第二届数学中国数学建模网络挑战赛
地址：内蒙古数学会　　网址：www.tzmcm.cn
电话：0471-4343756　　邮编：010021　　Email：ceo@madio.cn

第二届“数学中国杯”数学建模网络挑战赛

承　诺　书

我们仔细阅读了第二届“数学中国杯”数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛报名号为：1256

参赛队员　(签名)　：

队员 1：苏志峰

队员 2：龚雪萍

队员 3：王　杰

参赛队教练员 (签名)：　数模组

参赛队伍组别：本科组

第二届数学中国数学建模网络挑战赛
地址：内蒙古数学会 网址：www.tzmcm.cn
电话：0471-4343756 邮编：010021 Email：ceo@madio.cn

第二届“数学中国杯”数学建模网络挑战赛

编 号 专 用 页

参赛队伍的参赛号码：（请各个参赛队提前填写好）：
1256

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

第二届数学中国数学建模网络挑战赛
地址：内蒙古数学会　　网址：www.tzmcm.cn
电话：0471-4343756　　邮编：010021　　Email：ceo@madio.cn

2009 年 第二届“数学中国杯”
数学建模网络挑战赛

题　　目　**串行算法的并行化处理**

关 键 词　串行程序划分模型　负载均衡　并行化　RPDMA 算法

摘　　要：

由于网络和信息发展的需求，高性能机都采用了并行处理技术。而目前，各领域的应用积累了大量的串行程序，且很多领域对计算能力的需求越来越高，如何将串行程序并行化处理，以满足用户需要显得尤为重要。对于目前很多台式机和笔记本广泛的采用了双核和多核 CPU，本文通过建立模型实现串行程序并行化，使其能够在 CPU 两核心上并行运算，以提高总的算效率。对此问题，本文建立了两个数学模型分别为串行程序划分模型和关于负载均衡问题的模型。

模型一、串行程序划分模型，即解决如何将串行程序划分成可并行执行的部分。首先通过分析语句之间是否存在相关性，建立了理想化的串行程序并行划分的模型，即划分出来的程序段之间是完全不存在相关性，可以分到两个不同的 CPU 上独立进行。

但是实际中，绝大多数串行程序各个程序段中存在相关性，很少能将其划分成毫无相关的子程序，显然，若可以使各个划分程序段之间的相关程序降低，那么，也可提高双核 CPU 并行执行的效率。因此我们使用 RPDMA 算法，即降低程序段之间的相关性，建立了一般情况下串行程序并行划分模型，将串行程序划分成若干并行程序段。

模型二，关于负载均衡问题的模型，针对如何使划分出来的并行程序段能够合理的分配到不同的 CPU 中进行执行，以达到负载均衡效率最高的目标，我们建立了关于负载均衡问题的模型，利用 C 编程模拟计算机选择能够达到负载均衡效率最高的程序段组合。

参赛队号　1256

所选题目　A

参赛密码　_____
（由组委会填写）

第二届数学中国数学建模网络挑战赛
地址：内蒙古数学会 网址：www.tzmcm.cn
电话：0471-4343756 邮编：010021 Email：ceo@madio.cn

英文摘要（选填）

Abstract

Thanks to the need of the development of networks and information ,high-performance machines choose to use parallel processing technology.At present, the fields have accumulated a large number of serial programs, and the demand for computing power is getting higher and higher,How to change serial program to parallel processing to meet the users' needs is especially important.many desktops and laptops use dual-core and multi-core CPU widely, this paper,we establish the model of changing serial program to parallel program ,and enable the program to run on two CPU cores in parallel computing in order to improve the efficiency of the overall count.On this issue, this paper establish two mathematical models ,serial program division model and load balancing model

model 1 :serial program division model, that is, to solve how to change the serial program to parallel program.Firstly, through the analysis of whether there is correlation , we establish idealized model of serial program division.divied program which is completely non-existent between the correlation can be assigned to run two different CPU independently

In fact ,most of divied programs of serial programs have relationship among them ,Rare of them can be divied into subprograms which have no relationship .If we try to reduce the relationship, we can also raise the Nuclear CPU efficiency of parallel execution.So,we choose a kind of algorithm named RPDMA which can reduce the relationship,then a general model is established to change the serial program into parallel program

model 2:we establish the load balancing model which can range the different parallel program to different CPUs, To achieve the most efficient load balancing goal.we use a C programming to simulation the computing process of computer inorder to reach the high efficiency of load balancing.

