间内生成可布线性较好的布局结果.

1　相关工作

1.1　布局问题

布局问题属于组合优化问题, 是NP困难问题. 在布局问题中, 输入的集成电路门级网表描述文件等价于超图 $H=(V,E)$. 其中, 点集 V 代表标准单元、宏模块和焊块的集合; 超边集 E 代表线网的集合. w_i 和 h_i 分别代表单元 $v_i\in V$ 的宽度和高度, $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是所有单元左下角的横坐标和纵坐标组成的向量. 布局问题可以抽象成为一个约束最优化问题, 即在满足布局区域约束、电路实体不重叠约束和其他约束条件下最小化总线长, 以确定每个单元的具体位置.

目前, 解析布局[2,7-8]是解决布局问题的主流思想. 传统的解析布局一般分为总体布局、合法化和详细布局3个步骤. 总体布局的主要任务是通过优化目标解析函数使总线长达到最小, 此时基本确定了单元的大致位置. 合法化消除了单元之间的重叠, 并使单元按照行对齐. 为了将所有单元布局到合法位置, 不可避免地会使线长略微增加. 详细布局会逐步优化单元位置, 修复局部出现的问题, 进一步提高布局质量.

1.2　基于深度学习的布局算法

DREAMPlace[2]是一个使用深度学习视角解决超大规模标准单元布局问题的布局算法, 它以深度学习框架 PyTorch 为基础, 将解析布局与神经网络训练进行类比, 实现了常用线长函数和单元密度函数的关键 CUDA 内核. 在与先进的 RePlAce[7]算法的对比中, DREAMPlace 算法由于深度学习方法的应用, 使得其可以在总体布局上实现 30 倍加速的同时, 总线长不会明显增加.

1.3　可布线性驱动的布局算法

由于布局和布线过程之间的紧密联系, 且布局质量对后续的布线起到决定性作用, 可布线性驱动的布局算法也引起了学界关注. ISPD 2011[9]和DAC 2012[10]都是以可布线性为驱动的布局算法竞赛, 其中也产生了很多优秀的传统布局算法[3-6].

NTUplace4h[3]是一种可布线性驱动的布局算法, 它是一种根据线网分布、引脚密度和引脚布线方向实现的引脚密度控制技术. 此外, 该算法还实现了可布线性驱动的合法化和详细布局, 进一步优化了布局结果的可布线性.

2　本文算法模型

2.1　整体框架

本文算法作为一种基于深度学习的解析布局算法, 仍然按照总体布局、合法化和详细布局 3 个步骤解决布局问题, 其整体框架如图 1 所示.

首先, 在总体布局的开始阶段进行随机初始布局, 使单元分布在布局区域的中央并随机分开.

然后, 进入总体布局中核心的深度学习框架. 深度学习模型的结构如图 2 所示, 其主要作用是通过训练神经网络获取给定网表的总体布局结果, 因此其模型与传统深度学习模型有所不同. 2 种模型之间主要的区别在于 2 个方面. 首先, 由于模型不需要泛化能力, 因此其重点关注输入层偏置参数向量 $\boldsymbol{b}^0$ 的训练结果(即当前所有单元的横纵坐标)而忽略输入数据 $\boldsymbol{X}\equiv\boldsymbol{0}$. 其次, 神经网络中所有

