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 (一定内勾画)

phone: 62782530

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

考核方式

课堂测验和练习 50%

综合课程设计论文 50%

} 主要形式 (成绩≥80%)

论文 (Tsinghua, 清华的同学可以选这个)

用MPI/OMP/CUDA/OpenCL 等方法完成有实际应用的实际项目 } 特殊形式

协商 (前三周确定)

参考资料

MPI - the Complete Reference

Using MPI: portable parallel programming with the MPI

Using MPI-2: advanced features of the message-passing interface

高性能计算并行编程技术 - MPI并行编程技术 (都志辉)

MPI-forum.org ↳ { 语言实现性
并行编程例题

OpenMP: http://openmp.org

CUDA / OpenCL: https://developer.nvidia.com/cuda-zone
http://khronos.org/opencl

超级计算机排名
top500.org (http)
{ top500.org/green500 (https)
graph500.org (https)

网络编程库 (数学) www.netlib.org (函数、数值的库; 都是干活而非库的。若是写此自己写的)

课程目标:

- 知道: 并行计算问题的理念 (最重要的理论指导) 现实生活中问题: 有串行的事情, 也有并行的事情 而此课的编程训练即是"串行的"
- 掌握: 若干程序设计的基本方法、技巧
- 具备: 独立评估大规模科学与工程计算问题的能力
- 培养: 学科交叉的复合型人才

理想大幅度地提高问题的处理速度

支撑许多应用领域的发展 (物理、化学、生物、…)

domain-specific optimization (2019国家课程)

面向领域分别优化

课程基本内容

MPI (8学时) → 应用最广

其他技术 (2学时 - CUDA/GPU. HPF. OpenMP. VIA …)

并行程序的设计与实现 包括 (cluster可移动到单机, 和单机 (此章节最引)
移动上述 移动上述

用适当性能的计算系统 合适软件: 算法模式 (绝对绝对地位)
多核/众核处理、加速器协处理器。

用的多数算法相当适性的计算问题. 也是并行程序的实际平台

Lec 1 2019年9月11日
GB-208 THU

世界高性能计算机的发展

1986. Hans. Meuer 设计绝对计算机性能

Top500 ↓

1993年. 开始发布世界 Top500 (也就是从数据要在统计)

现在. 计算机系统排序 LINPACK BENCHMARK 性能排序
说明: Gauss 消元法解线性方程

Peak Performance
理论上浮点运算性能
(第六家族是理想上限)

目前Top排行: Summit > Sunway Taihu Light > Tianhe 2 > Titan > Sequoia > K Computer
(CPU+GPU) 日本
> Tianhe 1A > Jaguar > Roadrunner > Bluegene > The earth Simulator
> ASCI white … > CM-5
(1993年位列第一)

厂商 HPE(CHP+SGI)+Others 30% 惠普 Intel AMD NVIDIA 浪潮 曙光 联想 中国

10%

应用领域 Others 37.6% 多种应用都有 工业 38.4% 研究 20.4% 学术 11.8%

Asia 51.8% America 25.4% Europe 19.6%

Gigabit Ethernet 54.4% Infiniband 24.6% Custom 10.4% (定制的)

Cluster 90.6% MPP 9.4%

China 220 USA 116 日本 28

Linux 48.6% 其他 (多种UNIX、LINUX) 51.4%

Moore定律: 每隔18个月, 单位面积芯片可容纳晶体管的个数翻一倍

现在遇到瓶颈 (散热是大问题). 多核/众核是解决方案. 由此产生并行技术

如何充分发挥这台第一的威力

并行的算法和并行程序

软件与工具

人才

2016, 2017 Gordon Bell Award (面向应用的)

Atmosphere (Sunway) Earthquake (Sunway)

