# 摘 要

为了适应日益增长的宽带信号和非线性系统的工程应用，用于分析瞬态电磁散射问题的时域积分方程方法研究日趋活跃。本文以时域积分方程时间步进算法及其快速算法为研究课题，重点研究了时间步进算法的数值实现技术、后时稳定性问题以及两层平面波算法加速计算等，主要研究内容分为四部分。

……

**关键词：**时域电磁散射，时域积分方程，时间步进算法，后时不稳定性，时域平面波算法

# ABSTRACT

With the widespread engineering applications ranging from broadband signals and non-linear systems, time-domain integral equations (TDIE) methods for analyzing transient electromagnetic scattering problems are becoming widely used nowadays. TDIE-based marching-on-in-time (MOT) scheme and its fast algorithm are researched in this dissertation, including the numerical techniques of MOT scheme, late-time stability of MOT scheme, and two-level PWTD-enhanced MOT scheme. The contents are divided into four parts shown as follows.

……

**Keywords:** time-domain electromagnetic scattering, time-domain integral equation (TDIE), marching-on in-time (MOT) scheme, late-time instability, plane wave time-domain (PWTD) algorithm

# 目 录

# 第一章 绪论

## 1.1 研究工作的背景与意义

……

计算电磁学方法[1–6] 从时、频域角度划分可以分为频域方法与时域方法两大类。频域方法的研究开展较早，目前应用广泛的包括：矩量法（MOM）[7, 8]及其快速算法多层快速多极子（MLFMA）[9]方法、有限元（FEM）[1, 3]方法、自适应积分（AIM） [5]方法等，这些方法是目前计算电磁学商用软件 ① （例如：FEKO、Ansys 等）的核心算法。由文献[6, 7, 9]可知……

……

## 1.2 时域积分方程方法的国内外研究历史与现状

时域积分方程方法的研究始于上世纪60 年代，C.L.Bennet 等学者针对导体目标的瞬态电磁散射问题提出了求解时域积分方程的时间步进（marching-on in-time, MOT）算法[8]。……

……

## 1.3 本文的主要贡献与创新

本论文以时域积分方程时间步进算法的数值实现技术、后时稳定性问题以及两层平面波加速算法为重点研究内容，主要创新点与贡献如下：

……

## 1.4 本论文的结构安排

本文的章节结构安排如下：

……

① 脚注序号“①,……，⑩”的字体是“正文”，不是“上标”，序号与脚注内容文字之间空1个半角字符，脚注的段落格式为：单倍行距，段前空0磅，段后空0磅，悬挂缩进1.5字符；中文用宋体，字号为小五号，英文和数字用Times New Roman字体，字号为9磅；中英文混排时，所有标点符号（例如逗号“，”、括号“（）”等）一律使用中文输入状态下的标点符号，但小数点采用英文状态下的样式“.”。

# 第二章 时域积分方程基础

时域积分方程（TDIE）方法作为分析瞬态电磁波动现象最主要的数值算法之一，常用于求解均匀散射体和表面散射体的瞬态电磁散射问题。

## 2.1 时域积分方程的类型

## 2.2 空间基函数与时间基函数

利用数值算法求解时域积分方程，首先需要选取适当的空间基函数与时间基函数对待求感应电流进行离散[①]。

### 2.2.1 空间基函数

……

RWG 基函数是定义在三角形单元上的最具代表性的基函数。它的具体定义如下：

$$f_n(r)=\begin{cases}\frac{l_n}{2A_n^+}\rho_n^+=\frac{l_n}{2A_n^+}(r-r_+) & r\in T_n^+\\ \frac{l_n}{2A_n^-}\rho_n^-=\frac{l_n}{2A_n^-}(r_--r) & r\in T_n^-\\ 0 & \text{其它}\end{cases} \tag{2-1}$$

其中，$l_n$为三角形单元$T_n^+$和$T_n^-$公共边的长度，$A_n^+$和$A_n^-$分别为三角形单元$T_n^+$和$T_n^-$的面积（如图2-1所示）。

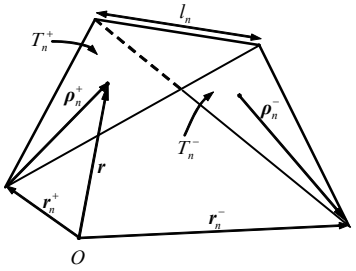



图 2-1 RWG 基函数几何参数示意图

……

① 脚注序号“①,……，⑩”的字体是“正文”，不是“上标”，序号与脚注内容文字之间空1个半角字符，脚注的段落格式为：单倍行距，段前空0磅，段后空0磅，悬挂缩进1.5字符；中文用宋体，字号为小五号，英文和数字用Times New Roman字体，字号为9磅；中英文混排时，所有标点符号（例如逗号“，”、括号“（）”等）一律使用中文输入状态下的标点符号，但小数点采用英文状态下的样式“.”。