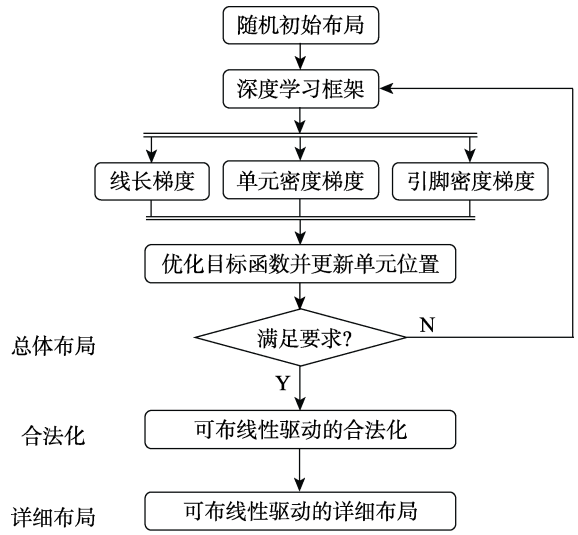


图 1 本文算法整体框架

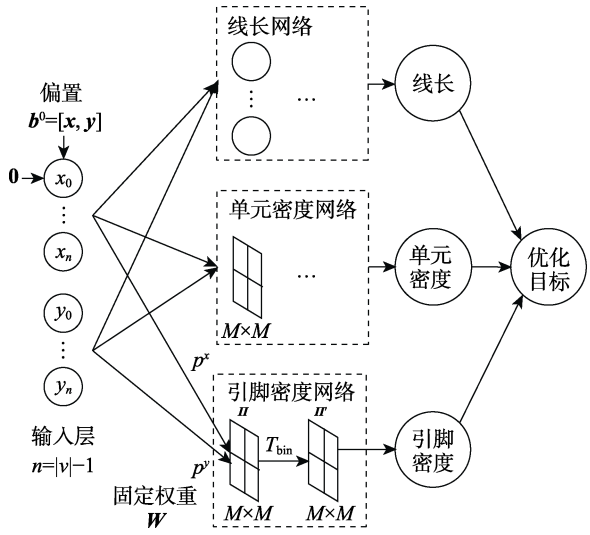


图 2 核心深度学习模型

层的结构和连接关系由具体的网表实例决定，所有权重 $\boldsymbol{W}$ 均为常量. 优化目标是线长函数、单元密度函数和引脚密度函数的加权和. 在每次正向传播中，按照模型计算目标函数值. 在每次反向传播中，传递线长函数、单元密度函数和引脚密度函数的梯度以更新输入层偏置参数 $\boldsymbol{b}^0$. 当目标函数满足要求或达到最大迭代次数时，退出神经网络的训练过程，生成总体布局的结果.

最后，对总体布局结果进行可布线性驱动的合法化和详细布局，得到最终的布局结果.

2.2 总体布局

作为可布线性驱动的布局算法，除了线长函数外，本文在目标函数中增加了单元密度函数，同时还增加了与可布线性相关的引脚密度函数. 故本文算法总体布局的优化目标函数为 $\min \mathrm{WL}(\boldsymbol{x},\boldsymbol{y})+\lambda D(\boldsymbol{x},\boldsymbol{y})+\mu P(\boldsymbol{x},\boldsymbol{y})$. 其中，线长函数 $\mathrm{WL}(\boldsymbol{x},\boldsymbol{y})$ 是主要优化目标，通常使用半周长线长(half perimeter wire length, HPWL)估计总线长；$D(\boldsymbol{x},\boldsymbol{y})$ 是用于减小单元重叠、减小合法化步骤难度的单元密度函数；λ 是单元密度惩罚因子；$P(\boldsymbol{x},\boldsymbol{y})$ 是用于减小网格内引脚密度、提高布局结果可布线性的引脚密度函数；μ 是引脚密度惩罚因子.

2.2.1 线长函数

本文算法以加权平均线长[11](weighted average wire length, WAWL)作为线长函数，它关于横纵坐标的计算相互独立.

对于给定线网 $e\in E$，横坐标的 WAWL 为

$$\mathrm{WAWL}_e^x(\boldsymbol{x})=\frac{\sum\limits_{v_i\in e}x_i\cdot \mathrm{e}^{x_i/\gamma}}{\sum\limits_{v_i\in e}\mathrm{e}^{x_i/\gamma}}-\frac{\sum\limits_{v_i\in e}x_i\cdot \mathrm{e}^{-x_i/\gamma}}{\sum\limits_{v_i\in e}\mathrm{e}^{-x_i/\gamma}} \tag{1}$$

其中，γ 为加权平均线长参数. 当 $\gamma\to 0$ 时，有 $\mathrm{WAWL}_e^x(\boldsymbol{x})\to\mathrm{HPWL}_e^x(\boldsymbol{x})$. 同理，纵坐标也有类似的计算式. 线长函数 $\mathrm{WL}(\boldsymbol{x},\boldsymbol{y})$ 是所有线网横纵坐标加权平均线长的和.

