# 计算科学与工程中的并行编程技术

# Parallel Programming Technology in Computational Science and Engineering

都志辉

清华大学计算机系

Email：duzh@tsinghua.edu.cn

Phone: 62782530

http://hpclab.cs.tsinghua.edu.cn/~duzh

# 主要内容

- 消息调度
- 消息传递过程中的类型匹配
- 不连续数据发送
- 虚拟进程拓扑

# 什么是消息调度？

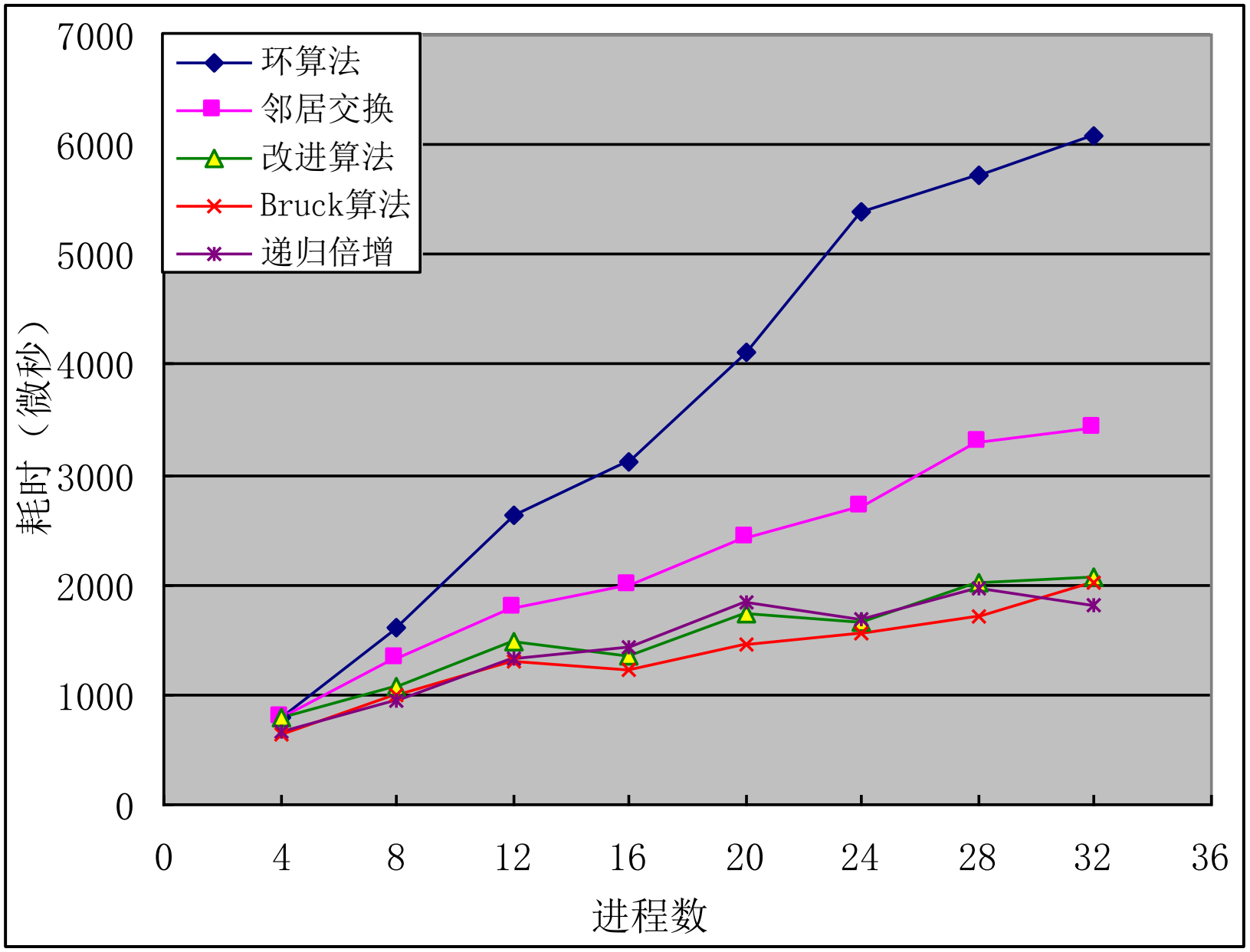
- 对于有确定目与源的一组消息，通过对消息传递的时间、路径、数量等进行规划，以满足消息传递在性能或者费用等方面的要求