报名号 # 1256

1．问题的重述

并行计算，是将一个计算任务分摊到多个处理器上并同时运行的计算方法。台式机和笔记本电脑现在也已广泛地采用了双核或多核 CPU。双核 CPU 从外部看起来是一个 CPU，但是内部有两个运算核心，它们可以独立进行计算工作。运行一个以常规的串行代码写成的程序时，如何将计算任务拆分成多个部分并分解到多个核心上同时运行，是很困难的事情。有时甚至需要人工来读懂原来代码的含义，并以适合并行计算的语言重写程序。这需要耗费大量的人力物力。最容易被并行化的计算任务称为“易并行”的，它可以直观地立即分解成为多个独立的部分，并同时执行计算。在软件行业中很自然地提出这个问题：希望设计一种方法，能够将一个常规的串行程序分解成两个部分，使之能够在 CPU 的两个核心上并行运算，并且希望能够尽量发挥双核心的计算能力。一般来说，如果两个核心的运算量基本相当，那么总的运算效率较高。

问题：请你们建立合理有效的模型，并依据模型对现成的串行算法进行处理。将能够使用双核心并行处理的部分分解开，并分配到两个核心上同时运行。以期达到比单核 CPU 处理更快速的目的。

具体要求：假设算法是使用 C 语言写成的，代码里只有顺序执行、分支、循环三种结构。为简单起见，我们假设只对整型变量和整型数组进行操作，不需要调用已有的库函数。程序中所有的语句只包括简单的代数运算、赋值、条件分支语句（if-else 语句），循环语句（while 语句）。不包括其他语句。

这里的关键是如何通过分析代码，从总的计算任务中尽量识别可独立运算的部分，并估计每部分的计算量，来尽量使双核 CPU 发挥出超过单核的效能。

2．模型的假设

①、串行程序本身具有正确性、可执行性。

②、程序使用 C 语言，代码中只有顺序执行，分支，循环三种结构，我们所处理的数据是整型变量和整型数组，不需要调用已用的库函数。

③、程序中所有语句只包括简单的代数运算、赋值、条件分支、循环语句。

3．符号规定

符号	符号说明
∇_c	控制相关
∇_f	流相关
∇_o	输出相关
∇_U	反相关
$\leftarrow$	相关于
Θ	并行划分模型
α,β	并行划分模型的任意两个子集
P	程序段集合
A,B	分公司 D1 总的销售额

L_i	程序中 i 语句
I_i	i 语句的输入变量的集合
O_i	i 语句的输出变量的集合
$\rho_{i,j}$	i 语句与 j 语句的相关程度
T	带权图
η	负载均衡效率
m	处理器个数
C_i	单个 CPU 运行的时间
t_i	单个并行子程序运行时间
sp	加速比
E	处理器利用效率

4. 问题的分析

从题目的要求中，知道了题目要求串行算法的并行化处理，以提高计算机执行效率，即将一个常规的串行程序进行分段，产生并行子程序，分别放入双核 CPU 中运行，使 CPU 尽量发挥双核的计算能力，达到提高计算效率的目的。在串行程序并行化及分配 CPU 的过程中，我们必须解决两个问题：

①、如何将一个串行程序划分成并行子程序

②、对划分的子程序，如何分配两个 CPU 上，使 CPU 负载均衡，以达到总的运输效率较高即负载均衡效率最大。

对于问题①，我们必须考虑到只有理想串行程序可以完全划分成毫无相关的并行程序模块，而实际上，大多数串行程序不能被划分成多个不相关的程序段。各个划分程序相互之间都存在一定的相关程度，这里我们采用 RPDMA 算法来降低并行程序相关程度，实现串行程序并行化。