线长函数的正向传播及其梯度的反向传播可以分解成多个步骤分别计算，每个步骤中的计算过程可以达到引脚级并行[2].

2.2.2 单元密度函数

本文算法的单元密度函数使用的是 ePlace[8]中基于静电学的单元密度函数. 计算单元密度函数时通常将布局区域划分为 $M\times M$ 的网格(bin)，第 m 行第 n 列的网格记为 $b_{m,n}$. 单元密度可以通过离散三角函数变换求解静电学泊松方程近似解的方法求解. 单元密度函数

$$D(\boldsymbol{x},\boldsymbol{y})=\sum_{v_i\in V,v_i\in b_{m,n}}w_ih_i\phi(m,n) \tag{2}$$

其中，w_ih_i 为单元 v_i 的面积，可以类比于物理学中的电荷量；

$$\phi(m,n)=\sum_{u=0}^{M-1}\sum_{v=0}^{M-1}\frac{d_{u,v}}{\omega_u{}^2+\omega_v{}^2}\cos(\omega_u m)\cos(\omega_v n)$$

可以类比于电势；$\omega_k=2\pi k/M$，$k=0,1,\cdots,M-1$ 为离散三角函数变换中的常量；

$$d_{m,n}=\sum_{u=0}^{M-1}\sum_{v=0}^{M-1}\rho(u,v)\cos(\omega_m u)\cos(\omega_n v)$$

为单元密度函数计算过程中的中间量，是经过二维离散余弦变换后的电荷密度；$\rho(m,n)=\sum\limits_{v_i\in b_{m,n}}w_ih_i$ 可

以类比于电荷密度.

单元密度函数的正向传播及其梯度的反向传播可以通过快速傅里叶变换达到单元级并行[2].

2.2.3　引脚密度函数

本文算法在总体布局的目标函数中增加了可布线性函数，这是其与 DREAMPlace 等算法的不同之处. 本文选择引脚密度函数作为可布线性函数.

由于引脚在单元上的相对位置是固定的，因此将引脚视为在单元上均匀分布. 与单元密度模型类似，布局区域被划分为 $M\times M$ 的网格. 根据图 2 中的引脚密度网络正向传播过程，可以通过单元级并行计算单元 v_i 和网格 $b_{m,n}$ 在横纵坐标上交叠的长度，即横纵坐标交叠函数. 横坐标交叠函数

$$p^x\left(n,x_i\right)=\begin{cases}w_i, & 0\leqslant 2d_{\mathrm{c}}<w_{\mathrm{b}}-w_i\\ \dfrac{w_{\mathrm{b}}+w_i}{2}-d_{\mathrm{c}}, & w_{\mathrm{b}}-w_i\leqslant 2d_{\mathrm{c}}<w_{\mathrm{b}}+w_i\\ 0, & \text{其他}\end{cases}\tag{3}$$

其中，w_{b} 为网格的宽度；d_{c} 为单元 v_i 中心与网格 $b_{m,n}$ 中心在横坐标上的距离. 同理，纵坐标也有类似的交叠函数. 横坐标的交叠函数如图 3 所示.

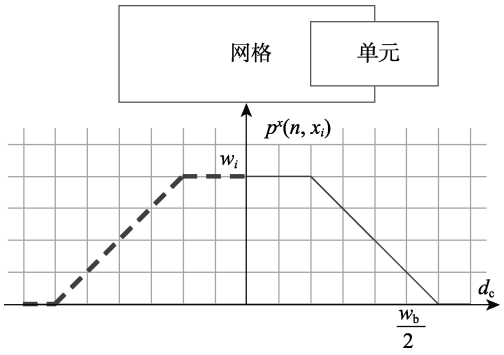


图 3　横坐标交叠函数

实现交叠函数后，神经网络继续正向传播，通过单元级并行计算每个单元对网格的引脚密度贡献，获得网格引脚密度矩阵，即

$$\boldsymbol{\Pi}_{m,n}=\sum_{v_i\in V}\frac{t_i}{w_ih_i}p^x\left(n,x_i\right)p^y\left(m,y_i\right)\tag{4}$$

其中，t_i 为单元 v_i 的总引脚个数.