实现高性能并行计算的几个层次

从顶级到高级：模型、算法、程序

流水线 pipeline：一个任务多个步骤，由不同的硬件部分完成；时间消耗 $T+(M-1)\frac{T}{N_A}$（N_A 流水线工人数）

指令（CPU 内部）级并行

线程（共享内存）级并行 → OpenMP

进程（并行系统内部，分布式内存）级并行 → MPI → 大机器架构

用户级（并行系统外部）

从低级到高级

↳ 之前很火的网格计算：特定领域用网格连接各连接

也是 Top500 测试的基础

最重要最常见的并行计算 model

消息传递（更难一点） MPI：是消息传递的编程标准 model，所有架构都支持

数据并行（更简单一点）

有的语言，不可改数据并行

集群计算系统组成

$$集群 = \frac{计算结点 + 网络（硬件）}{软件}$$

节点：PC/workstation / PC/workstation

PC/workstation / MiniComputer … → 互联网络（e.g. Infini Band） → 并行系统与工具（Linux, MPI）

软件

集群计算系统的特点：性能很重要

都是可批量生产的通用部件

相对可以实现，容易实现（比起结点及网络的 Hacking 都是吗）

性能价格比的指标大

体现了多面治之的特点和技术

并行

我国高性能计算面临的问题：

缺件基础薄弱（算法/用户的参与生产数力）

软件人才缺乏（缺计算机操作用，应用，model，算法，设计，开发等都要结合）

企业需求不大（Made in China / Designed in China）

国家投入不足

高端制造业才需要设计，计算需求

我们的工作

集群式系统的建造（Deep Super-21C）

163/500，2003年，1.5 Tflops 相当于国际前一百位置

256 核 Intel P4

Cluster 硬件

Cluster Software System

Cluster Algorithm

THVIA 通信软硬件开销来源（软件开销占大）

计算与通信 overlap 开销，做用户层通信

网格计算 Grid Computation → 借用了 Power Grid 电力网的意义

2000年最大的技术（表决今天的 AI）

很多起来中心，联合起来，协同解决更大的问题

不同的算法，算法方式完全不同，编程策略 → 太理想化

将一个公司的计算资源集成 → 可行，为今天的"云计算"

总结：高性能计算是

先进基础研究，先进科学与工程活动所需要的计算

国家安全，科学发现，产业引领

连接科研院所的算法与软件（软件：实验室，计算）Bridge 和科学发现的新模式

课堂练习：给出一个算法并估计问题的 size

并行性在哪或是最大的？

并行计算的一些概念：

加速比（speed-up）：系统 T_1 > 方法 T_2 时，Speed up = T_1/T_2

Amdahl's law：若有一问题，其 size 我们可以为 1 = α（可并行）+ $(1-\alpha)$（串行）

那么理想的 speed up = $\frac{1}{1-\alpha+\frac{\alpha}{n}}$（n 处理器数目）

- 当 $n\to\infty$，则 Speedup = $\frac{1}{1-\alpha}$

比如 $\alpha=0.5$，即 Speedup = 2

即，若可并行部分占比太大，若结果与为单位于串行 size 1

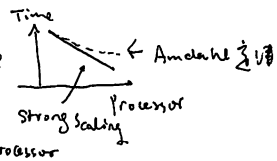
也即：问题的"模型"往往是求解问题最重要的一步

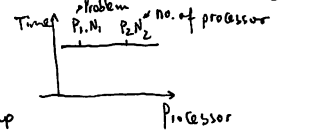
Amdahl 定律的问题是，问题 size 是固定的

Strong scaling：给定问题的大小，加机时间为此现象，数目呈线性下降

Weak scaling：问题的大小要大，之所以加机器数目也可等量大

求解时间基本保持不变





也即：只要按 Amdahl 定律作为 Speedup 的上限，但当问题很大，可以求解更大的问题而保持时间不变

基于MPI 编程技术

Lec 2　2019年9月18日
6B-208 THU

本节主要内容：
- 基本概念
- 基本通信语句
- 一些例子

如何判断是否可以并行执行？

- 自动并行化（软件）→ 编译器，硬件（CPU）
- 手动并行化改造 → 并行程序

现在多数 CPU 都有

自动并行的主流竞争领域是并行化

以下特定的问题有效

并行面临的挑战

自动并行的瓶颈（性能，范围都很窄）
单一处理单元性能的提高面临（physical limit）
多核发展趋势

并行程序和串行程序最大的不同

- 并行程序不一定在并行机上执行（可以模拟并行，只是效果不好而已）
- 如何并行的算法是否容易应用前面的积累

（embarrassingly parallel）

简单的，不需要交互与同步的并行问题

什么是MPI（Message passing Interface）

是消息传递编程模型

是消息传递编程的标准

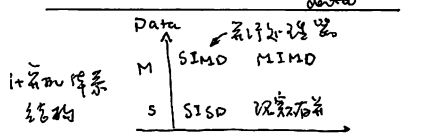
是消息传递编程的库

不同语境下，含义不同

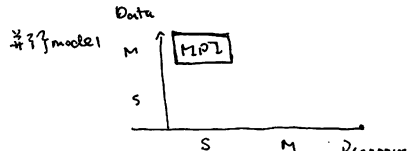
"MPI"的

MPI 编写的是 SPMD 程序

single Program Multiple data



计算机体系结构



一个源程序，可有多个进程，运行时体现出不同的行为

"MPI的最高境界：高度抽象"

一来说，普多事情

MPI 消息

通过通信的方式，基本方法

自动分配处理

分为
- 消息信封 envelop：已指定源/目的地，标识，通信域（Communicator）三元组
  - Source
  - destination
  - tag
- 消息内容：data