### 2.2.2 时间基函数

……

#### 2.2.2.1 时域方法特有的展开函数

……

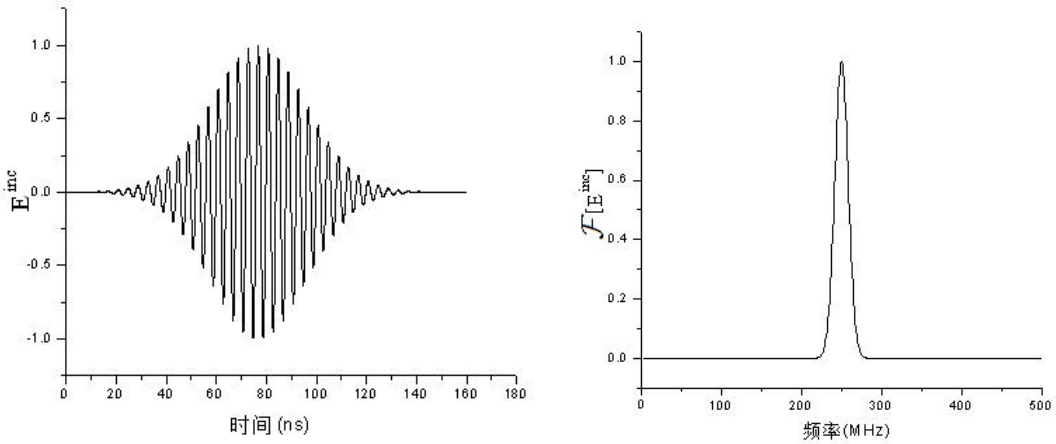
#### 2.2.2.2 频域方法特有的展开函数

……

## 2.3 入射波

……

如图2-2(a)和图2-2(b)所示分别给出了参数$E_0 = \hat{x}$，$a_n = -\hat{z}$，$f_0 = 250MHz$，$f_w = 50MHz$，$t_w = 4.2\sigma$时，调制高斯脉冲的时域与频域归一化波形图。




(a) (b)

图 2-2 调制高斯脉冲时域与频率波形(a)调制高斯脉冲时域波形;(b)调制高斯脉冲频域波形

## 2.4 本章小结

本章首先从时域麦克斯韦方程组出发推导得到了时域电场、磁场以及混合场积分方程。……

# 第三章 SDN网络下的并行路由优化算法设计

## 3.1 引言

路由优化能够有效提高网络性能，提高网络资源利用率，保障用户Qos需求，软件定义网络是一个新兴的，将控制和数据平面分离的网络范型，在SDN网络中，控制平面被分离驻留在一个集中的网络控制器上，它提供了用于网络应用程序和控制数据平面的编程接口（使用标准协议，例如OpenFlow[1]）。SDN的结构使得网络管理者可以根据网络当前情况来进行有效的在线路由优化。微软[1]和谷歌[1]的实验结果证明，在SDN网络结构的数据中心网络中，路由优化能够在网络吞吐量和链路利用率上达到接近最优化性能的表现。但是另一方面，SDN网络下中心控制的路由优化面临大规模计算问题，第一，随着网络应用的快速增加，在SDN网络中短时间内可能会有大量业务到达控制器，所以控制平面必须短时间内为大量业务计算路由。第二，为了适应大业务量的加入，网络规模也快速增大(e.g., a data center network may have hundreds of thousands of switches [9]).因此，对大量业务的快速和高效的路由优化成为了一个重要却困难的问题。为了加快网络优化过程，减小网络延迟，本文利用GPU的大规模的并行能力来加速网络优化算法的计算过程。对大规模业务的路由优化问题可以建模成一个多商品流问题，将网络路由业务作为输入，寻找最优的路由路径来最优化效用函数，效用函数通常设置为对网络拥塞程度的评价水平，比如，最常用的效用函数是最小化最大链路利用率（MLU），简单地被定义为利用率最高的那条链路的链路利用率[],另外一些把所有链路的链路利用率的和作为效用函数（[],[]）。这些效用函数的逻辑是：（1）低链路利用率意味着低的网络延迟。（2）.维持低的链路利用率意味着预留更多的空间给其他将来到达的业务。但是大量基于实际拓扑的实验表明链路利用率效用函数，特别是链路利用率，在网络利用率没有达到拥塞程度的时候，不是对网络优化的较好评价函数[]。在这个实验中，当链路利用率低于0.9的时候会造成不可忽略的网络性能中断。所以作为替代，文章在链路容量约束的情况下来优化路由总代价。我们假设已经知道短时间内到达的一批业务，控制器需要计算出满足链路容量约束的路径，并且最优化链路路由总代价。为了使得加入网络的业务尽量多，我们设定被阻塞的业务代价为一个较大值。本文中考虑的优化问题是一个NP-hard问题，因为他等价于一般的带整数约束的商品流问题[]。路由优化问题是一个组合优化问题，一般来说，在大规模网络中求解路由优化问题是计算困难的，因此，为了在短时间内求解路由优化问题，很多启发式算法被提出来[]。