然后，通过网格引脚密度阈值 T_{bin} 将 $\boldsymbol{\Pi}$ 阈值化为 $\boldsymbol{\Pi}'$. 引脚密度函数 $P(\boldsymbol{x},\boldsymbol{y})$ 是 $\boldsymbol{\Pi}'$ 中所有元素的平方和，即

$$P(\boldsymbol{x},\boldsymbol{y})=\sum_{m=0}^{M-1}\sum_{n=0}^{M-1}\left(\max\left(\boldsymbol{\Pi}_{m,n}-T_{\mathrm{bin}},0\right)\right)^2\tag{5}$$

在反向传播中，目标函数的梯度依据链式法则从引脚密度函数逐层向输入层参数传递，即

$$\frac{\partial P(\boldsymbol{x},\boldsymbol{y})}{\partial x_i}=\sum_{m=0}^{M-1}\sum_{n=0}^{M-1}2\max\left(\boldsymbol{\Pi}_{m,n}-T_{\mathrm{bin}},0\right)\frac{\partial\boldsymbol{\Pi}_{m,n}}{\partial x_i}\tag{6}$$

网格引脚密度矩阵 $\boldsymbol{\Pi}$ 继续向输入层传播梯度，即

$$\frac{\partial\boldsymbol{\Pi}_{m,n}}{\partial x_i}=\sum_{v_i\in V}\frac{t_i}{w_ih_i}\frac{\partial p^x\left(n,x_i\right)}{\partial x_i}p^y\left(m,y_i\right)\tag{7}$$

交叠函数在 2 个折点处使用单侧导数，可以继续向输入层参数反向传播引脚密度函数梯度. 横坐标交叠函数梯度的计算式为

$$\frac{\partial p^x(n,x_i)}{\partial x_i}=\begin{cases}0, & 0\leqslant 2d_{\mathrm{c}}<w_{\mathrm{b}}-w_i\\ \pm1, & w_{\mathrm{b}}-w_i\leqslant 2d_{\mathrm{c}}<w_{\mathrm{b}}+w_i\\ 0, & \text{其他}\end{cases}\tag{8}$$

纵坐标也有类似的反向传播过程.

2.3　合法化及详细布局

在合法化过程中，单元移动至拥塞区域时会产生与该单元互连线个数成正比的拥挤度代价. 在决定合法化可行位置时会权衡最小化线长增量与拥挤度代价，生成可布线性更好的结果.

在详细布局的单元匹配过程会在给定窗口内进行独立单元与空槽的二分匹配，并选择总线长和总溢出值最小的结果进行交换. 单元交换过程会以总线长和总溢出值为优化目标交换相邻单元. 在详细布局中对于总溢出值的关注和优化提升了布局结果的可布线性.

整体框架中集成了 NTUplace4h，以实现上述的可布线性驱动合法化和详细布局方法[3].

3　实验结果

3.1　本文算法实验结果

本文算法的相关实验均在 3.60 GHz Intel(R) Core(TM) i9-9900K CPU，GeForce RTX 2080 Ti GPU，内存为 32 GB 的 Linux 服务器上完成.

本文算法在布局时所使用的数据实例是 ISPD 2011 可布线性驱动布局算法竞赛中公布的 8 组基准套件，以及 DAC 2012 可布线性驱动布局算法竞赛中公布的 6 组基准套件. ISPD 2011 数据实例用于评估本文基础算法，详细介绍如表 1 所示.

DREAMPlace 2.2.0 在迭代中加入快速总体布线，优化了布局结果的可布线性. 本文算法在此基础上，同样在目标函数中增加了引脚密度函数，以进一步提高布局结果的可布线性，得到了优化算法. 使用 DAC 2012 中的部分与 ISPD 2011 不重复的基准套件作为评估本文优化算法的数据实例，共有 6 组，详细介绍如表 2 所示.