# 时间开销建模

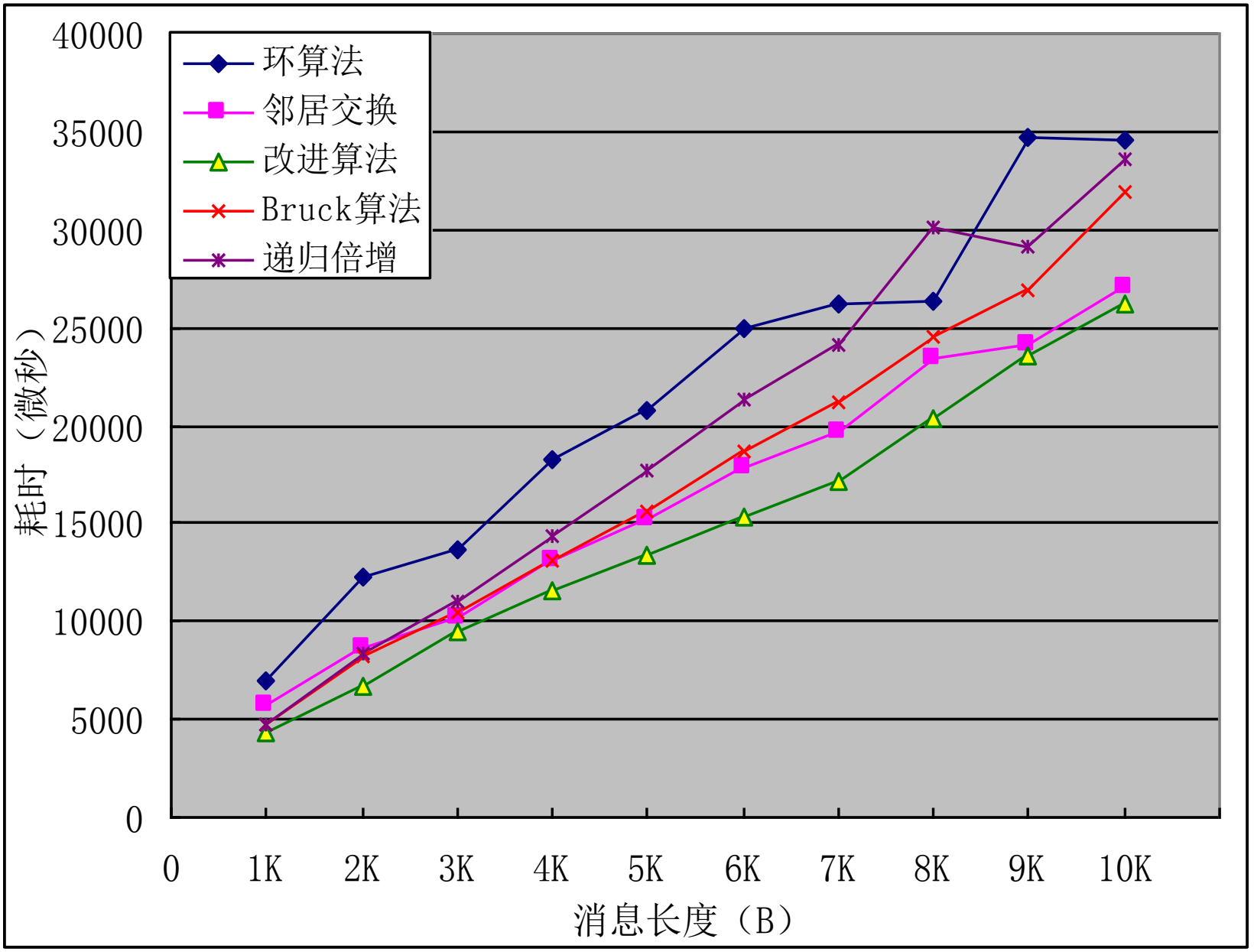
一次消息传递的时间开销

=delay+volumn/bandwidth

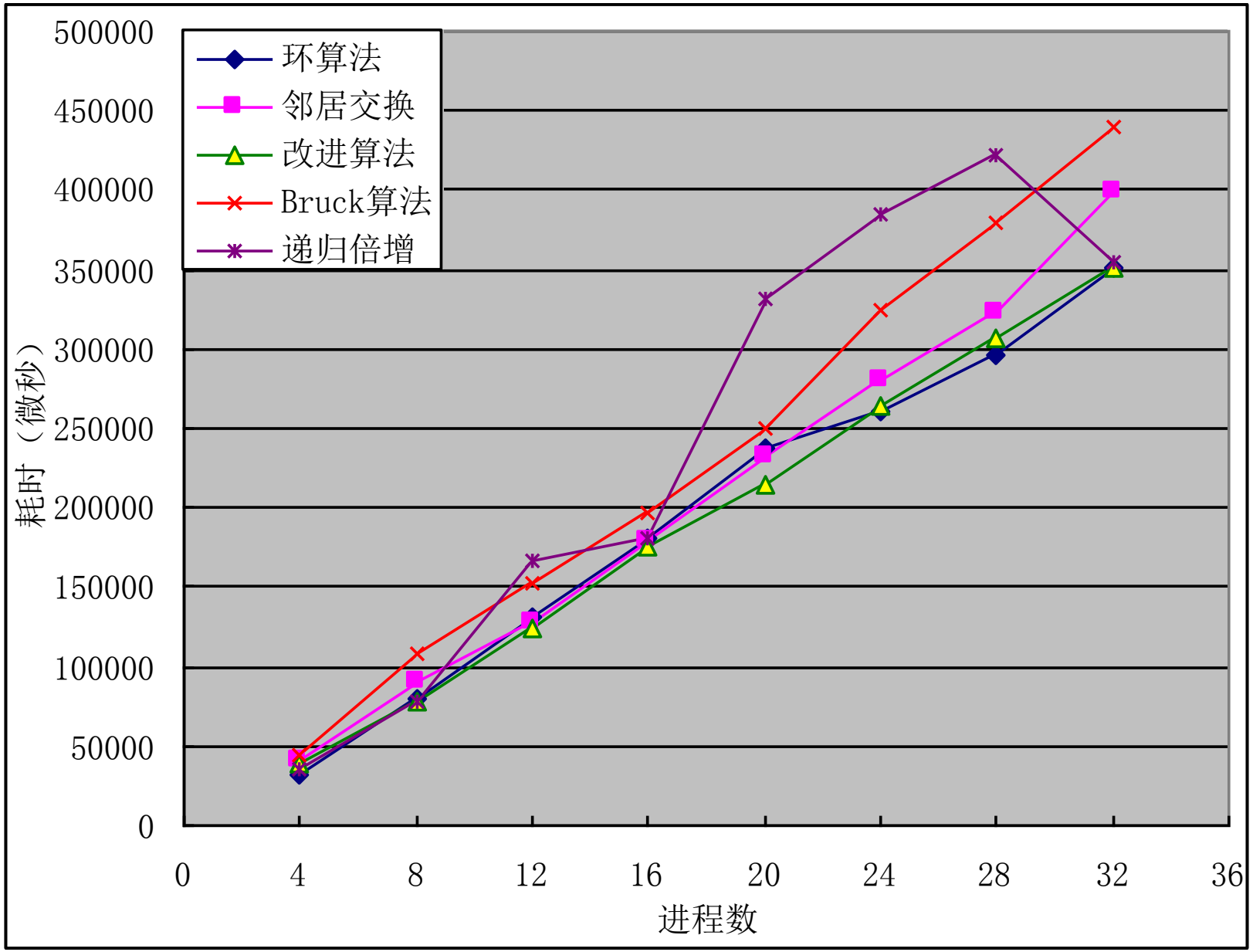
# 消息调度的效果（短消息）


环算法
邻居交换
改进算法
Bruck算法
递归倍增
耗时（微秒）
进程数
0
1000
2000
3000
4000
5000
6000
7000
0
4
8
12
16
20
24
28
32
36