然而，大部分算法都是单线程的串行算法，串行算法的复杂度随着网络规模大小呈指数上升，SDN网络下的在线路由优化要求很短的路由优化时间，为了加速算法，一个很自然的选择就是设计并行优化算法，采用大量的线程同时计算路由路径。本章主要设计两种路由优化算法，第一种是基于备选路径选择的路由优化算法，采用遗传算法来优化目标函数，并且设计了遗传算法的并行版本，获得几十倍的加速比，第二种是基于拉格朗日松弛的优化算法，算法把链路容量约束松弛到目标函数，并把路由优化问题分解成一堆路由路径计算问题，从而采用GPU进行并行计算。

## 3.2 网络模型和问题建模

### 3.2.1 网络模型

本文将SDN网络建模成无向图 $G(V,E)$，$V$表示所有的点集合， $E$是所有边的集合， $n=|V|$ 和 $m=|L|$分别表示点数和边数。对每一条边$(i,j)\in E$， $w_{ij}$表示此边 $(i,j)$上的权重（传输一单位的流量需要的代价），不失一般性，我们假设每条链路上的$w_{ij}$是整数，对每一条边 $(i,j)$， $c_{ij}$表示此边上的容量，假设$D$表示需要被路由的业务需求集合，业务$d\in D$是一个元组 $(s_d,t_d,bw_d)$，其中， $s_d$表示业务的源节点，$t_d$表示业务的目的节点，$bw_d$表示业务d需要的流量带宽。业务量工程问题将网络业务需求和网络拓扑作为输入，计算出每条业务的路由路径以使得效用函数代价最小化，在SDN网络中，业务的路径在中心控制器上计算出来，为了满足用户业务的QoS要求，本文假设链路利用率不能超过一个固定阈值 $\theta$，因此，一些业务会因为链路上容量不足而被阻塞，本文用 $\hat{D}$来表示这些被阻塞的业务集合。

### 3.2.2 问题建模

本小节，我们把路由优化问题建模成一个混合整数规划模型（MILP），通常，路由优化中应用最广泛的效用函数是[][]，与链路利用率成正比的，但是，最近的研究表明链路利用率不能很好的代表网络表现情况，而且一味的最求低的链路利用率，容易造成出现路由代价过大，路由跳数过长的问题，所以，本文结合了两种情况，采用新的效用函数，如下：

$$f(\mathbf{d})=\begin{cases}\sum\limits_{d\in D} c(p_d) & \text{if available bandwidth is enough}\\ \sum\limits_{d\in \hat{D}} bw_d & \text{otherwise}\end{cases} \tag{3-1}$$

其中$p_d$是计算出来的对应于业务$d$的路径，$c(p_d)$ ($c(p_d)=\sum_{(i,j)\in p_d} w_{ij}$)表示的是此路径$p_d$的代价。因为不知道带宽是否足够容纳所有业务，（1）并不是衡量所有情况，为了衡量所有情况，我们对$G(V,E)$构建辅助图$G_a(V_a,E_a)$，初始时，让$G_a(V_a,E_a)=G(V,E)$，然后，对每个点$v\in V$ 和 $u\in V$，在$G_a(V_a,E_a)$中添加一条链路 $(v,u)$，并且设置链路 $(u,v)$的容量和代价分别为 $\infty$和 $nM$ ，其中 $M$是 $G(V,E)$中最大的链路代价，这样，$G_a(V_a,E_a)$就有足够的容量来容纳业务需求，如果某条业务被路由到$G_a(V_a,E_a)$中，那么就表示这条业务被阻塞了，加入了辅助图$G_a(V_a,E_a)$后，路由优化的效用函数可以表示为：

$$z^* = minimize\ f(\mathbf{d}) = \sum_{d\in D} c(p_d) = \sum_{d\in D}\sum_{e\in p_d} w_e \tag{3-2}$$

在路由优化问题中，每个业务只能够路由到一条路径上，以下整数约束能够保证每个业务只走一条路径

$$\sum_{(i,j)\in E_a} x_{ij}^d - \sum_{(j,i)\in E_a} x_{ji}^d = \begin{cases} 1 & \text{if } i = s_d \\ -1 & \text{if } i = t_d \\ 0 & ohterwise \end{cases} \quad \forall i\in V_a, \forall d\in D \tag{3-3}$$

其中 $x_{ij}^d$是一个0，1整数变量， $x_{ij}^d=1$表示业务$d$路由经过链路$(i,j)$，为了避免链路拥塞，路由路径需要满足以下的链路容量约束:

$$\sum_{d\in D}\sum_{(i,j)\in E_a} x_{ij}^d \cdot bw_d \le \theta \cdot c_{ij} \ \ \forall (i,j)\in E_a \tag{3-4}$$