表 1 ISPD 2011 基准套件数据统计

实例	节点个数	线网个数	引脚个数	单元面积利用率/%
superblue18	483452	468918	1864306	67
superblue4	600220	567607	1884008	70
superblue5	772457	786999	2500306	77
superblue1	847441	822744	2861188	69
superblue2	1014029	990899	3228345	76
superblue15	1123963	1080409	3816680	73
superblue10	1129144	1085737	3665711	75
superblue12	1293433	1293436	4774069	56

表 2 DAC 2012 部分基准套件数据统计

实例	节点个数	线网个数	引脚个数	单元面积利用率/%
superblue19	522775	511685	1714351	78
superblue14	634555	619815	2049691	72
superblue16	698741	697458	2280931	69
superblue9	846678	833808	2898853	73
superblue3	919911	898001	3110509	73
superblue11	954686	935731	3071940	79

在这 14 组实例中, 标准单元数量从十万级至百万级, 单元面积利用率基本适中.

对于布局结果的可布线性进行评估需要用到总体布线器. 本文使用 DAC 2012 可布线性驱动布局比赛中所使用的总体布线器 NCTU-GR 2.0[12]作为所有布局结果的可布线性评估工具, 使用 DAC 2012 标准配置文件作为总体布线器的配置文件. 对于本文优化算法的评估还使用了 DAC 2012 公布的评估脚本.

本文基础算法使用标准参数在 ISPD 2011 各个布局实例上的运行时间, 以及使用 NCTU-GR 2.0 对基础算法布局结果进行可布线性评估得到的总溢出值(total overflow, TOF)、最大溢出值(maximum overflow, MOF)和总线长(WL)如表 3 所示.

表 3 本文基础算法布局结果可布线性评估

实例	时间/s	TOF	MOF	WL
superblue18	144.085	211978	22	8668287
superblue4	422.674	870378	68	11866325
superblue5	441.333	591260	46	16517237
superblue1	469.560	411542	26	14317725
superblue2	542.420	1931052	46	27782090
superblue15	360.077	196084	16	14785160
superblue10	355.377	561366	26	24864513
superblue12	375.975	2696916	104	15989922

本文优化算法使用标准参数在 DAC 2012 各个布局实例上进行布局, 并使用 NCTU-GR 2.0 和 DAC 2012 评估脚本对优化算法布局结果进行可布线性评估. 评估结果中的布线拥挤度(routing congestion, RC)可通过峰值加权拥挤度(peak weighted congestion, PWC)和边平均拥挤度(average congestion of edge, ACE)计算. ACE 是在总体布线图中拥挤度前 x%的边的平均拥挤度. PWC 是 ACE 的平均值, 即

$$\mathrm{PWC}=\frac{\mathrm{ACE}(0.5)+\mathrm{ACE}(1)+\mathrm{ACE}(2)+\mathrm{ACE}(5)}{4}.$$

然后可以进一步计算

$$\mathrm{RC}=\max(100,\mathrm{PWC}) \tag{9}$$

折算线长(scaled half perimeter wire length, sHPWL)的方法计算为

$$\mathrm{sHPWL}=\mathrm{HPWL}\times(1+0.03\times(\mathrm{RC}-100)) \tag{10}$$

本文优化算法布局的可布线性评估结果如表 4 所示.

表 4 本文优化算法布局结果可布线性评估

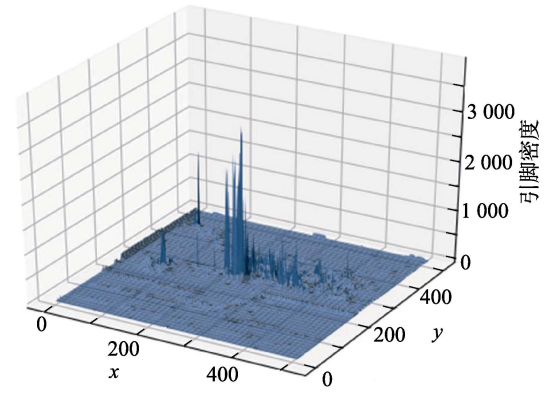
实例	时间/s	TOF	RC	sHPWL
superblue19	131.514	8162	101.57	14.41M
superblue14	166.263	4862	100.48	21.44M
superblue16	157.180	13994	101.63	25.39M
superblue9	157.628	8294	100.52	21.38M
superblue3	594.867	6580	100.43	30.27M
superblue11	434.477	13946	100.62	32.94M

在实际布局过程中, 单元个数、单元面积利用率与宏模块的分布对布局结果的可布线性都会产生复杂的影响. superblue1 是单元个数和单元面积利用率较为适中的实例, 因此对其进行重点分析. 本文基础算法在对 superblue1 实例进行布局时, 迭代中网格引脚密度快照如图 4 所示.