# 消息调度的效果（长消息）


环算法
邻居交换
改进算法
Bruck算法
递归倍增
40000
35000
30000
25000
20000
15000
10000
5000
0
耗时（微秒）
0
1K
2K
3K
4K
5K
6K
7K
8K
9K
10K
消息长度（B）

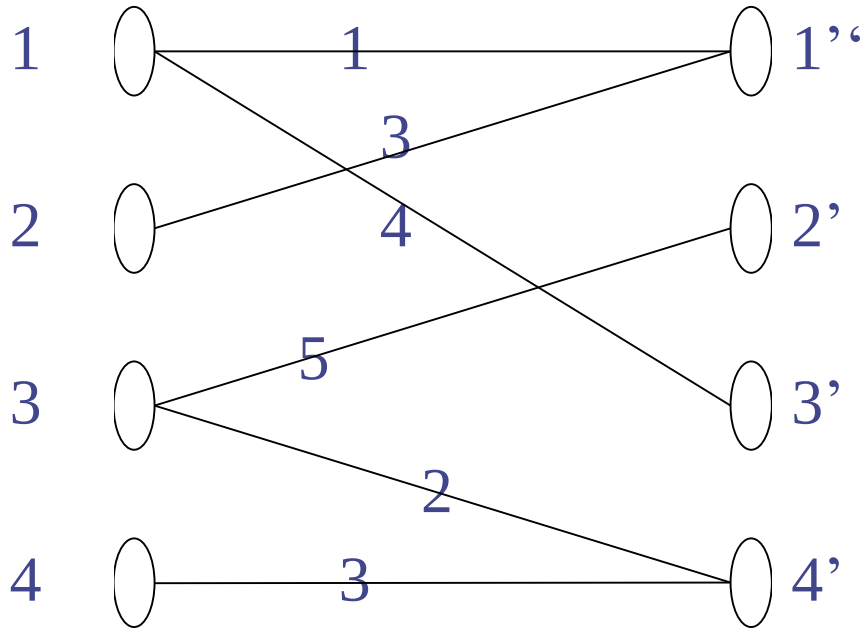
# 消息调度的效果（中等长度消息）



# 如何抽象消息调度问题

## 消息调度模型

# 二部图模型




- G=(V1,V2,E,W)
  - V1:左边结点的集合
  - V2:右边结点的集合
  - E:边的集合
  - W:权重的集合

矩阵其实只是图的一种实现

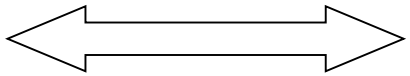
# 矩阵模型

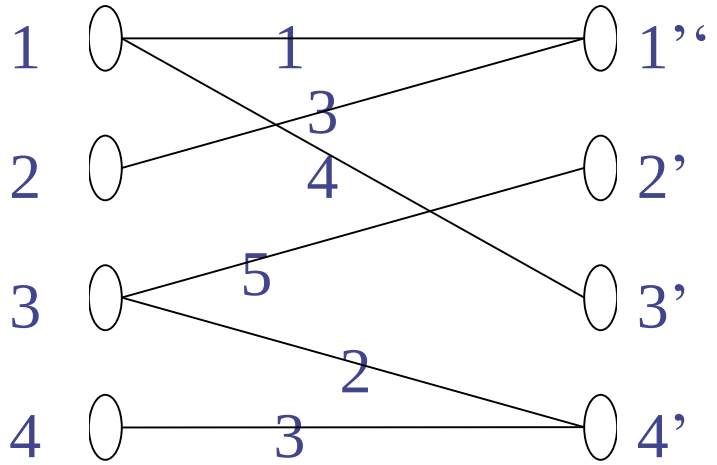
$$A=\begin{pmatrix}1 & 0 & 4 & 0\\ 3 & 0 & 0 & 0\\ 0 & 5 & 0 & 2\\ 0 & 0 & 0 & 3\end{pmatrix}$$

- 矩阵 $A=(a_{ij})_{M\times N}$
  - Aij:表示从结点i到结点j有大小为aij的数据需要传输

# 两种模型之间的关系

$$A=\begin{pmatrix}1 & 0 & 4 & 0\\ 3 & 0 & 0 & 0\\ 0 & 5 & 0 & 2\\ 0 & 0 & 0 & 3\end{pmatrix}$$





# 问题？

- 前提
  - 同一时刻一个结点只能与另外一个结点通信
  - 所有结点之间的通信延迟为0，带宽都相同
  - 互不相同的结点之间可以并行通信
- 是否存在下界（最短的通信时间）
- 如果存在，它是多少？

# 最优算法

$$S = \max(\lceil T/K \rceil, C)$$

$$C = \max(r_1, r_2, \ldots, r_N; c_1, c_2, \ldots, c_N)$$

- 对于给定的通信矩阵D，计算其下边界S，其中ri为第i行元素之和，cj为第j列元素之和。
- 5个定义
  - 定义1 如果一个矩阵某一行/列之和等于边界S，则该行/列成为关键行/列
  - 定义2 对于D的K个元素α={x1,x2,…,xK}，如果它们处在不同的行与列上，则称α为D的一个SKR（System of K Representatives）
  - 定义3 如果D的每一个关键行/列都包含某个SKR的元素，则称该SKR为CSKR（Critical SKR）
  - 定义4 对于N×N的矩阵D，其所有的通信量为T，对于常数P，如果 1 D 的所有行/列之和都小于或者等于P；2 T=KP，则称D关于常数P是K完全的。
  - 定义5 对于N×N的矩阵D，如果其所有行/列之和都相同，则称其为准双随机矩阵QDS（Quasi-doubly stochastic matrix）

G. Bongiovanni, D. Coppersmith, and C.K. Wong, "An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders," IEEE Trans. Comm., vol. 29, no. 5, pp. 721-726, 1981