消息传递的范围，场景/环境；MPI_COMM_WORLD 是预定义的通信域

每个在一个 Comm 中指定的唯一标号

每个可在多个 Comm 中

已指定<源/目的地，标识，通信域>三元组

区别同一进程的不同消息

有时，Tag 是隐含的（组通信，内通信语句决定消息匹配）

已指定<起始地址，数据个数，数据类型>三元组

- data 在组通信的不同的形式
- 使用三元组，可以给我的系统带来（兼容性与可移植）

  另外，可以使编程更简单，更容易（更抽象的数据）

  此外，可以发送不连续的数据类型（增加知识）

MPI 的数据类型

是一个或多个相同的基本数据类型，以连续或不连续的方式形成的新类型

MPI 是强数据类型的
- A=B 当且仅当 A，B 同类型时才允许
- 在 MPI 中，发送与接收的类型必须匹配

MPI 预定义的基本数据类型

MPI_INTEGER	↔	int
REAL		float
⋮		
CHAR		char
INT		int
⋮		
BYTE		
PACKED		

三种通信方式

点到点通信（Point-to-Point）

有且只有两方（发送方与接收方）参与

双方协作共同完成一次消息传递和通信语句

编写大型程序，经常犯的错误

S TAG ↔ R TAG　Recv 语句 ↔ Send 语句

组通信（Collective Communication）

组内所有进程都参与才能完成（语法，先序完成）

各进程的调用形式都相同

有例外的进程，则是致命错误（有root调用，有root不调用）

广播与 Recv 可以调用参数不同

都是错误

单边通信（One-side Communication / Remote memory Access）

→ 90年代的 PVM，现在流行的 virtual machine 引入了单边通信

类通信单方完成

放 PUT

拿 GET　（可否用类似通信代替？）

MPI中通信

- 是通信的对象
- 是MPI通信若干单位
- 是组内具有唯一身份的对象

e.g. THLIB 有许多基本语句 BEGIN, END, SEND, RECEIVE

与MPI(6个)不同

MPI的最小集合

基本的通信语句

基本发送 MPI-SEND (buf, count, datatype, dest, tag, comm);

data: buf, count, datatype; envelop: dest, tag, comm

基本接收 MPI-RECV (buf, count, datatype, source, tag, comm, status);

接收缓冲区Count通常不小于发送Count; 源地址(dest-src), tag通配

data: buf, count, datatype; envelop: source, tag, comm

通信接收状态

注意:

对于接收语句, source 可以是 MPI_ANY_SOURCE (通配符)

tag 可以是 MPI_ANY_TAG (通配符)

FORTRAN 和 C 的调用语法不同

- 大小写无关
- 有关

总是比C多一个最后参数 error, 而C语言通信函数的值返回

获取个体身份

MPI-COMM-RANK (comm, rank);

MPI是一个SPMD模型

通过rank来获取自己的身份, 从而不同individual执行不同任务, 或处理不同数据

- 消息具有唯一性与不唯一性
  - 同一通信在一个comm中
  - 同一通信, 在不同comm中

已与其他通信的关系 (e.g. recv from rank-1, send to rank+1)

获取组内有多少人

MPI-COMM-SIZE (comm, size);

- 可以使程序更加通用, 当任务规模不同时, 通信结构为每一个任务取也不同 N / size

初始化与结束语句

MPI-INIT( );
MPI-FINALIZE( );

→ MPI是一个C/FORTRAN库. 高性能计算环境中不可避免的

但是, MPI程序的第一条语句必须是 MPI_INIT, 结束必须是 MPI_FINALIZE. } 不可省略, 否则会出现莫名其妙的错误

e.g. 以语言方式提供的并行机制, 可自动实现环境设置)

若以串行语言扩展提供, 则介于两者之间

程序设计示例

① Hello world

```
#include "mpi.h"
MPI_INIT( );
... (e.g. print "helloworld")
MPI_FANALIZE( );
```

Comm_rank, Comm_size, Get_processor_name

```
include "mpif.h"                integer rc, ierr
call MPI_INIT(ierr)
call MPI_COMM_RANK, MPI_COMM_SIZE, MPI_GET_PROCESSOR_NAME
... write(*,10) myid, numprocess, processor_name
call MPI_FANALIZE(rc)           10 FORMAT('...')
```

注意: 以不同的通信数、在不同的环境执行

结果不同;

CS FORTRAN 的用词和我不同, 含义不同;

② A将消息发给B 和 简单收发

```
Hinclude "mpi.h"
Init( );
Comm_rank(-, &myid); id = myid;
    -size( );
if (myid==0) { MPI_SEND(&id, 1, MPI_INT, 1, 100, -); }
else if (myid==1) { MPI_RECV(&id, -, -, 0, 100, -); printf(~, id); }
Finalize( );
```

分析:

通信 rank	0	1	2 …
执行流	init myid = 0 id = 0 send(id) ⟶	init myid = 1 id = 1 recv(id), 得到 id = 0 printf(id), 得到 0 被 print 出来	init myid = 2 id = 2
	finalize	finalize	finalize

通常对 MPI 而言 "局部串行" 与 "全局并行" 的特点 ("必须有两个以上节点")

- 一个通信有, 程序在串行
- 看所有通信, 却是并行的

消息传递有两种编程模式 { 主从式 / 对等式 , SPMD 可以写出两种模式

主与序设计方法: 把所有并行任务

- 没有单个任务的事件
- 所有任务的统一表示 SPMD

MPI 并行程序设计原则

- 理解并应用一种并行计算 model：大问题→多个
- 扩大两个 communication 之间 work 的量，减少通信调用的等待
- 减小通信次数，基本是以计算换通信

作业：完成程序中，每个进程向 rank = myrank + k 的进程发送字符串 "hello world!"，并从 rank = myrank - k 的进程接收。这里 rank 是进程的编号，k 是给定的正整数。