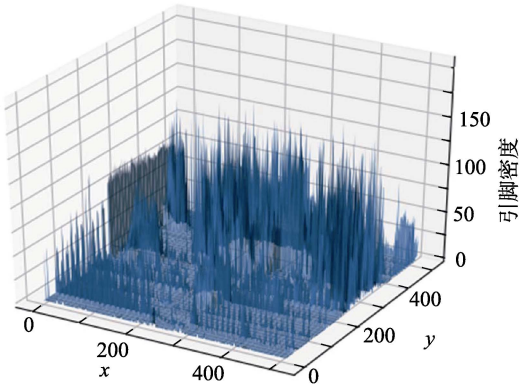
在图 4 中, 初始大部分单元聚集在布局区域中央, 使布局区域中间部分引脚密度极高. 经过逐步迭代后, 最大引脚密度逐渐下降, 同时引脚密度分布在布局区域变得更加平均.

3.2 与 DREAMPlace 的对比实验

本文是以 DREAMPlace 算法为基础, 在其上增加了可布线性优化的布局算法. 在可布线性对比实验中, 本文算法使用与 DREAMPlace 相同的标准参数. 以本文基础算法布局结果为基准, 将 DREAMPlace 算法布局结果及其可布线性评估情况标准化, 如表 5 所示.



a. 100次迭代后的引脚密度



b. 最终引脚密度

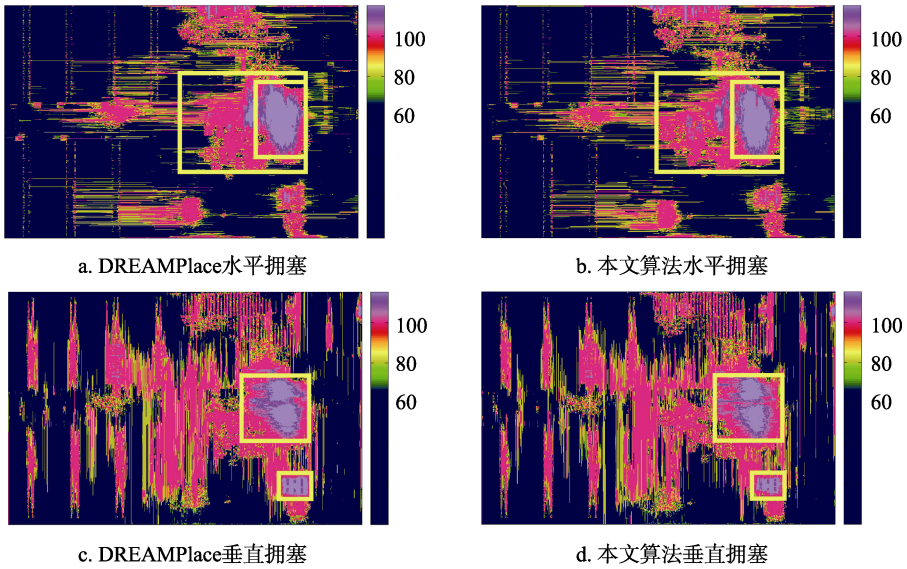
图 4　迭代中的网格引脚密度(superblue1 实例)

表 5　DREAMPlace 算法布局结果可布线性评估

实例	时间/s	TOF	MOF	WL
superblue18	170.559	281 782	24	8 729 764
superblue4	613.873	1 247 526	104	12 233 826
superblue5	519.220	516 516	40	16 606 072
superblue1	508.108	422 164	34	14 439 762
superblue2	642.799	1 935 978	46	27 668 455
superblue15	456.864	195 866	16	14 846 753
superblue10	389.385	501 296	24	24 952 260
superblue12	470.927	2 736 112	104	16 041 272
标准化	1.212	1.071	1.090	1.007

经过对比，加入引脚密度惩罚项后，本文基础算法运行时间和总线长有微小的减少，TOF 平均减少了 7%. 因此本文算法目标函数中的引脚密度项对可布线性具有提升作用.