# 最优算法的存在性

- 定理1
  - 给定通信开销为T,边界为S的N×N矩阵D，可以通过在D的不同元素上添加一些非负的整数来得到一个新的N×N矩阵D’，它相对于S是K完全的
- 推论：
  - 任何H×H的QDS矩阵D，它至少包含一个SHR。
- 定理2
  - 设D为一个N×N的矩阵，其总通信量为T，下边界为S，对于任意K，1=<K<=N，如果D相对于S是K完全的，则D至少包含一个CSKR
- 定理3
  - 设D是一个关于边界S的N×N的K完全矩阵，令α={x1,x2,…,xK}是D的一个CSKR，令L是D中没有被α覆盖到的所有各行或者各列之和中的最大值，令D’为将D中α代表的元素减去b得到的新矩阵，其中b=min(S-L, x1,x2,…,xK)。则D’是一个相对于S’=S-b的K完全矩阵，而且S’是新得到的矩阵D’的边界

# PBS（Preemptive Bipartite Scheduling）问题近似算法

- 2近似算法 P. Crescenzi, D. Xiaotie, and C.H. Papadimitriou, "On Approximating a Scheduling Problem," J. Combinatorial Optimization, vol. 5, pp. 287-297, 2001
  - 首先将任意属于E的边e分割为$\lceil \omega(e) \rceil$ 条边，把结果图称为H
  - 在H中找到Δ(H)个匹配，每一个匹配的权重都为1
- 2-1/(d+1)近似算法 F.N. Afrati, T. Aslanidis, E. Bampis, and I. Milis, "Scheduling in Switching Networks with Set-Up Delays," J. Combinatorial Optimization, vol. 9, no. 1, pp. 49-57, 2005
  - 将二部图G=(V1,V2,E,ω)所有边通过增加尽可能少的数值，变为给定值α＝d+1的倍数，得到的新图为G'=(V1,V2,E,ω')
  - 把G'的每一条边eij分裂为$\frac{\omega'(e_{ij})}{\alpha}$ 条边，每一条边的权重都为α，称新得到的图为Gα
  - 在Gα中找到$\frac{W(G_\alpha)}{\alpha}$ 个匹配，可以覆盖Gα中所有的边

# k-PBS（k-Preemptive Bipartite Scheduling）问题 近似算法

- 通信的并发度为k
- 下界的计算 $\eta(G) = \eta_d(G) + \beta\eta_s(G)$
  - Δ(G)为二部图G的最大度数，m(G)为二部图G的边的条数，w(v)为与v连接的边的权重之和，w(G)为G中最大的权重之和，p(G)为所有边权重之和，n(G)为二部图G的结点的个数。其中β是一次传送所需要的额外开销。

$$\eta_s(G) = \max(\Delta(G), \left\lceil \frac{P(G)}{k} \right\rceil) \qquad \eta_d(G) = \max(W(G), \frac{P(G)}{k})$$

# 这些模型的缺点

过于理想化，没有考虑通信的局部性

# 通信局部性

- 如何把通信尽可能在同一个交换机内部完成？（物理通信路径最短？）
- 假设
  - 在一个局部单元内部的通信时间为T0
  - 需要一级跳跃的通信时间为T1
  - 需要两级跳跃的通信时间为T2
  - …
  - i>j则Ti>Tj

# Broadcast广播（树形扩散算法）

- 假设局部单元内有K个结点，整个系统共有N个结点
- 广播步骤
  - 局部单元先广播 $\lceil \log_2 K \rceil$*T0→K
  - 局部单元结点向不同的一级跳跃结点广播$\lceil \log_2 K \rceil$*T0+T1+ $\lceil \log_2 K \rceil$*T0→K*K
  - 向二级跳跃阶段广播$\lceil \log_2 K \rceil$*T0+T1+ $\lceil \log_2 K \rceil$*T0+T2+ $\lceil \log_2 K \rceil$*T0 →K*K*K（上一级结点的一级跳跃结点）

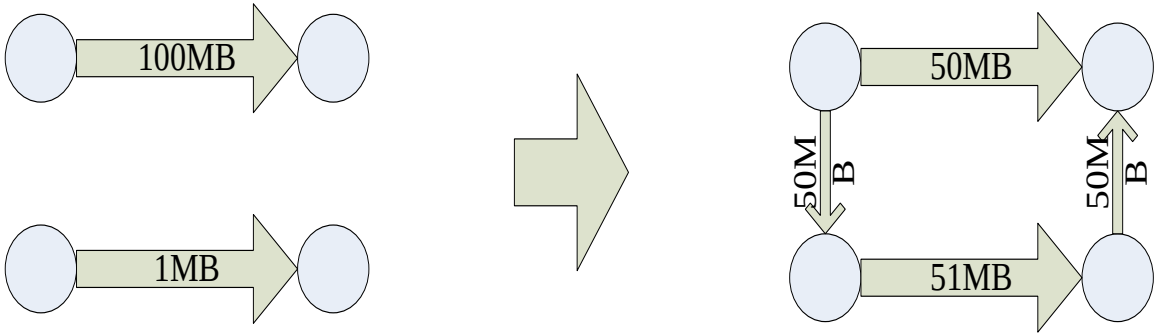
# 引入内部消息调度机制的集群间并行数据重分布

# 问题描述

- 两个集群间数据重分布
  - 骨干网连接
  - 1-port constraint
  - K-constraint
  - 启动延迟，消息可拆分
    - KPBS问题
  - 已经证明为NPC问题

# 现有模型和算法的缺陷

- 高速内部互联通信
  - 内部互联通信速度远高于backbone通信速度（绕路是否一定慢？）。
  - 当数据重分布不平衡时候，珍贵的backbone资源没有得到有效的利用
  - Case: k=2,n1=n2=2



# 基于内部通信的集群间消息调度模型

- MKPBS (Multiple k-Preemptive Bipartite Scheduling )