收到消息后，将字符串打印出来。

常见错误：① `for (int i=0; i< rank_size ; ++i) { if (i = myrank) { send & receive; } }` 这是串行的思想

@接收可使用 MPI_ANY_SOURCE 没问题。

成可读写 source = (myrank - k + rank_size) % rank_size;

dest = (myrank + k) % rank_size;

指定消息的标识和流向（envelop 参数）

Lec 3　2019 年 9 月 25 日　6B208　THU

MPI 的安装与使用

本节内容：MPICH 简介；基于 Linux 的 MPICH（重点）→ 基于 Linux 上的程序都是"在 Linux 上运行"；基于 Windows 系统的 MPICH

MPICH　Argonne 国家实验室与 Mississippi 州立大学开发；使用最广泛，开放性最好，可移植性最好；与 MPI 标准同步发展；MPICH 是实现性的，也可以不是最佳的。　cmcap：MPI = 模型 + 特性 + 库实现

安装：google "http://www.mpich.org/downloads"（可以不使用了）

```
解压 $ tar zxvf mpich.tar.gz
配置与编译 $ cd mpich
$ ./configure --prefix=安装目录/mpich-install   ← 编译器不同(C, FORT)，环境不同
$ make
$ make install
```

可以先此设置。e.g. CC=icc. CXX=icpc F77=ifort FC=ifort --with-pm=hydra ← 选择前缀

将安装路径下 /bin 加入 PATH 环境变量 ← 很多系统可能默认有安装了 MPICH。注意这两个通常冲突，并好查找的优先级问题

```
doc
examples 测试实例
src 源程序
lib 库
man 参考手册
www 参考手册
```

→ 主要为测试是否安装成功

MPI 程序的编译

```
mpicc (C)/mpiCC (C++)/mpif77/mpif90 -o test test.[cc/c/f/f90]
                              编译器   输出文件   输入文件
```

MPI 程序的运行与信息控制

如何建立运行程序的分布的信息控制 → 例：

SPMD 程序 — 4 — 执行 exe

机器/node1 — 1 — 2 — 3 — 执行 exe

执行 exe → 执行 exe（请求）

不同的机器上运行程序的分布式环境

做法1.

/etc/hosts.equiv 文件建立信任机制

修改机器信任机器名或 IP 地址即可。

（equiv 含义是：能够不受约束）

e.g. tp1.cs.tsinghua.edu.cn　机器名
tp2.cs.tsinghua.edu.cn（常见别名方式）
192.168.166.211

做法2.

建立 .rhosts 文件建立信任机制

（以用户为单位建立信任机制）

修改机器名及用户名

e.g tp1.cs.tsinghua.edu.cn mpi

做法3（配合做法1、做法2）

ssh 省略输入密码

第一步，生成密匙对 $ssh-keygen -t rsa

（在 $HOME/.ssh/ 目录下，生成 id_rsa, id_rsa.pub）

私匙，不要外传　公钥，可以内容放在远程服务器的 $HOME/.ssh/authorized_keys

第二步 $ cd $HOME
cat .ssh/id_rsa.pub | ssh username@w.x.y.z
'cat >> $HOME/.ssh/authorized_keys'

文件准备：所有机器都必须有可执行程序

{ 同构 → 直接 $scp cpi tp1:/home/mpi/mpich/examples/basic/.
异构 → 分别编译 } 放在相同目录（即在机器名上登录）

若使用相对路径，则取决于 ssh tp1 登陆上什么路径

另外，所需要的数据文件也要被交互读写。

没有 Cluster，每个机器都有 local disk

为此还有用网络通过共享 disk，只要将数据放在共享 disk，即不用拷贝；每个进程的输出文件名不同（按 rank 命名即可）

和 PVM 差别是：PVM 可以用图形控制的

而 MPI 则不行，后台时运行

（MPI-2 是加入了一些机制，使其可以如此）

程序运行

```
$ mpiexec -n N executive-file        （单机运行，相同缺省配置文件）
$ mpiexec -f mf -nN executive-file
$ mpirun -machinefile mf -np N executive-file   （先编译）
```

} 使用配置文件，使用缺省路径

帮助命令 $ mpiman

配置文件如 tp1 ← 起一个进程在 tp1 机器上
tp2:4 ← 起四个进程在 tp2 机器上
tp3:2
...

注意：若 machine file 中指出的总 cpu 进程数少于 -n 指定 cpu 进程数，
则 cpu 循环分配重新遍历 machine file 中指定 cpu 和总与进程 cpu 比总计数 cpu 进程。