在这个模型中，变量的数量随着业务量大小和网络规模大小呈指数增长，所以这个MILP模型在大规模情况下很难求解。

## 3.3 基于遗传算法的路由优化算法

遗传算法是一种模拟自然进化过程搜索问题最优解的启发式算法，遗传算法模仿达尔文进化论和自然选择过程来评价挑选最优解集合，从而找寻较优化的解，遗传算法从一个代表问题的可行解的种群出发，一个种群中不同个体代表了不同的解，每个个体实际上是一个染色体，染色体携带表达当前解的信息编码，初代种群产生后，对每个染色体个体进行评价，按照适者生存，优胜劣汰的原则，使得较优的个体更有可能把自己的遗传信息传递给下一代，从而得到更优化的后代，算法过程中，对一部分基因进行变异，好的变异能够提高解的质量，增加算法的搜索空间，避免算法收敛于局部最优解。遗传算法的步骤流程图如下所示：

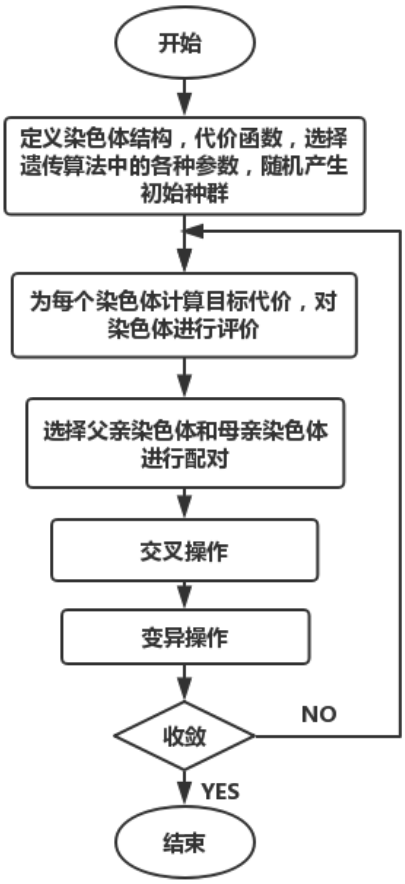

开始
定义染色体结构，代价函数，选择遗传算法中的各种参数，随机产生初始种群
为每个染色体计算目标代价，对染色体进行评价
选择父亲染色体和母亲染色体进行配对
交叉操作
变异操作
收敛
NO
YES
结束

图 3-1 GA算法流程图

### 3.3.1 遗传算法设计

#### 3.3.1.1 定义染色体结构

[5][], 由前面的讨论可以知道，路由优化问题求解过程是寻找最优的业务路径集合，使得效用函数最小化，在论文[]中作者（效用函数是常见的最小化最大链路利用率）提出了一种路由优化算法，在文章中，为每一个业务$d \in D$,产生$K$条不同的备选路径作为备选路径集合$P_i = p_d^1, p_d^2, p_d^3 ... p_d^K$,通过遗传算法过程来确定每个业务选择哪一条备选路径，从而找到最优解，本文采用相同的思想来求解路由优化问题，假设业务数量为$|D|$,初始染色体集合大小为$POP$对于第$j$（$jin[1:POP]$）号染色体一个染色体$C_j$是一个$|D|$维数组，$C_j^i = k$fi$k \in [1:K]$表示在第$j$号染色体中，业务$i$选择了第$k$条备选路径，$C_j^i = -1$表示业务$i$不加入网络中（在辅助图上路由），不占用链路资源，$|p_d^i|$表示业务$d$的第$i$条备选路的路径代价，$rp_d^i$表示路径$p_d^i$上的最小可用容量,$rp_d^i = min(r_e | e \in p_d^i)$,其中$e \in p_d^i$ 表示路径$p_d^i$上的边，$r_e$表示此边$e$上的剩余容量。