对于问题②，如何将并行任务分配到两个 CPU 中进行，一般来说，如果两个核心的运算量基本相等，那么总运算效率较高，即实现两个 CPU 负载均衡。这里，我们同样设计一 C 算法实现，根据并行程序运行时间，将并行程序最大程度均衡分配到两个 CPU 中，使均衡效率达到最大，提高 CPU 计算效率。

在最简单的情形下，并行化处理是使用多个计算要求去解决可计算问题。主要要求如下：

①、用双核 CPU 来运行；

②、问题被分解成离散的部分可以被两个 CPU 独立同时解决；

③、每一串行程序被细分成一系列独立程序段；

④、双核处理时间的计算与单核处理时间计算相比较；

⑤、如何使其负载均衡。

5. 模型的建立与求解

5.1 串行程序并行划分模型

5.1.1 模型相关符号的定义[4][2]

定义 1 控制相关 ∇_c：如果程序段 B（语句 j）的执行依赖于程序段 A（语句 i）（i 必须

在 j 之前执行),则程序段 B（语句 j）与程序段 A（语句 i）控制相关；

定义 2 流相关∇_f：如果从程序段 A（i）到程序段 B（j）存在执行通路,而且如果程序段 A（i）至少有一个输出可供程序段 B（j）用作输入,则程序段 B（语句 j）与程序段 A（语句 i）流相关；

定义 3 输出相关∇_o:如果程序段 A，B（i,j 两语句）能产生(写)同一输出变量,则两者是输出相关；

定义 4 反相关∇_U:如果程序段 B（语句 j）紧接程序段 A（语句 i）,而且如果的程序段 B（j）输出与的程序段 A（i）输入重叠,则程序段 B（语句 j）语句 j 与程序段 A（语句 i）反相关；

定义 5 对于程序段 A，B 之间的相关性用$\leftarrow$表示即定义为

$$A \leftarrow B \Leftrightarrow (A\nabla_f B)\cup(A\nabla_o B)\cup(A\nabla_u B)\cup(A\nabla_c B)$$

定义 6 对于并行划分模型Θ，α,β是并行划分模型的任意两个子集，那么α,β中的程序分别放在两个独立的 CPU 上进行。

5.1.2 理想情况下的串行程序并行划分模型

5.1.2.1 模型的建立

对于理想情况下，我们假设串行程序可以划分成完全相互独立的子程序，使其在不同的 CPU 上并行执行。

[2]程序段集合 P 的并行划分模型为Θ，其任意两个子集

α,β，$\forall_{\alpha,\beta}\in\Theta,(\alpha\neq\beta),\alpha\cap\beta=\phi,\forall A\in\alpha,\forall B\in\beta$，A、B 不存在相关性。

由Θ的定义（定义 7）可知：每两台处理机上执行的程序段部分不存在相关性，CPU 运行时无任何消息传递和时间等待，一直到各 CPU 上的程序段执行完毕。

假设程序中的任何一个程序段不会被执行多于一次，程序中每一个程序段都可以被执行。

5.1.2.1 模型的求解与分析

这里我们定义一个包含若干程序段的集合 P，设每个元素$L_i(1\le i\le n)$为程序中的一行代码，则得到[3]

$$P=\{L_1,L_2,\ldots\ldots L_n\},$$

$$\forall_{\alpha,\beta}\in\Theta,(\alpha\neq\beta),\alpha\cap\beta=\phi,\forall A\in\alpha,\forall B\in\beta$$

$$\mathop{Y}_{\alpha\in\Theta}\alpha = P;$$

分划模型Θ中的任意两个子集α和β,这两个子集中分别任意取的一个元素 A 和 B，A 和 B 不存在相关性。

下面用一个简单的例子，程序$P=\{L_1,L_2,L_3,L_4,L_5,L_6,L_7,L_8,L_9\}$;为

例 1：

L_1:a=2;

L_2:b=a+5;

L_3:x=3;

L_4:y=2x;

L_5:if(b>a)

{

L_6:a++;

}

Else

L_7:b--;

L_8:While(x<y)

{

L_9:x++;