注意：我机器 Titan（曾经 cpu 超算霸主），用 aprun 而不是 mpirun

基于 openPBS cpu 执行

http://www.pbsworks.com/

开源 cpu 作业管理系统，制定作业策略；用户向队列 cpu 递交请求

在超算上不允许登录计算结点　　e.g. 作业队列中，每次循环，找可以满足要求（核、内存请求）cpu 作业机群。

qsub / qdel / qstat

分别用于递交、删除、查询递交任务

$ qsub 脚本；

脚本示例 #!/bin/bash

PBS -N 名字

-l nodes=4:ppn=16

-e 指定标准出错

-o 指定 output 的文件

-l walltime=99:00:00 ← 时间之后，运行结束；有助于调度，充分利用资源

-V

cd $PBS_O_WORKDIR

mpirun -np 64 -machinefile $PBS_NODEFILE $PBS_O_WORKDIR/bin.exe

一般不用管理，要哪个结点有空去哪个结点。

$PBS_NODEFILE 是一个宏，是动态变化的

e.g. 神威太湖之光 cpu 使用。

通过 easyconnect 软件，连接 VPN 实现第双户 shell。

然后，通过 ssh 登录管理另一个超算登录结点

递交任务命令 bsub -I -q q_sw_share -b -m 1 -N 16 -np 4 -J … -share_size …

q 可以到 share 有权限　　每个任务占 4 个进程，共 16*4 个进程

-host_stack 1024 -cgsp 64 ~/swvasp > log 2>&1 &

每个任务占 64 个小核

在 WINDOWS 下使用 MPICH

下载与用户 cpu 网站，解压，安装，注册

wmpiregister.exe （初次，为每台机器一注册）

执行 mpiexec -n 2 d:\helloworld.exe

-hosts 2 m1 m2 d:\hello world.exe　　（2 节点，机器名 m1, m2）

-configfile d:\config

-n 1 -host m1 D:\helloworld.exe

-n 1 -host m2 D:\helloworld.exe

也可以使用 MS MPI STUDIO / Developer

练习：连接、配置一种 MPI，运行一个 MPI 程序；写出在超算运行报告

机器 OS、环境、配置、编译、运行 所有出现 cpu 问题和解决方案。

MPI的数据类型

Lec4 2019年10月9日
GB208 THU

引入的原因：抽象
为编码
移植性考虑

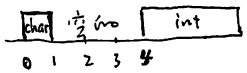
预先类型带来的：非常麻烦，可描述很抽象的数据

分类：预定义的数据类型（MPI_INTEGER, MPI_CHAR, MPI_SHORT, ...）
自定义的数据类型

又以此类型或已定义的数据类型为基础，最终是以基类型为单位来表示的
是把这个类型，已定义数据类型在基合自身结构的那时

对齐规则：若 size(T)=s，对于使 address % s = 0 的位置，称为对齐单元；

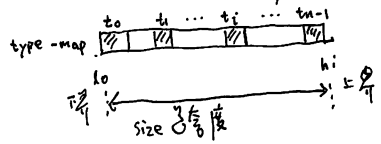
例如 size(int) = 4，那么 int 只能放在整4的单元.



MPI是预类型的：send/recv 的是基础数据类型声明而已；

通信的匹配是指：若两种 datatype 最终对应的基类型相同，则匹配.

类型图：类型图 = { <基类型.偏移>, <基类型.偏移>, ... }



1. 连续类型的复制：

MPI_TYPE_Contiguous(count, oldtype, newtype);

将 oldtype 重复多次，得到一个新的 newtype.

例如：有时常常连续发送知道的好，可定义先对做一份为一个的 datatype.

每次发送一份即可. MPI_TYPE_Contiguous(1000, MPI_FLOAT, &C_R);
MPI_SEND(&(a[0][0]), 1, C_R, right, tag, comm);

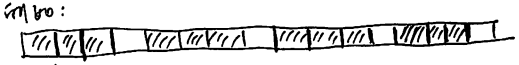
注意：连续重复多次，中间也可以有空洞（数据的size和extent不同时）

2. 向量类型的数据：

MPI_Type_vector(count, blocklength, stride, oldtype, newtype);

类型重复多块，块内有 blocklength 个 oldtype 连续排布，块之间有间隔 stride
count （单位为旧类型的长度）

例如：



Count = 4, blocklength = 3, stride = 4, oldtype = T

注意：也可以知道，vector 在 contiguous 基础上，

contiguous(c, 0, N) ⇔ vector(1, c, 0, 0, N)
stride 的值是跨度

（注意：对 extent 计算结构的影响）

例如：在FORI 中定义 matrix 的一行，向在C语言中定义

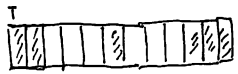
对应的一列.

MPI_Type_vector(1000, 1, 1000, MPI_FLOAT, &C_COL);
MPI_SEND(&(a[0][0]), 1, C_COL, up, tag, comm);

3. MPI_Type_indexed(count, array_of_blocklengths, array_of_displacements, oldtype, newtype);
每块长 每块的偏移

oldtype 可以是多块，每块可不等长度，块的位置的偏移
也可以是负例（以 oldtype 的长度为单位）

例如：



Count = 3, array_of_blocklength = {2,1,3}, array_of_displacements = {0, 5, 9}
oldtype = T

注意：indexed 比 vector 更一般

vector(C, L, S, 0, N); ⇔ indexed(C, {L,L,...}, {0, S, 2×S, 3×S, ..., (C-1)×S}, 0, N);
C