#### 3.3.1.2 初始可行解生成

可行解表示满足容量约束的解，为了使得遗传算法有效，初始解的质量很重要，产生的初始解要尽量好，要有更多的业务要能加入网络中，而且保证业务的路径代价较小，文章中采用一种简单的贪心算法产生出初始可行解，算法过程如下所示： 对某一个染色体，算法随机为每个业务生成所选择的备选路径编号，但是这样选择出来的路径集合有可能会超过网络链路的容量限制，从而使得解变得不可行，要得到可行解，必须从染色体中剔除一部分业务，使得他们阻塞，为了得到比较优秀的初始可行解，本文提出一种启发式过程来确定能加入的业务，以及必须剔除的业务，一方面，要使得目标函数变小，那些流量需求较大的业务应该优先被加入到网络中，但是如果大流量的业务的路由代价很大，经过了一条很长的路径，就会大量的浪费网络中的链路容量资源，所以算法过程对当前染色体$j$中的业务和其路径按照$\frac{bw_d}{\sqrt{|p_d^{k_j^d}|}}$的值进行排序，其中$bw_d$代表当前业务$d$所需要的流量大小，$|p_d^{k_j^d}|$代表当前染色体$j$所选择的$p_d^j$中的第$k_j^d$条路径的代价大小。这样算法优先加入$\frac{bw_d}{\sqrt{|p_d^{k_j^d}|}}$值较大的业务,观察目标函数，目标函数是优化路由代价最小，而$\frac{bw_d}{\sqrt{|p_d^{k_j^d}|}}$较大意味着较大的流量经过较小代价的链路进行路由，这种路由是很理想的，尽量节省网络的链路使用资源的同时，又减小了总体目标函数，所以这个比例值是对业务路由个体优劣程度的较好评价，于是文章采用这个比例值

**Algorithm 1** 初始可行解产生

**Require:** $G(v,E)$:网络拓扑; $P$:备选路径集合; $C$:未初始化的染色体集合;

**Ensure:** $C$:可行染色体集合;

1: **for** each $c_j \in C$ **do**
2: **for** each $c_j^d \in c_j$ **do**
3: $c_j^d \leftarrow -1$
4: **end for**
5: **end for**
6: **for** each $c_j \in C$ **do**
7: **for** each $c_j^d \in c_j$ **do**
8: $c_j^d \leftarrow k_j^d$，其中$k_j^d$为1到K之间的随机值，随机选择一条备选路
9: **end for**
10: 对染色体中的的每个业务需求按照值$\frac{bw_d}{\sqrt{|p_d^{k_j^d}|}}$进行降序排序。
11: **for** each $c_j^d \in c_j$ **do**
12: **if** $rp_d^{k_j^d} >= bw_d$ **then**
13: 加入路径$p_d^{k_j^d}$到网络，更新网络链路容量。
14: **else**
15: $c_j^d \leftarrow -1$
16: **end if**
17: **end for**
18: **end for**

来确定路径加入网络的优先级（后面第节也会应用类似的思想来调整路由），每次按照比例值排序的结果将遗传染色体所选择的路径$p_d^{k_j^d}$尝试加入到网络中，如果$rp_d^{k_j^d} >= bw_d$，表示路径经过的链路有足够的容量来容纳这一个业务，所以加入业务到网络中，并且更新网络的链路容量大小，反之，如果$rp_d^{k_j^d} < bw_d$,这个业务选择这一条路径会超过网络链路的容量限制，于是这条业务被阻塞，染色体中的相应基因位置被设置为-1。大量重复以上可行解的产生过程，则可以得到一个较好的初始染色体集合$C$。

#### 3.3.1.3 评价与交叉

评价过程对本轮产生的染色体，计算其相应的目标效用函数值，并且对染色体按照效用函数目标进行降序排序，由于算法过程随机交叉，可能会产生不可行

的染色体解（链路容量超限），把这样的染色体评价为一个个很大的代价，从而在选优时被排除掉，目标值排在最前面的前$\alpha$个染色体为最优集合$A$，这个集合中的染色体为精英染色体，精英染色体将直接保留到下一轮迭代，此后的$\beta$个染色体为较优染色体集合$B$，较优染色体集合中的染色体不会直接保留到下一轮，但是他们有繁殖的权利，可以和精英染色体一样产生后代，遗传自己的选路信息，排在最后面的$\gamma$个染色体，组成劣等染色体集合$G$,由于其目标值一般较大，其选路策略不可取，算法直接扔掉这一部分劣等染色体，而且不让劣等染色体进行繁殖。交叉过程从精英染色体集合中$A$中随机选取一个染色体$c_i \in A$作为父亲,从精英染色体集合和较优染色体集合的并集$A \cup B$中随机选取一个染色体$c_j \in A \cup B$作为母亲，将$c_i$和$c_j$进行均匀交叉得到新的染色体$s$，均匀交叉过程示意如下所示，均匀交叉的过程是，对$s$的每一个基因点位以$\%50$的几率选择继承父亲或者母亲的相应点位的路径选择。重复以上过程$\beta+\gamma$次，从而产生$\beta+\gamma$个新的子染色体来替换当前染色体集合评价函数排在后$\beta+\gamma$个的染色体，因此得到新的染色体集合$C$。