}

通过其相关性的判断，我们可以画出该程序的进程流图如下（图 1，图 2）：

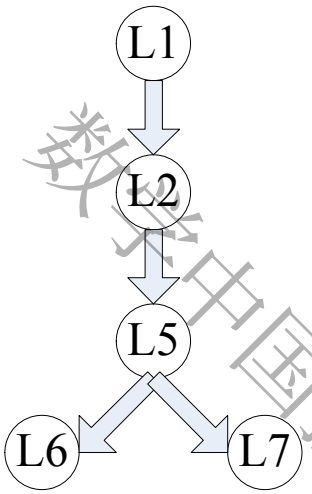

图 1

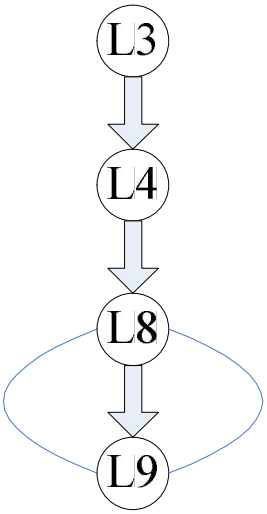

图 2

由图可以看到，根据规则程序 P 的划分 $\Theta=\{\alpha_1,\alpha_2\}$，

其中 $\alpha_1=\{L_1,L_2,L_5,L_6,L_7\},\alpha_2=\{L_3,L_4,L_8,L_9\}$，很显然，$\alpha_1,\alpha_2$ 中的元素 L_1 之间不存在相关性。因此，程序 P 可以按照分划 Θ 分派到两个 CPU 上独立执行，那么在这两个 CPU 上执行的具体子程序如下(表 1，表 2)：

CPU1

行号	P_1
L_3	x=3;
L_4	y=2x;
L_8	While(x<y)
L_9	x++;

表 1

CPU2

行号	P_2
L_1	a=2;
L_2	b=a+5;
L_5	if(b>a)
L_6	a++;
L_7	b--;

表 2

5.1.3 一般情况下串行程序并行划分模型

5.1.3.1 模型的建立

理想中的串行程序可以划分或完全相互独立的模型，使其在多个 CPU 上并行执行，但是实际中，绝大多数串行程序各个程序段中存在相关性，很少能将其划分成毫无相关的子程序，显然，若可以使各个划分程序段之间的相关程序降低，那么，也可提高双核 CPU 并行执行的效率。

程序段的相关程度 ρ

$\rho_{BA}=|I_A \cap O_B|+|I_B \cap O_A|+|O_A \cap O_B|+\nu(B\nabla_c A)$

$B\nabla_c A$ 为真，$\nu(B\nabla_c A)$则为 1，否则为 0；

5.1.3.1 模型的求解与分析

这里，我们采用了一个降低相关程度的并行划分算法 RPDMA，其主要思想：将串行程序中每个语句标识为 $L_i(i=0.....n)$;作为节点，它们之间的相关程度作为各个节点的边的权值，从而形成了包含程序语句集合以及它们相互之间相关性的带权图 T。

[2]RPDMA 具体算法是：首先找到 T 中权值最小的边 v，然后让每条边的权都减去 v，如果产生的新图是非连通图，则表明串行程序划分成带有一定相关性的子程序，如果新图是连通图，则再按照上述方法进行划分直到产生新图为非连通图。

先给出算法符号定义：T 是一个集合，集合中每个元素是一个三元组(A, B,v),A 和 B 是并行划分模型Θ的程序段，s 表示一个任意权值，v 是 A 和 B 的相关程度。mv 表示最小带权值。

算法描述如下:

a、生成 $T=\{(A,B,\nu)|A,B\in P;\nu=\rho_{BA}\}$

b、使得 $\forall s\in T, s\neq mv, s.v>mv.v;$

d、令 $r.v:=r.v-mv.v,$

e、若 $T\neq\phi$(非连通图)，则转 d;

f、若 T 包含两个或两个以上连通图，即 $T_1,T_2,.....T_n$；

h、$\Theta=(T_1\cup T_2.....T_3)$;