  <1>建立二部图GD=(U,V,E) ,表示集群间消息传输

  <2>建立全相连图GU=(U,E')。表示集群U中内部消息通信。

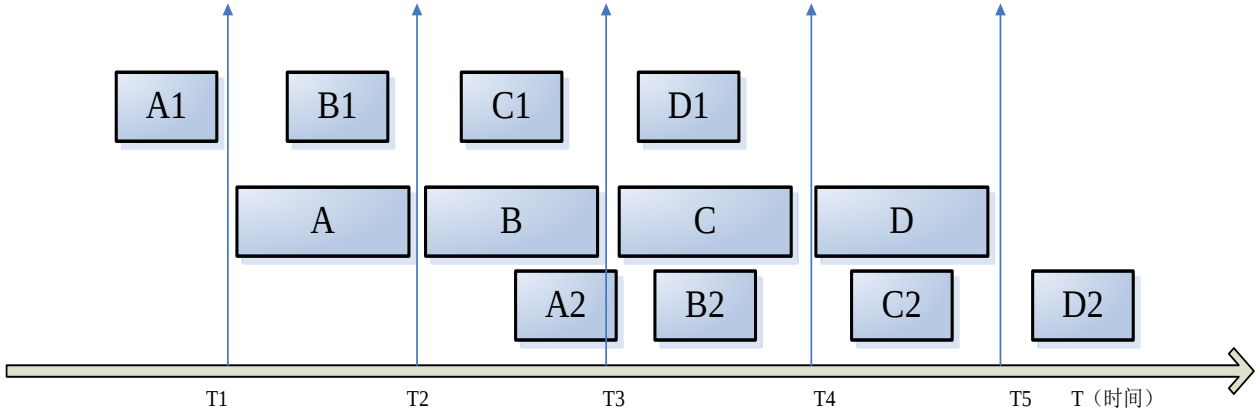
  <3>建立全相连图GV=(V,E'')。表示集群V中内部消息通信。

  <4>内部消息通信约束:

  - 1-port constraint
  - K-Constraint
  - 启动延迟，消息可拆分

# MKPBS模型的优点

- 更准确的刻画了集群的通信特点，集群内部实现消息的拆分与合并。
- 对消息传输线路进行重新规划，并采用消息流水线的方式对消息进行传输。



# 基于消息流水线的TSMP算法

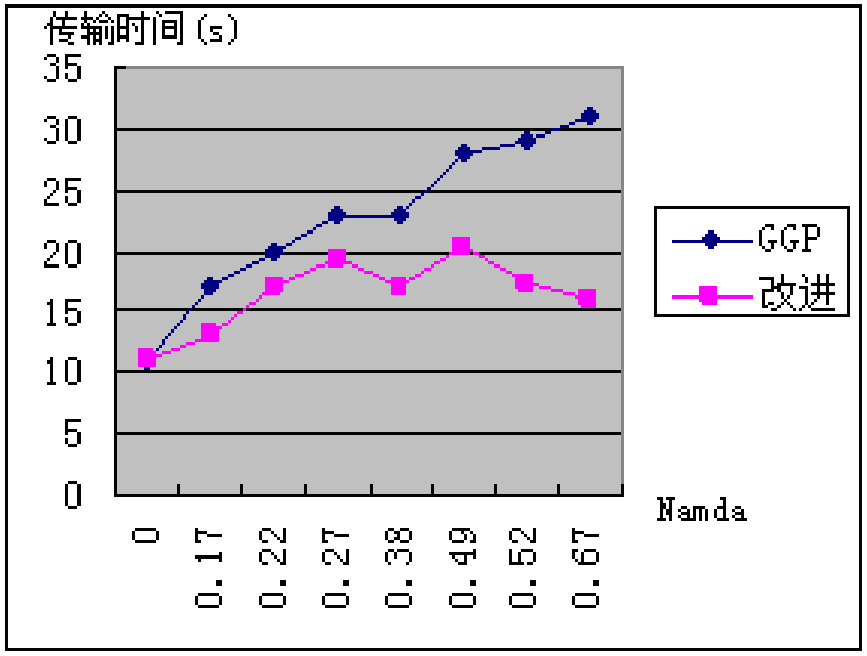
- TSMP算法设计思想
- Step1：对U集群进行内部调整，并且调整新生成的内部消息和外部消息的依赖关系。
- Step2：对V集群进行内部调整，并且调整新生成的内部消息和外部消息的依赖关系。
- Step3: 对调整完毕的二部图使用GGP算法划分成n块,记为block[n]。每一次从所有block中选出一个与内边相关度最小的一个block 。按照先后选出的顺序对Block[n]进行排序。
- Step4:为每个块中的边设置优先级，最先选择出块中边的优先级最低。最后选择出块中边的优先级最高。