#### 3.3.1.4 变异与迭代终止

变异过程采用随机变异，随机在已经交叉后的集合$C$中选取$M \in [0:POP]$条染色体，对某一选定的染色体$c_j \in C$,随机选取$m$个业务基因点位，进行变异，将当前已经选择的路径编号随机改变为备选路集合中的另外一个值，由于变异过程是为了提高算法的的搜索空间，避免算法陷入局部最优解，但是实际实验过程中发现如果$M$和$m$值设置较大，可能使得算法收敛较慢，因为大量的变异可能会导致较优秀的可行解变成不可行，因此会丢掉这些优秀的解，因此实验中$M$和$m$的值设置得较小，变异的作用有限，算法的收敛效果主要来自于交叉步骤。算法每次迭代都会记录当前可行染色体解的最优目标值，如果当前迭代找到的最优可行解目标值小于全局最优值，则更新全局最优值，并且记录对应的染色体为最优解，如果迭代$L$次，全局最优值不被更新，则判定算法收敛，算法停止。

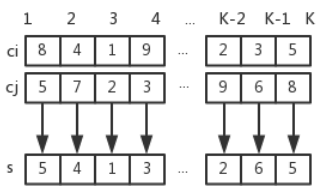



图 3-2 遗传算法均匀交叉过程

### 3.3.2 基于GPU的并行遗传算法设计

#### 3.3.2.1 并行评价算法设计

遗传算法中最消耗时间的部分是染色体评价部分，由于需要评价大量的染色体，评价每个染色体都需要大量计算开销，但是幸运的是遗传算法具有天然的并行性，每个不同的染色体评价可以并行执行，更进一步，每个染色体中的不同基因的计算也可以并行执行，这样并行粒度是很大的。下面将介绍评价过程的具体并行实现算法，算法主要包括可行性判断和效用函数计算两个步骤，算法计算流程如下图所示： 符号解释：
染色体（chromosome）基因编号：$c_i^d$表示第$i$个染色体的第$d$个基因位置。
备选路径集合（paths），$p_i$表示第$i$业务的所有备选路集合。
业务带宽（bandwidth）,$bw_i$表示第$i$个业务需要的带宽大小。
链路流量（flow），$f_e$表示第链路$e$上占用的流量大小。
链路单位代价（weight）,$w_e$表示链路$e$上的代价。
共享内存中间数组（shared），$sh_e$表示链路$e$上的总代价。
如图所示，由于每个染色体的计算过程是独立的，算法为每一个染色体开辟一个block，每个block内部每个线程负责染色体上的相应业务，首先通过寻址备选路径集合找到这个业务选择的路径，路径上的每一个链路都需要被占用流量，于是遍历这条路径，将业务的流量加到相应的链路上，所有线程同时计算，最后得到链路上占用的流量大小为数组$flow$,并行比较$flow$和$capacity$数组，如果某一线程发现容量超限，则设置链路代价为无穷大（INF）表示此染色体不可行，最后对得到的链路代价数组进行求和，求和采用GPU上经典的并行规约算法，最后得到染色体对应的效用函数值。其中$flow$和$shared$两个数组被同一个block内部的线程多次访问，利用程序访问的局部性，将两个数组分配到共享内存中，这样避免了对global memery的大量慢速访问，大大提高程序计算速度。 以上代码GPU上的评价过程伪代码，算法开始时先开辟两个大小为边数大小的共享内存数组，然后每个线程负责一个基因点，寻址基因点选择的路径，对路径上的每一条边的流量加上业务的流量大小，注意到这个时候可能同时存在多个线程访问同一个$flow$位置，所以需要同步保护，使得每个线程的加法操作都能正确执行，使用CUDA提供的atomicAdd函数来对$flow$数组进行加法操作，atomicAdd保证其调用的操作是原子操作，从而多个线程对同一$flow$位置的加法操作必修是串行执行的，一个add操作必修一次性执行完成（取址、译码、执行、访存、写回），在当前add操作执行完成之前，其他线程的add操作必修排队等待。$blockDim$表示一个block内的线程数量，代码中的for循环每次迭代标号$i$偏移$blockDim$的长度，这是因为业务量