对于模型的具体求解如下：

一个串行程序，一般情况下，包含了 n 条语句，我们设这 n 条语句在一个程序集为 $P=\{L_1,L_2.....L_n\}$;

I_i是指 i 语句输入所需要变量的集合，O_i是指 i 语句输出所需要变量的集合;例如：语句 z=x+y;则 $I_i=\{x,y\}$，$O_i=\{z\}$；

通过输入输出之间的相关性，我们求出前面定义的相关度，根据条件我们对语句 $L_j(j=1.....n)$与之间的语句 $L_i(i=1.....n)$求相关程度

$\rho_{ji}=|I_i\cap O_j|+|I_j\cap O_i|+|O_i\cap O_j|+\nu(L_j\nabla_c L_i)$;

以每个语句 $L_i(i=1.....n)$作为节点，以语句之间的相关程度 ρ_{ji} 作边权值，作带权图，其相应的邻接矩阵如下：

$$\begin{bmatrix}L_1\\L_2\\.\\.\\L_n\end{bmatrix}=\begin{bmatrix}0 & & \rho_{L_1L_n}\\ \rho_{L_2L_1} & & \rho_{L_2L_n}\\ . & \cdots\cdots\cdots\cdots\cdots & \\ . & & .\\ \rho_{L_nL_1} & & 0\end{bmatrix}$$

（语句 i 与其本身的相关性为 0）

在 $\rho_{ij}(i=1.....n, j=1.....n)$ 的集合中找出一最小的相关度权值 $\rho_{\min}$，使得Θ中$\rho_{ij}\geq\rho_{\min}$，同时使得Θ中$\rho_{ij}=\rho_{ij}-\rho_{\min}$；

设程序集为 $P=\{L_1,L_2,L_3.....L_n\}$; α 为并行划分模型的一个子集，设 $\alpha=\phi$（即划分前没有任何语句）；若节点 L_i 和相关程度 ρ_{ji} 形成的带权图在经过上面的操作过后没

有划分成非连通图，则转上面操作，直至带权图划分成非连通图即可（即划分成了若干个子集α_i，α_i子集即是划分模型划分出来的可独立放入不同 CPU 中进行执行的程序段。

实例如（例 2）：

例 2

L_1:x=2;

L_2:y=5*x;

L_3:z=2x+y;

L_4:p=x+y+z;

L_5:if(p>10)

{

L_6:P--;

}

L_7:While(z<20)

{

L_8:z++;

}

我们可以看到，这个程序中各程序段的输入输出集为：

$I_{L_1}=\phi$，　　$O_{L_1}=\{x\}$;

$I_{L_2}=\{x\}$，　　$O_{L_2}=\{y\}$;

$I_{L_3}=\{x,y\}$，　　$O_{L_3}=\{z\}$;

$I_{L_4}=\{x,y,z\}$，　　$O_{L_4}=\{p\}$;

$I_{L_5}=\{p\}$，　　$O_{L_5}=\phi$;

$I_{L_6}=\{p\}$，　　$O_{L_6}=\{p\}$;

$I_{L_7}=\{z\}$;　　$O_{L_7}=\phi$;

$I_{L_8}=\{z\}$;　　$O_{L_8}=\{z\}$;

求出它们之间的相关程度:

$\rho_{L_2L_1}=1$;　　$\rho_{L_3L_2}=1$;

$\rho_{L_3L_1}=1$;　　$\rho_{L_4L_3}=1$;

$\rho_{L_4L_2}=1$;　　$\rho_{L_4L_1}=1$;

$\rho_{L_5L_4}=1$;　　$\rho_{L_5L_3}=0$;

$\rho_{L_5L_2}=0$;　　$\rho_{L_5L_1}=0$;

$\rho_{L_6L_5}=2$;　　$\rho_{L_6L_4}=2$;

$\rho_{L_6L_3}=0$;　　$\rho_{L_6L_2}=0$;

$\rho_{L_6L_1}=0$;　　$\rho_{L_7L_6}=0$;

$\rho_{L_7L_5}=0$;　　$\rho_{L_7L_4}=0$;