图 5 通过更直观的方法对可布线性进行了对比. 与 DREAMPlace 相比，本文基础在水平和垂直互连线拥塞图的黄色方框中都出现了紫色拥塞区域面积减小、红色较拥塞区域变得更加分散的现象，数据结果也表明了本文算法的可布线性有所提升.



a. DREAMPlace水平拥塞　　b. 本文算法水平拥塞

c. DREAMPlace垂直拥塞　　d. 本文算法垂直拥塞

图 5　互连线拥塞对比(superblue1 实例)

同时，对于单元个数较少的布局实例(如 superblue18 和 superblue4)，本文基础算法在可布线性上有更大的优势. 这是因为在这种实例中，2 个布局器都能在到达最大迭代次数之前完成布局. 因此，在完整的迭代过程中，目标函数的引脚密度项对可布线性的优化作用较为明显，TOF 的减少最多能够达到 30%.

在迭代中加入快速总体布线，以优化可布线性，将 DREAMPlace 2.2.0 算法与本文优化算法在同一标准参数下进行对比. DREAMPlace 2.2.0 算法布局结果及其可布线性评估情况标准化，如表 6 所示.

表 6 DREAMPlace 2.2.0 算法布局结果可布线性评估

实例	时间/s	TOF	RC	sHPWL
superblue19	129.216	14488	102.73	14.67 M
superblue14	164.850	7066	100.51	21.24 M
superblue16	154.424	14464	101.94	25.53 M
superblue9	152.902	7574	100.65	21.23 M
superblue3	582.472	7428	100.65	30.18 M
superblue11	429.089	11780	100.79	32.80 M
标准化	0.982	1.191	1.003	1.000

表 6 中, 由于文献[2]的实验环境与本文不同, 且未提供 NCTU-GR 2.0 可布线性评估的 TOF, 因此运行时间与 TOF 是通过 DREAMPlace 算法的开源代码在本文实验环境下运行获得, RC 和 sHPWL 引用自文献[2].

经过对比, 加入引脚密度惩罚项后, 本文优化算法比 DREAMPlace 2.2.0 算法的运行时间有微小的增加, sHPWL 基本不变, 但 TOF 平均减少了 19%. 在所有数据集上, 本文优化算法的 RC 均有所降低, 其在数值上平均减少 0.337.

综上所述, 加入引脚密度模型后, 本文算法的可布线性获得了提升. 可见, 引脚密度项对于将单元按照可布线性更好的方式分散有着指导作用.

3.3 与传统布局算法的对比实验

以本文优化算法为基准, 对传统可布线性驱动布局算法 NTUplace4 和 Ripple[4-5]的布局结果进行可布线性评估, 结果如表 7 所示.

表 7 传统可布线性驱动布局算法对比

实例	NTUplace4①		Ripple[4-5]	
	RC	sHPWL	RC	sHPWL
superblue19	100.67	15.34 M	102.10	16.90 M
superblue14	101.25	23.36 M	102.79	24.64 M
superblue16	102.99	28.51 M	101.52	27.08 M
superblue9	105.97	27.94 M	105.91	30.33 M
superblue3	106.38	39.00 M	112.40	45.86 M
superblue11	101.02	39.11 M	101.04	36.73 M
标准化	1.022	1.177	1.034	1.245

表 7 中, 由于 DAC 2012 可布线性驱动布局算法竞赛中所使用的实验环境与本文实验不同, 因此运行时间不在表中列出, 但在 DAC 2012 官方公布的运行时间中, NTUplace4 算法的运行时间均大于 8000 s, Ripple 算法的运行时间均大于 2000 s.