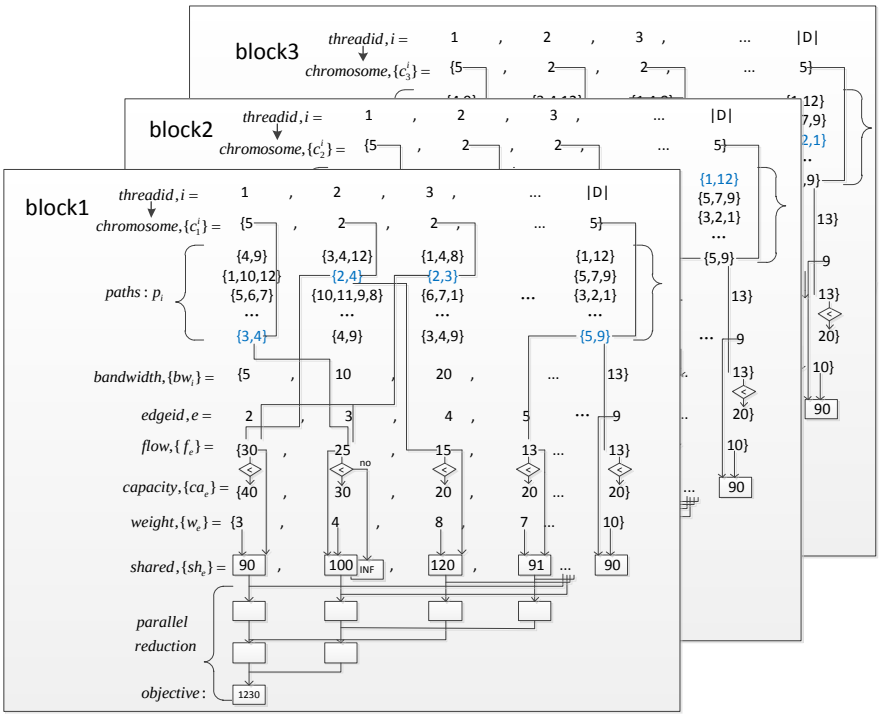



图 3-3 GPU上遗传算法评价函数计算过程图

数$|D|$可能大于$blockDim$,如果这样的话，每个线程可能会负责多个业务的统计计算，每个业务标号相差$blockDim$大小。当线程$flow$大小统计完成之后，必修进行同步（syncthreads()）,同步操作保证block内部的所有线程执行到同一步骤，也就是统计完成的线程必修等待其他统计线程都执行完成后才能继续执行，因为只有所有线程都完成对$flow$数组的加法操作，$flow$数组的统计才完整，才能够进一步进行比较操作。代码第19到25行计算每一条链路的路由代价，并把结果储存在$shared$数组中，如果链路上的流量小于其容量约束，那么链路代价就等于单位代价$weight$和链路流量$flow$的乘积，反之，如果流量超限，就设置$shared$为无穷大。同理，当线程计算完成后必修进行同步（syncthreads()）操作，以使得shared数组正确完整。为了充分利用GPU多线程，代码最后进行并行规约操作进行求和，for循环中每次将后一半的$shared$数组加到前一半，规约过程中必修进行同步（syncthreads()），以保证加法过程计算完整，最终求和值规约到一个下标0，将shared[0]中的值写入到$objective$数组中。另外，由于CUDA每个block支持的shared memory 大小有限，并且分配shared memory太多，会使得SIMT上的资源不足，一个block中的活跃warp数量不足，造成SIMT上不能有足够的活跃warp数量来进行切换，从而掩藏其他warp的访存延迟开销，这样会使得执行速度下降很多，所以在实际设计计算

```
1  void evaluate(float*capacity,
2                    float*bandwith,
3                    float*weight,
4                    int chromsome[][],
5                    int paths[][]
6                    int &objective[]){
7
8  __shared__ float flow[E]={};
9  __shared__ float shared[E]={};
10 for(i=threadid;i<|D|;i+=blockDim){
11         int p[]=paths[i][chromsome[blockid][i]];
12         for(j=0,j<length_of(p);j++)
13                 flow[p[j]]+=bandwidth[blockid];
14 }
15 __syncthreads();
16 for(i=threadid;i<E;i+=blockDim)
17 {
18         if(flow[i]<capacity[i])
19                 shared[i]=flow[i]*weight[i];
20         else
21                 shared[i]=INF;
22 }
23 __syncthread();
24 for(int s=E;s>1;s=(s+1)/2)
25 {       if(e<s/2)
26                 shared[e]+=shared[e+(s+1)/2];
27         __syncthreads();
28 }
29 if(threadid==0)
30         objective[blockid]=shared[0];
31 }
```

图 3-4 GPU上遗传算法评价函数计算伪代码

时$flow$和$shared$使用的是同一段共享内存。

#### 3.3.2.2 并行排序，变异与交叉

由于遗传算法中最消耗时间的部分是评价部分，本设计中对其他部分的并行步骤采用较简单的算法。在评价部分结束后，需要对所有的染色体按照效用函数的大小降序排序，本文采用GPU上的odd-even算法进行排序操作，odd-even算法的计算过程如下图所示： 如图所示，奇偶排序算法每次两两比较数组中的值的大小，然后将较小的那个值交换到前面，奇数次的时候比较下标为$2k2k+1$的值,其中$k \in [0, POP/2]$,偶数次的时候比较下标为$2k-12k$的值，其中$k \in [1, POP/2]$，最终经过$POP/2$轮的奇偶比较，就可以得到排序好的数组，从奇偶排序的执行过程可以看出，每一轮比较中的$POP/2$次比较是是相互独立无关的，其具有天然的可并行性，而且其并行实现较简单。下面介绍染色体排序算法的GPU代码实现：