$\rho_{L_7L_3}=1$;　　$\rho_{L_7L_2}=0$;

$\rho_{L_7L_1}=0$;　　$\rho_{L_8L_7}=2$;

$\rho_{L_8L_6}$=0;　　$\rho_{L_8L_5}$=0;

$\rho_{L_8L_4}$=1;　　$\rho_{L_8L_3}$=2;

$\rho_{L_8L_2}$=0;　　$\rho_{L_8L_1}$=0;

由它们的相关程度可以画出带权图如下（图 3）：

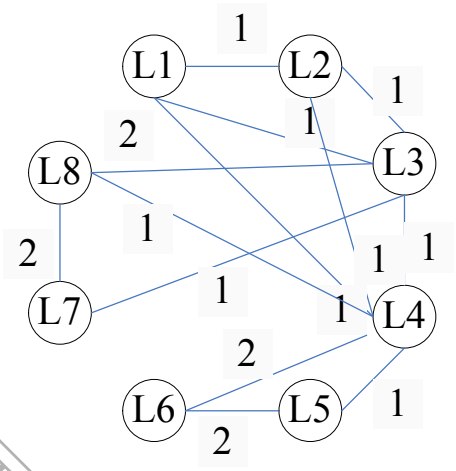

图 3

根据数学模型中的相关性降低计算，可以将此带权图，简化为四个连通子图如下（图 4）：

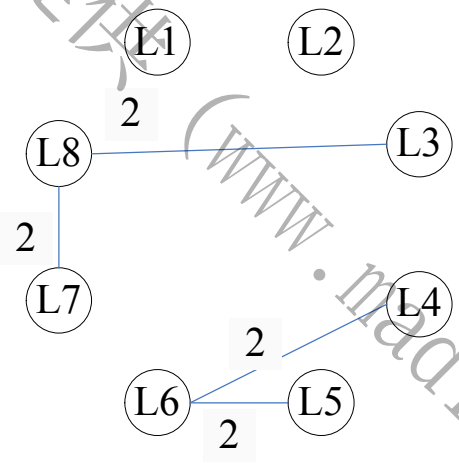

图 4

因此，我们可以划分 $P=\{L_1,L_2,L_3,L_4,L_5,L_6,L_7,L_8\}$ 为 $\Theta=\{\{L_1\},\{L_2\},\{L_3,L_7,L_8\},\{L_4,L_5,L_6\}\}$。

5.2 关于负载均衡问题的模型

在上述过程中，我们讨论了如何将一个程序划分为若干个程序段的模型，以上我们采用 RPDMA 算法将串行程序划分成了具有一定相关性但可独立执行的若干子程序，但是因为并行模型中的子程序段个数不确定，如果是 2 个子程序段，那就直接将其分别放到不同的 CPU 中进行执行即可；但若是 n 个子程序段，那么我们就考虑如何组合这 n 个独立的子程序段，使其分为两个计算量尽量相当的程序段，放入到双核 CPU 中独立运行，使其达到负载均衡。

设 CPU 负载均衡效率 η

$$\eta=\frac{\sum_{i}^{m}C_i}{\max\limits_{i=1}^{m}\{C_i\}\times m};$$

η指某个期间内机群的负载平衡效率

m 指 CPU 的个数；

C_i指在该期间内第 i 个 CPU 执行完程序花费的时间（在该 CPU 上运行的并行子程序的时间之和）；

在我们的问题中，因为是双核运算，那么如果两个核心的运算量基本相当，那么 CPU 总的运算效率最大，负载均衡效率达到 1。

对于并行计算的 CPU 执行，CPU1 执行一个程序段花费的时间为C_1，CPU2 执行另外一个程序段花费的时间为C_2，则有

$|C_1-C_2|\to 0$时，负载平衡效率η达到最大为 1。

那么如何使其尽量接近呢？我们可以假设划分出来的并行子程序各自运行的时间$t_1,t_2.....t_n$；

这里对于划分出来的并行子程序各自运行时间$t_1,t_2.....t_n$的估算我们需要说明如下：

一个程序段的计算时间通常是语句的执行的次数来衡量的，因此我们也通过分析每个划分子程序中语句的执行次数来估算每个子程序的计算量，由于我们这里考虑的源程序只包括基本赋值，代数运算的语句，条件语句以及循环语句。所以我们把串行程序中的语句分为两种：