例如：定义一个三角形数据（含对角线）

```
double a[100][100];
int disp[100], blklen[100]; int i;
for(i=0; i<100; ++i) {
    disp[i] = 100*i + i;
    blklen[i] = 100 - i; }
MPI_Type_indexed(100, blklen, disp, MPI_DOUBLE, &upper);
MPI_SEND(a, 1, upper, dest, tag, comm);
```

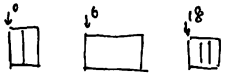
发送特殊的矩阵

4. 结构数据类型

MPI_Type_Struct (count, arr_of_blocklengths, arr_of_displacements, array_of_types, newtype);

将多个不同的基本类型组合，形成新的类型。

例如：



3, {2,1,3}, {0,6,18}

count, arr_of_blocklen, arr_of_displ ↗ 注意：以字节为单位。类型是 MPI_Aint,

注意：struct 是比 indexed 更一般的类型。

indexed (c, {L1, L2, ... Lc}, {D1, D2 ... Dc}, o, N);

struct (c, ------, {O1, O2, ... Oc}, N);

例如：struct part_t { char class; double d[6]; char b[7]; } 对应的类型

MPI_Datatype type[3] = { MPI_CHAR, MPI_DOUBLE, MPI_CHAR };

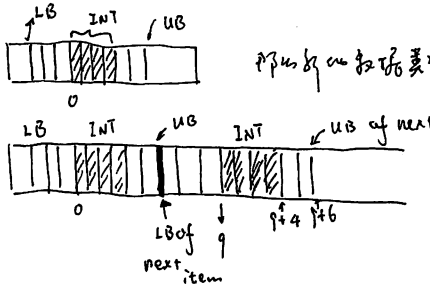
int blocklength[3] = {1, 6, 7};

MPI_Aint disp[3] = { 0, sizeof(double), 7 * sizeof(double) };

MPI_Type_struct (3, blocklength, disp, type, &newtype);

MPI_LB, MPI_UB 是特殊的类型，长度为 0，用于指定新类型的 lo, hi bound

MPI_TYPE_STRUCTURE (3, {1,1,1}, {-3,0,6}, {MPI_LB, MPI_INT, MPI_UB}, newtype);



用于新的数据类型的 extent 为从 UB-LB=9；注意 extend 用于计算连续的偏移量，与 size 无关

向新 newtype 传递时 Contiguous 为连续 byte

新的类型需要用 MPI_Type_commit (newtype) 和 MPI_Type_free (newtype) 进行声明。

MPI_Address (location, address); ↗ MPI_Aint

返回相对于 MPI_BOTTOM 的地址；

例如：&a[1], &a[10][10]

struct {int a; float b;} value;

MPI_Aint displ;

MPI_Address (&value.a, &displ[0]);

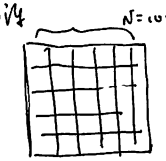
MPI_Address (&value.b, &displ[1]);

displ[1] -= displ[0];

displ[0] = 0;

也等于可以避免手动计算相对偏移量；

例子：通过 Datatype 来实现元素不连续的连续 (在集合通信中，利用 sendrecv 实现一维数组转置)



float a[100][100], b[100][100];

MPI_Type_vector (100, 1, 100, MPI_REAL, &column);

MPI_Type_extent (MPI_REAL, &sizereal);

MPI_Type_struct (2, {1, 1}, {0, sizereal}, {column, MPI_UB}, &column1);

MPI_Type_commit (&column1);

MPI_SENDRECV (a, 100, column1, myrank, 0, &status

b, 100×100, MPI_REAL, myrank, 0, comm);

将 send 和 recv 等在一起

- 有助于 mpi 进行优化
- 避免死锁

发送 100 列，接收 100 行。

由于是连续的，可以当成一个新类型。

若 Vector 参数，从设为 type_struct，可使用 hvector（按字节为单位指定偏移量）

MPI_Type_hvector (100, 1, sizereal, column, xpose);

MPI_Type_commit (&xpose);

MPI_Sendrecv (a, 1, xpose, myrank, 0, &status

b, 100×100, MPI_REAL, myrank, 0, comm);

MPI_BOTTOM 与结构类型变量一起使用

MPI_Address (&value.a, &displ[0]);

&value.b, &displ[1]

MPI_Type_Struct (2, ..., displ, {MPI_INT, MPI_FLOAT}, newtype);

MPI_SEND (MPI_BOTTOM, 1, newtype, ...);

作业：使用 MPI 的类型，发送一个元素为 (M×M) 的左上三分之一部分 (包括对角线)；

Lec 5
2019年10月16日
6B208 THU
郭达雄 教授

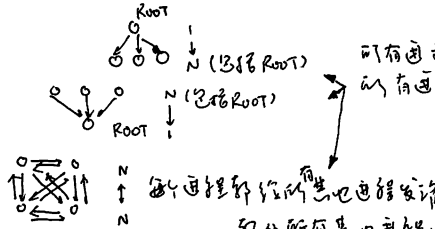
MPI 组通信 { 通信功能
　　　　　 { 通信同步

这会讲的通信一个是集合优化方法，很多场景适用

MPI-3 加入了非阻塞的组通信，这里暂略

Collective
组通信的通信功能

一对多
多对一
多对多



可有通信都加以限定为（为发生一错误），且通信不用同一调用(SPMD)
的有通信由ROOT协调时，各次通信无法实现
(ROOT不同，意味着不同的组通信)