## 3.4 基于Lag的优化算法设计

### 3.4.1 问题建模

### 3.4.2 基于GPU的并行lag算法设计

### 3.4.3 仿真实验分析

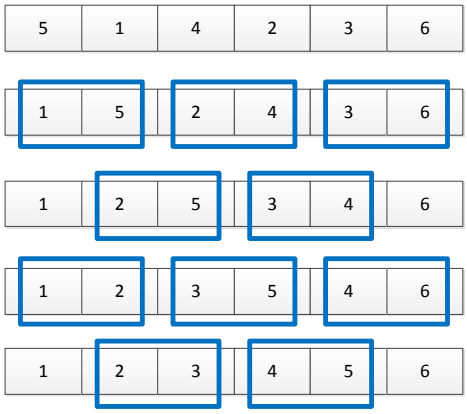



图 3-5 并行奇偶排序例子

```
1  void sort_kernel(float*objective,
2                   int**chromsome,
3                   int round)
4  {
5  int id=2*threadid;
6  if(round/2==0)
7    if(objective[id]<objective[id+1])
8    {
9           swap(objective[id],objective[id+1]);
10          swap(chromsome[id],chromsome[id+1]);
11   }
12 else
13   if(objective[id-1]>objective[id])
14   {
15          swap(objective[id-1],objective[id]);
16          swap(chromsome[id-1],chromsome[id]);
17   }
18 };
19 void sort(float*weight,
20                  int**chromsome,
21                  int round,
22                     int POP
23                     )
24 {
25 for(int i=0;i<POP/2;i++)
26     sort_kernel(objective,chromsome,i);
27 }
```

图 3-6 GPU并行奇偶排序伪代码

# 第四章 全文总结与展望

## 4.1 全文总结

本文以时域积分方程方法为研究背景，主要对求解时域积分方程的时间步进算法以及两层平面波快速算法进行了研究。

……

## 4.2 后续工作展望

时域积分方程方法的研究近几年发展迅速，在本文研究工作的基础上，仍有以下方向值得进一步研究：

……

# 致 谢

在攻读博士学位期间，首先衷心感谢我的导师 XXX 教授，……
……

# 参考文献

[1] 王浩刚, 聂在平. 三维矢量散射积分方程中奇异性分析[J]. 电子学报, 1999, 27(12):68–71

[2] X. F. Liu, B. Z. Wang, W. Shao. A marching-on-in-order scheme for exact attenuation constant extraction of lossy transmission lines[C]. China-Japan Joint Microwave Conference Proceedings, Chengdu, 2006, 527–529

[3] 竺可桢. 物理学[M]. 北京: 科学出版社, 1973, 56–60

[4] 陈念永. 毫米波细胞生物效应及抗肿瘤研究[D]. 成都: 电子科技大学, 2001, 50–60

[5] 顾春. 牢牢把握稳中求进的总基调[N]. 人民日报, 2012年3月31日

[6] 冯西桥. 核反应堆压力容器的LBB分析[R]. 北京: 清华大学核能技术设计研究院, 1997年6月25日

[7] 肖珍新. 一种新型排渣阀调节降温装置[P]. 中国, 实用新型专利, ZL201120085830.0, 2012年4月25日

[8] 中华人民共和国国家技术监督局. GB3100-3102. 中华人民共和国国家标准–量与单位[S]. 北京: 中国标准出版社, 1994年11月1日

[9] M. Clerc. Discrete particle swarm optimization: a fuzzy combinatorial box[EB/OL]. `http://clere.maurice.free.fr/pso/Fuzzy_Discrere_PSO/Fuzzy_DPSO.htm`, July 16, 2010

# 攻硕期间取得的研究成果

[1] J.-Y. Li, Y.-W. Zhao, Z.-P. Nie. New Memory Method of Impedance Elements for Marching-on-in-Time Solution of Time-Domain Integral Equation[J]. Electromagnetics, 2010, 30(5):448–462

[2] 张三, 李四. 时间步进算法中阻抗矩阵的高效存储新方法[J]. 电波科学学报, 2010, 25(4):624–631

[3] 张三, 李四. 时域磁场积分方程时间步进算法稳定性研究[J]. 物理学报, 2013, 62(9):090206–1–090206–6

[4] 张三，李四.时域磁场积分方程时间步进算法后时稳定性研究.电子科技大学学报[J]（已录用，待刊）

[5] S. Zhang. Parameters Discussion in Two-Level Plane Wave Time-Domain Algorithm[C]. 2012 IEEE International Workshop on Electromagnetics, Chengdu, 2012, 38–39

[6] 张三, 李四. 时域积分方程时间步进算法研究[C]. 电子科技大学电子科学技术研究院第四届学术交流会, 成都, 2008, 164–168

[7] 张三（4）.人工介质雷达罩技术研究.国防科技进步二等奖，2008 年

[8] XXX， XXX， XXX， XXX， 王升. XXX的陶瓷研究. 四川省科技进步三等奖，2003年12月