①基本语句：赋值及代数运算语句；

②复合语句：条件及循环语句。

假设一个程序中包含n_1条基本语句，n_2条 if…else 语句，n_3条 while 语句，其中 if…else 语句中包含$p_i(i=1,2.....n_2)$个基本语句，while 语句中包含$q_j(j=1,2.....n_3)$个基本语句，且循环次数为$N_j(j=1,2.....n_3)$，每个基本语句的执行时间为Δt，则该程序的执行时间为$t=n_1\times\Delta t+(\sum_{i=1}^{n_2}p_i)\times\Delta t+(\sum_{j=1}^{n_3}q_jN_j)\times\Delta t$

要达到负载平衡效率$\eta\to 1$，则要使集合$U=\{t_1,t_2.....t_n\}$中的元素任意组合成两部分，并使两部分各自之和C_1，C_2满足$|C_1-C_2|\to 0$，于是我们用一 C 程序（见附录）来实现这一目的。

具体算法思想如下：

①取集合U中一元素$t_{\max}$，使$\forall t_i\le t_{\max}$，放入集合V_1中，同时$U=U-t_{\max}$；

②在集合U中取一元素$t'_{\max}$，使$\forall t_i\le t'_{\max}$，放入集合V_2中，同时$U=U-t'_{\max}$；

③比较V_1，V_2中元素之和S_1，S_2，若$S_1<S_2$，重复①步骤，否则重复②步骤，直到集合$U=\phi$；

④集合V_1，V_2是将集合U划分成比较均衡的两部分，即可放入到 CPU1 和 CPU2 中分别运行，以达到负载均衡效率最高。

通过以上分析，我们已将模型 1 中的 n 个并行子程序组合成两个部分，且达到了负载均衡效率最高，实现模型的最优化。

报名号　#　1256

6．模型的结果分析与检验

6.1 模型的检验

上面建立的**理**想情况下的串行程序并行划分模型和一般情况下的串行程序并行划分模型是否合理，已经在上述模型建立和分析中给出实例，能得到合理结果，证明其模型是正确的。

下面通过分析串行程序例 2 来检验负载均衡模型是否合理。

根据一般情况下的串行程序并行划分模型将串行程序例 2 划分为了四个并行子程序如下：

$\Theta = \{\{L_1\},\{L_2\},\{L_3,L_7,L_8\},\{L_4,L_5,L_6\}\}$

通过负载均衡效率模型，分别计算每个部分的执行时间

$t_1 = \Delta t, t_2 = \Delta t$，　$t_3 = 2\times\Delta t$，　$t_4 = 2\times\Delta t$；

再利用上述负载均衡效率模型，将 4 个程序段组合成两部分分配到双核 CPU 上执行，程序运行结果如下（设 Δt=1）：

由上述运行结果可知：

可分为如下组合 $CPU1:\{\{L_1\},\{L_3,L_7,L_8\}\}$，$CPU2:\{\{L_2\},\{L_4,L_5,L_6\}\}$

或组合 $CPU1:\{\{L_1\},\{L_4,L_5,L_6\}\}$，$CPU2:\{\{L_2\},\{L_3,L_7,L_8\}\}$

这两种组合的负载均衡效率是相同的。

于是，可计算双核 CPU 中 CPU1 执行时间 $C_1 = 3\Delta t$，CPU2 执行时间 $C_2 = 3\Delta t$，由以上给出的 CPU 负载均衡效率 η

$$\eta = \frac{\sum_{i}^{m} C_i}{\max_{i=1}^{m}\{C_i\}\times m};$$

得出负载均衡效率 $\eta = \dfrac{\sum_{i}^{m} C_i}{\max_{i=1}^{m}\{C_i\}\times m} = \dfrac{3\Delta t + 3\Delta t}{3\Delta t\times 2} = 1$，此时串行程序并行化处理后在双核的速度达到单核 CPU 处理的一倍，均衡负载达到理想化。