每个通信都给所有其他通信发消息
都从所有其他通信收消息

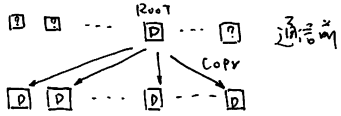
点-点通信对比，P-P必须有且仅有两个通信参与（除MPI_PROC_NULL作src/dest时）
↳ 空通信，通信将被忽略

- MPI_PROC_NULL 有利于编写方法. e.g. 在 Cartesian topology 时. 若无相邻时，直接通信给MPI_PROC_NULL作src/dest，则有通信由MPI调用将一般.

广播 MPI_BCAST(buffer, count, datatype, root, comm);

将一个通信的数据传送给其他所有通信 (envelop 没有 tag，都调用以序匹配)

示意：



e.g.
```
printf("n=%d\n", n);
if (myid==0)
        scanf("%d", &n);
MPI_Bcast(&n, 1, MPI_INT, 0, comm);   // 注意，Bcast的含义，Root通信执行"发"
printf("after bcast, n=%d\n", n);     //  其他通信执行"收"
```

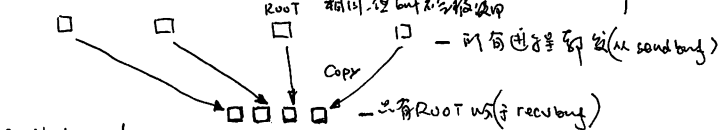
e.g.
```
test_array = (int*)malloc(sizeof(int)*nproc);
for (int i=0; i<nproc; ++i) {
        test_array[i] = rank+i; }
MPI_Bcast(test_array, nproc, MPI_INT, 0, MPI_COMM_WORLD);
```
// 通信0：0,1,2,3,…
// 通信1：1,2,3,4,…
…
// 所有通信：0,1,2,3,…

不同 Bcast 与 Recv 混用

收集 MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm);

注意：recvcount == sendcount，即所有通信都一个相同 datatype 的数据，且个数是一样的，且和接收通信的接收个数相同（字节读写接收）
ROOT 的 Recvbuf 要足够大；非 ROOT 的 recvcount 也要相同，但 buf 不会被使用

示意：

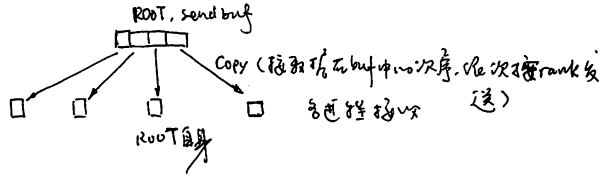


recvcount*N个);

散发 MPI_SCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm);
在接收区有序存放，按在 comm 中 rank 顺序

注意：当 GATHER 的逆通信，即 sendcount=recvcount (实际是 sendcount*N)

示意：



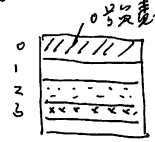
注意：与 BCAST 不同，每个通信收到的数据不同

组收集 MPI_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, Comm);

注意：等价于一次集+一个播
所有通信都得到 gather 的结果，所以没有 root 域.

e.g.
```
int table[MAxProcess][MAxProcess];
blocksize = MAxProcess/size;
sendcount = blocksize * MAxProcess;
recvcount = sendcount;
for(i=rank*blocksize; i<(rank+1)*blocksize; ++i)
    for (j=0; j<MAxProcess; ++j)
        table[i][j]=rank+10;          //  对自己负责的行涂色
MPI_Allgather(&table[rank*blocksize][0], sendcount, MPI_INT,
              &table[0][0], recvcount, MPI_INT, MPI_COMM_WORLD);
```

示意：gather后的结果



MAxProcess = 5
size = 4

// 所有通信获取其他通信的数据

问题：接收缓冲区与发送缓冲区有重叠，可以出错吗？

MPI_ALLGATHER(v)几种实现方法

环形交换算法(RING) ：交换N-1次

- 比较适应于两两之间通信，简单规则；但是通信次数比较多

递归倍增(Recursive doubling) ：适合在2的幂次的算法编写。

- 第0号为2^0=1次第一个进程依次交换数据
  第K号，第0号为2^{K-1}个进程交换数据；每一步，相当于两两为一块对应的数据；

- 局部性差，但通信次数少

Bruck算法

第I个进程，K等时，发送2^K个数据到 $\mathrm{mod}\left((I-\exp(2,K)),P\right)$ 进程

从 $\mathrm{mod}\left((I+\exp(2,K)),P\right)$ 收数据

邻居交换算法(Neighbor Exchange)算法 - 左右左右算法

01	23	45	67	第1次
70	12	34	56	第2次
01	23	45	67	第3次

局部性很好，并且每一步的都拿到更多一些数据。

适合多核与通信编一所表

混合算法

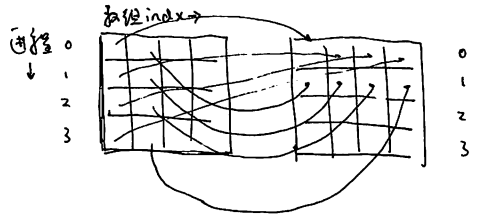
把数据进程数·数据量

全交换 MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, Comm);