在 DAC 2012 可布线性驱动布局算法竞赛中, 传统的可布线性驱动布局算法 NTUplace4 和 Ripple 分别获得了第 1 名和第 2 名的成绩. 但是相比于传统布局算法, 本文优化算法的 RC 数值上平均减小了 2.795, sHPWL 减少了约 20%. 同时, 由于 CPU 和通用 GPU 计算能力的提高以及深度学习框架的使用, 原来传统算法需要用数小时才能完成的布局过程, 本文优化算法在 10 min 内即可完成. 综上, 本文算法所需要的布局时间远远少于传统算法, 同时在线长和可布线性方面优于传统的可布线性驱动布局算法.

4 结 语

本文实现了一个基于深度学习的可布线性驱动布局算法. 该算法设计并实现了基于深度学习的可布线性驱动布局器的整体框架, 以可布线性驱动进行基于深度学习的总体布局、合法化和详细布局. 在总体布局过程中使用了引脚密度函数, 并通过深度学习框架使引脚密度函数的计算达到了单元级并行. 本文算法在 ISPD 2011 基准套件和 DAC 2012 基准套件上进行了实验. 实验结果表明, 无论是否在迭代中加入快速总体布线,与 DREAMPlace 算法相比, 本文算法在可布线性上均获得了提升, 在目标函数中添加引脚密度项能够提高布局结果的可布线性. 此外, 在运行时间、线长和可布线性等方面, 本文算法比传统的可布线性驱动布局算法具有极大的优势.

未来的工作将从多个方面对本文算法进行改进, 使其在保持布局结果可布线性基本不变的同时, 进一步减少布局时间.

参考文献(References):

[1] Paszke A, Gross S, Massa F, *et al*. PyTorch: an imperative style, high-performance deep learning library[C] //Proceedings of the 33rd on Neural Information Processing Systems. Cambridge: MIT Press, 2019: 8026-8037

[2] Lin Y B, Dhar S, Li W X, *et al*. DREAMPlace: deep learning toolkit-enabled GPU acceleration for modern VLSI placement[C] //Proceedings of the 56th Annual Design Automation Conference. New York: ACM Press, 2019: Article No.117

[3] Hsu M K, Chen Y F, Huang C C, *et al*. NTUplace4h: a novel routability-driven placement algorithm for hierarchical mixed-size circuit designs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014, 33(12): 1914-1927

① http://archive.sigda.org/dac2012/contest/dac2012_contest_benchmarks.html#head-results

[4] He X, Huang T, Xiao L F, *et al*. Ripple: an effective routability-driven placer by iterative cell movement[C] //Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Los Alamitos: IEEE Computer Society Press, 2011: 74-79

[5] He X, Huang T, Chow W K, *et al*. Ripple 2.0: high quality routability-driven placement via global router integration[C] // Proceedings of the 50th Annual Design Automation Conference. New York: ACM Press, 2013: Article No.152

[6] Kim M C, Hu J, Lee D J, *et al*. A SimPLR method for routability-driven placement[C] //Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Los Alamitos: IEEE Computer Society Press, 2011: 67-73

[7] Cheng C K, Kahng A B, Kang I, *et al*. RePlAce: advancing solution quality and routability validation in global placement[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 38(9): 1717-1730

[8] Lu J W, Chen P W, Chang C C, *et al*. ePlace: electrostatics based placement using Nesterov's method[C] //Proceedings of the 51st Annual Design Automation Conference. New York: ACM Press, 2014: 1-6

[9] Viswanathan N, Alpert C J, Sze C, *et al*. The ISPD-2011 routability-driven placement contest and benchmark suite[C] //Proceedings of the International Symposium on Physical Design. New York: ACM Press, 2011: 141-146

[10] Viswanathan N, Alpert C, Sze C, *et al*. The DAC 2012 routability-driven placement contest and benchmark suite[C] //Proceedings of the 49th Annual Design Automation Conference. New York: ACM Press, 2012: 774-782

[11] Hsu M K, Chang Y W, Balabanov V. TSV-aware analytical placement for 3D IC designs[C] //Proceedings of the 48th Design Automation Conference. New York: ACM Press, 2011: 664-669

[12] Liu W H, Kao W C, Li Y L, *et al*. NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013, 32(5): 709-722