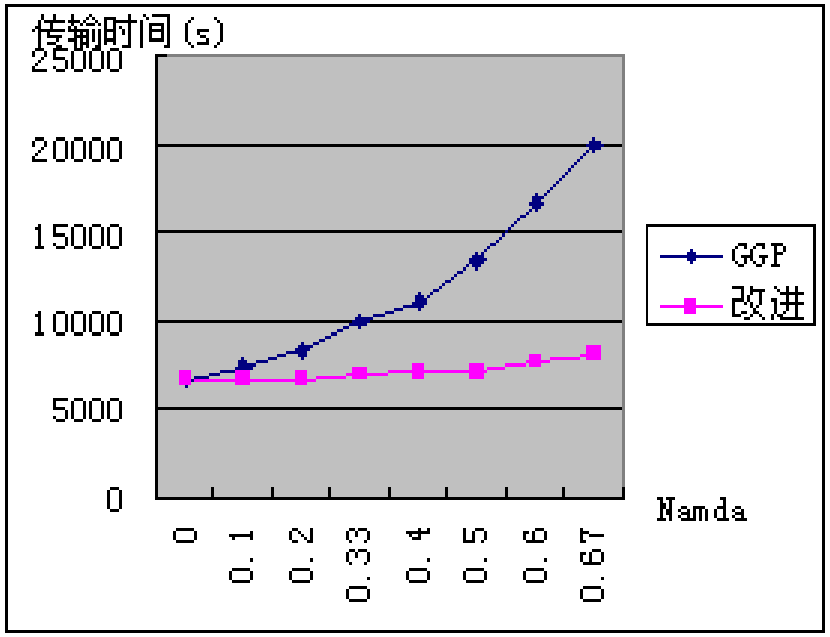
# 实验结果分析

- 实验条件：

| 符号 | 意义 |
| --- | --- |
| inner_bandwidth =1Gb/s | 节点连接到 Myrinet 网卡速率 |
| Agg_inner_band width=2Gb/s | 集群内部聚合频宽 |
| B'=10ms | 集群内部高速互联网络启动延迟 |
| B=10ms | 集群之间 Ethernet 网络启动延迟 |
| Backbone_band width=300Mb/s | 集群之间骨干网带宽 |
| inter_bandwidth =100Mb/s | 节点连接到 Ethernet 网的网卡速率 |
| K=3 | 集群之间可以同时发起的连接数目 k=Backbone_bandwidth/inter_bandwidth |
| K'=2 | 集群内部可以同时发起的连接数目 K'= Agg_inner_bandwidth/inner_bandwidth |
| n1=6 | 集群 1 中节点的数目 |
| n2=6 | 集群 2 中节点的数目 |

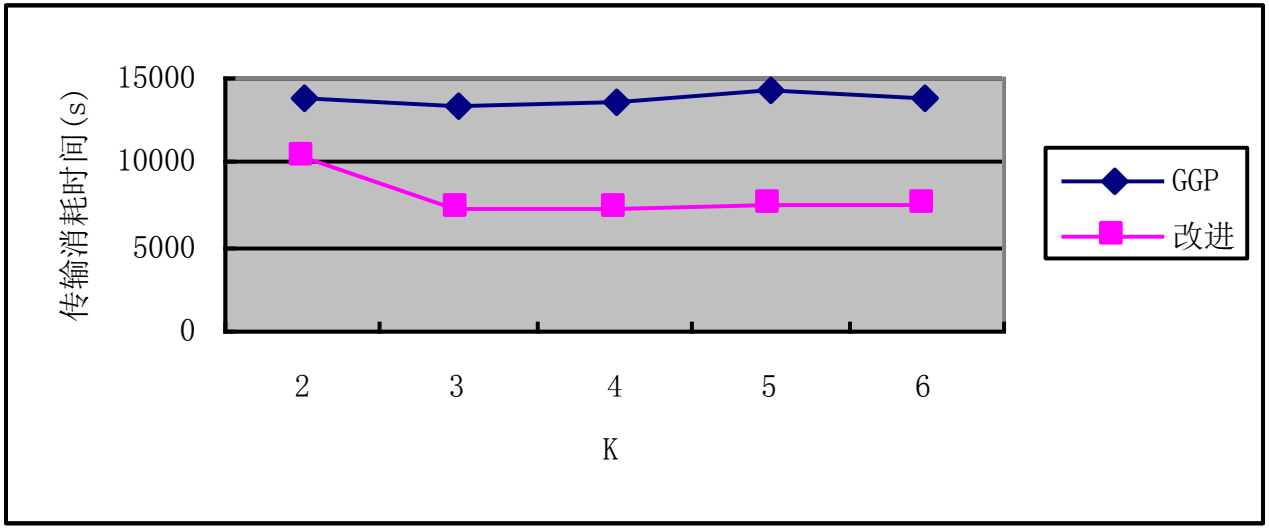
# 实验结果分析




对于短消息传输和长消息传输*GGP*和TSMP算法对比

# 实验结果分析




在k值不同情况下对于GGP和TSMP算法对比

# MPI消息传递过程

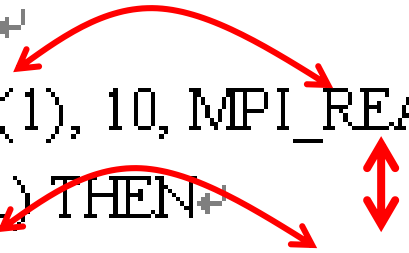
- 消息装配
- 消息传递
- 消息拆卸

# MPI的类型匹配

- 发送缓冲区中变量的类型必须和相应的发送操作指定的类型相匹配
- 发送操作指定的类型必须和相应的接收操作指定的类型相互匹配
- 接收缓冲区中变量的类型必须和接收操作指定的类型相匹配

# 例子

```
...
REAL a(20),b(20)
...
CALL MPI_COMM_RANK(comm, rank, ierr)
IF(rank.EQ.0) THEN
CALL MPI_SEND(a(1), 10, MPI_REAL, 1, tag, comm, ierr)
ELSE IF (rank.EQ. 1) THEN
CALL MPI_RECV(b(1), 15, MPI_REAL, 0, tag, comm, status, ierr)
END IF
```



# 类型匹配的内容

- 宿主语言的类型和通信操作所指定的类型相匹配
- 发送方和接收方的类型相匹配

# 类型匹配归纳

- 有类型数据的通信，发送方和接收方均使用相同的数据类型。
- 无类型数据的通信，发送方和接收方均以MPI_BYTE作为数据类型。
- 打包数据的通信, 发送方和接收方均使用MPI_PACKED。

# 不连续数据的发送（打包/解包）

- 基本操作
  - 发送方打包
  - 传递
  - 接收方解包

# 打包操作

- MPI_PACK(inbuf, incount, datatype, outbuf, outcount, position, comm )
  - 把<inbuf, incount, datatype>指定的数据进行打包
  - 放到从outbuf开始共outcount大小的缓冲期中
  - 在outbuf缓冲期中开始放置的起始偏移地址是position，放置完后下一次打包的起始偏移地址同时由position指定
  - Position在操作前后数值会发生变化

# 例子

```
int position , i,j,a[2];
char buff[1000];
...
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank ==0) { /* 进程0发送消息*/
    position =0;/*打包的起始位置*/
    MPI_Pack(&i,1,MPI_INT,buff,1000,&position,
MPI_COMM_WORLD);
                    /*将整数i打包*/
    MPI_Pack(&j,1,MPI_INT,buff,1000,&position,
MPI_COMM_WORLD);
                    /*将整数j打包*/
    MPI_Send(buff,position,
MPI_PACKED,1,0,MPI_COMM_WORLD); }
                    /*将打包后的数据发送出去*/
```