6.2 模型的性能分析

问题是让我们建立合理有效的模型，并依据模型对现成的串行算法进行处理，将能够使用双核心并行处理的部分分解开，并分配到两个核心上同时运行。以期达到比单核 CPU 处理更快速的目的。因此评价模型的性能指标有如下：

运行时间：1、一个程序的串行运行时间使程序在一个处理器上开始执行到执行完成所经过的时间段长度，记为 T_s。

2、并行运行时间则定义为并行计算开始到最后一个处理器完成的任务

的时间长度，记为T_p。

加速比：指并行算法（或并行程序）的执行速度相对于串行算法（或串行程序）的执行速度加快多少倍，定义为$sp=\frac{T_S}{T_p}$；

效率：用来衡量一个处理器的计算能力被有效利用的比率，定义为 E=sp/m（sp加速比 m 处理器个数）。

由以上可知：串行程序执行时间：$T_s=C_1+C_2=6\Delta t$；而取C_i中的较大者即并行运行时间$T_p=3\Delta t$。

则加速比$sp=\frac{T_S}{T_p}$ =2；

效率E=sp/m=1。

由以上性能指标说明例 2 中的程序在理想情况下（CPU 没有其他进程运行等）双核同时运行，能够达到比单核处理更快速的目的，且能提高一倍。

7．模型评价

7.1 优点

串行程序划分模型

⑴从理想的（程序段之间完全不存在相关性）并行划分模型到一般模型，考虑到了实际中绝大多数串行程序各个程序段中存在相关性，很少能将其划分成毫无相关的子程序；

⑵采用了 RPDMA 算法使各个划分程序段之间的相关程序降低，划分若干并行子程序段 ；

⑶可以根据模型，有效地得到若干个可独立执行的程序段。

关于负载均衡问题的模型

⑴使用负载均衡效率公式作为基本模型，从而直观的得到负载均衡效率；

⑵简化了程序计算量的运算；

7.2 缺点

串行程序划分模型

⑴此模型建立在理论基础上，带有一定的主观性；

⑵对于进程之间的相互通信未作考虑。

关于负载均衡问题的模型

⑴程序中基本语句的计算量有差异，但在本模型中将其计算量视为相等。

8．参考文献

[1]姜启源，谢金星，叶俊，数学模型（第三版），北京：高等教育出版社，200308。

[2]江文毅，庞丽萍，高兰，韩宗芬，串行程序中的并行划分算法研究，华中理工大学学报，卷期号：第 28 卷 12 期，200012。

[3]江文毅，串行程序并行化的探讨，99 青岛-香港国际计算机会议论文集（上册），中国计算机学会，267-270，199910。

[4]蒋作，高毅，关于串行程序并行化，云南民族大学学报（自然科学版），卷期号：第 16 卷 3 期，200707。

附录

C 程序代码如下：

```
#include<stdio.h>
#define N 4
int sum(int *p,int n)
{
    int s=0,i;
    for(i=0;i<n;i++)
        s+=*(p+i);
    return s;
}
void balance(int *arr,int n)
{
    int i,s1,s2;
    printf("并行子程序各自运行时间={");
    for(i=0;i<n;i++)
        printf(" %d",*(arr+i));
    printf(" }\n");
    //s=sum(arr,n);
    s1=arr[n-1];
    s2=arr[n-2];
    printf("在 CPU1 上运行的子程序={ %d ",arr[n-1]);
    for(i=0;i<=n-3;i++)
    {
        if(s1>s2)
        {
            //t2[i+1]=t[N-3-i];
            s2+=arr[n-3-i];
        }
        else
        {
            //t1[i+1]=t[N-3-i];
            s1+=arr[n-3-i];
            printf(" %d ",arr[n-3-i]);
        }
    }
    printf(" }\n");
    printf("剩余子程序在 CPU2 上运行\n");
}
void main()
{
    int t[N]={1,1,2,2};
    balance(t,N);
}
```