# 解包操作

- MPI_UNPACK(inbuf, insize, position, outbuf, outcount, datatype, comm )
  - 接受的消息内容< outbuf, outcount, datatype>
  - 打包数据存放的缓冲区起始地址是inbuf, 该缓冲区的大小是insize
  - 开始解包的位置是position，解包后下一次解包的起始位置是position

# 例子

```
int position , i;
float a[1000];
char buff[1000];
MPI_Status status;
....
MPI_Comm_rank(MPI_Comm_world,&myrank);
if (myrank ==0) {
    /* 进程0发送数据 */
    int len[2];
    MPI_Aint disp[2];
    MPI_Datatype type[2], newtype;
    i=100；
    /* 设置新类型中包含数据的个数 */
    len[0]=1;
    len[1]=i;
    MPI_Address( &i,disp);/*i相对于MPI_BOTTOM的偏移*/
    MPI_Address( a,disp+1); /*a相对于MPI_BOTTOM的偏移*/
    type[0]=MPI_INT;/*设置第一个类型是整型*/
    type[1]=MPI_FLOAT; /*设置第二个类型是浮点型*/
    MPI_Type_struct(2,len,disp,type,&newtype);/*定义新的数据类型，它包括一个整型与i浮点型*/
    MPI_Type_commit(&newtype);/*新类型递交*/
    /*数据打包*/
    position =0;/*打包的开始位置*/
    MPI_Pack(MPI_BOTTOM, 1,newtype, buff,
    1000,&position,MPI_COMM_WORLD);/*将i和数组a打包到buff*/
    /* 将打包数据发送出去*/
    MPI_Send(buff,postion, MPI_PACKED,1,0, MPI_COMM_WORLD)
}
```

# 例子

- else if(myrank ==1) {
- MPI_Recv(buff, 1000,MPI_PACKED,0,0,&status); /* 接收打包数据 */
- position =0;
- MPI_Unpack(buff,1000,&position,&i,1,MPI_INT,MPI_COMM_WORLD);
- /* 先将打包的浮点数个数解包*/
- MPI_Unpack(buff,1000,&position,a,i,MPI_FLOAT,MPI_COMM_WORLD);
- /*再将浮点数解包*/
- }

# MPI的进程拓扑

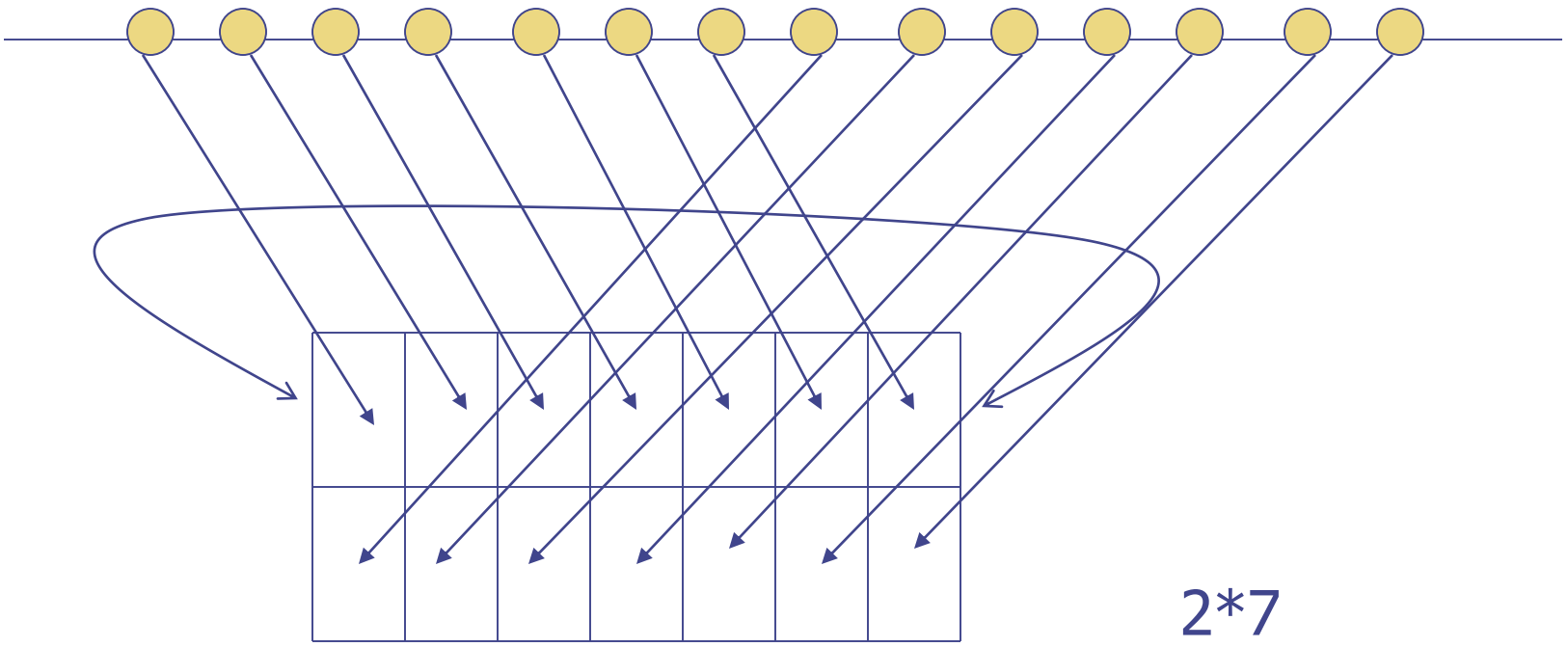
# 分类

- 图拓扑
- 笛卡儿拓扑

# 作用

- 高级并行程序设计
- 与底层物理拓扑结构的匹配，提高性能

# 创建笛卡儿拓扑



2*7

# 调用

MPI_CART_CREATE(comm_old, ndims, dims, periods, reorder, comm_cart)

得到一个新的通信域，该通信域附有进程拓扑信息

MPI_COMM_WORLD,2,{4,5},{false,false},false,newcomm

# 平移操作

- MPI_CART_SHIFT(comm, direction, disp, rank_source, rank_dest)

# 得到每一维的大小

MPI_DIMS_CREATE(nnodes, ndims,dims)

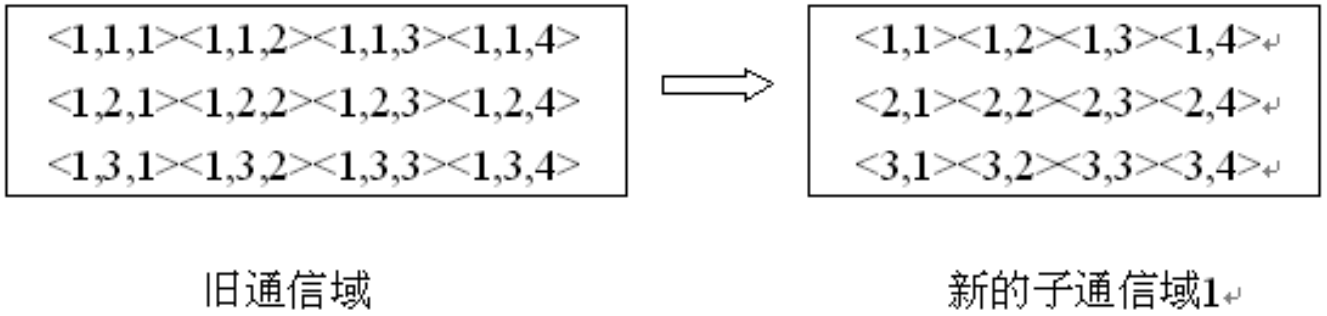
（20，2，{4,5}）

# 得到卡氏坐标

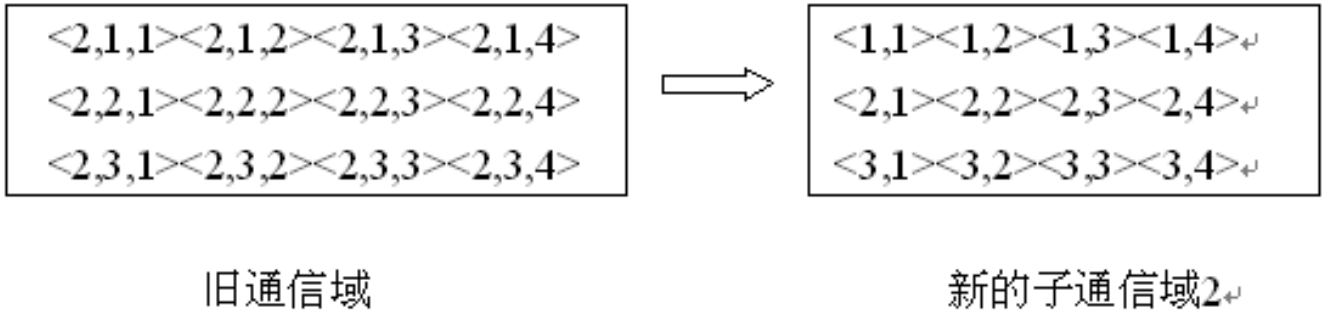
MPI_CART_COORDS(comm, rank, maxdims, coords)

# 子划分

MPI_CART_SUB(comm, remain_dims, newcomm)




2*3*4,<false,true,true>



# 例子1

```
#include "mpi.h"
#include <stdio.h>
#define NUM_DIMS 2
int main( int argc, char **argv )
{
    int             rank, size, i;
    int             dims[NUM_DIMS];
    int             periods[NUM_DIMS];
    int             new_coords[NUM_DIMS];
    int             new_new_coords[NUM_DIMS];
    int             reorder = 1;
    MPI_Comm        comm_cart;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    /* Clear dims array and get dims for topology */
    for(i=0;i<NUM_DIMS;i++) { dims[i] = 0; periods[i] = 0; }
    MPI_Dims_create ( size, NUM_DIMS, dims );
    /* Make a new communicator with a topology */
    MPI_Cart_create ( MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_cart );
    /* Does the mapping from rank to coords work */

    MPI_Comm_free( &comm_cart );
    MPI_Finalize();

    return 0;
}
```

# 例子2

```
#include "mpi.h"
#include <stdio.h>
#include "test.h"
#define NUM_DIMS 2
int main( int argc, char **argv )
{
    int             rank, size, i;
    int             errors=0;
    int             dims[NUM_DIMS];
    int             periods[NUM_DIMS];
    int             coords[NUM_DIMS];
    int             new_coords[NUM_DIMS];
    int             reorder = 1;
    MPI_Comm        comm_temp, comm_cart, new_comm;
    int             topo_status;
    int             ndims;
    int             new_rank;
    int             remain_dims[NUM_DIMS];
    int             newnewrank;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    /* Clear dims array and get dims for topology */
   for(i=0;i<NUM_DIMS;i++) { dims[i] = 0; periods[i] = 0; }
   MPI_Dims_create ( size, NUM_DIMS, dims );
```

# 例子2（续）

```
/* Make a new communicator with a topology */
    MPI_Cart_create ( MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_temp );
    for (i=0;i<NUM_DIMS;i++) {
      int source, dest;
      MPI_Cart_shift(comm_temp, i, 1, &source, &dest);
#ifdef VERBOSE
      printf ("[%d] Shifting %d in the %d dimension\n",rank,1,i);
      printf ("[%d]    source = %d  dest = %d\n",rank,source,dest);
#endif
    }
```

# 练习

- 请编写具有二维拓扑的MPI程序，每个程序都与其上下左右进程进行通信，请打印出它的上下左右进程的进程号和通信的内容。

- MPI_Dims_create ( size, NUM_DIMS, dims );
- MPI_Cart_create ( MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_temp );
- MPI_Cart_shift(comm_temp, i, 1, &source, &dest);