每个进程都给不同进程发不同长度一些数据

MPI_Alltoallv(sendbuf, sendcounts, displs, sendtype, recvbuf, recvcounts, rdispls, recvtype, Comm);

相当于把矩阵转置；相当于N次scatter；相当于N次gather;



组通信的同步功能 MPI_Barrier(comm);

组内所有进程都执行了该调用后，才返回执行。

(调试程序，或者是为了一次的，使用)

e.g. 在程序中，变量a会不断的更新；每次更新时，所有进程都打印当前值；

```
printf("myid = %d, the value of a = %d\n", myid, a);
MPI_Barrier(comm);
a = ...;
printf("myid = %d, the value of a = %d\n", myid, a);
```

组通信的计算功能

归约 MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm);

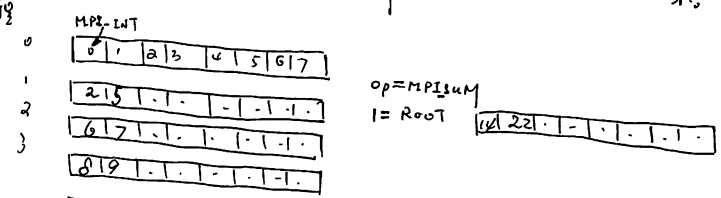
count是有多个个数(每个手里的参与计算)

以发一组缓冲区为依定全一样

同理还有All reduce.

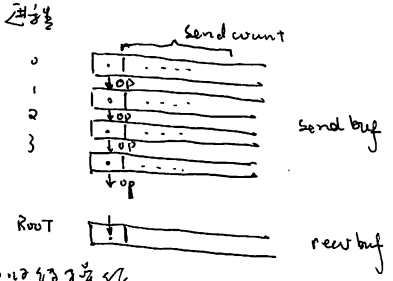
所有进程都得结果。

求和 for MPI_SUM



每个进程一等1个元素，都发给root进程。最后收到root由recv等1个元素。

求和：



MPI预定义一些好的操作

MPI_MAX
MIN
SUM
PROD
LAND BAND
LOR BOR
LXOR BXOR
MAXLOC
MINLOC

e.g. 求PI的值，利用积分 $\int_0^1 \frac{1}{1+x^2}dx = \frac{\pi}{4}$ ；于是 $f(x) = \frac{4}{1+x^2}$，计算 $\int_0^1 f(x)dx$

```
#include <mpi.h>
.... double f(double x) { return 4.0/(1.0+x*x); }

int done=0, n, myid, numprocs, i;
double PI25DT = 3.141592653589793238462643;
double mypi ...;

n=0
if (myid==0) { scanf ("%d", &n); }
MPI_BCAST (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h = 1.0/(double)n;

sum=0.0;
for ( i=myid+1; i<=n; i+=numprocs ){      // 采用Cyclic(循环)划分分布
        x = h*(double)i - 0.5;            // 第一个进程完成 1, numprocs+1, 2*numprocs+1 ...
        sum += f(x); }                    //  =          2, numprocs+2, ...
                                          // numprocs  numprocs, ...
mypi = h*sum
MPI_Reduce( &mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (myid==0){
        printf ("pi = %f\n",pi); }
```

大作业：2~3组一组，用 MPI/OpenMP/CUDA/OpenCL 实现一种感兴趣的并行算法或应用

提交：书面报告及运行说明；（如软件包）
详细的技术报告；

可以包括：
基于现有并行环境（数值中实现的含有并行性的优化）
典型算法并行化（新软件的关键性算法实现，e.g. PSO，遗传算法，相应串行算法）
典型并行优化算法的应用

e.g. 优化问题 $f(\text{in-coordinate}, \text{out-coordinate}) = y$（$x_0$，$x_1$；待优化参数和 x_i 的取值）
输入初始值，返回优化的结果；
各进程以随机初值调用 f，比较输出，选最小的那一个进行下一次调用；直到返回的结果好于给定的时候都退出；（各个进程的算法不同，所以相同输入，输出不同；通常应用一定：当多次取不同值时，会有最优的算法）

为了结果可靠性，仅以独立随机数，来模拟真实的优化算法。

Lec 6

2019年10月23日

GB208 THU

郭志辉 教授

上次课回顾：

实现一：使用 MPI_Gather() 将0进程作为root，得到所有其他进程的进程号 MIN_PROC。

然后root将MIN_PROC广播出去，使用BCAST();

然后将MIN_PROC中的 x 广播出去，使用BCAST(); → 也可用 MPI_Allgather() 直接广播

Y和x，进行通信都各自找最小的Y和对应的x。（通信量较大）

实现二：

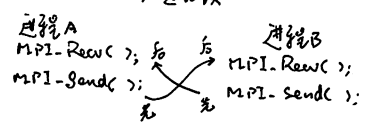
用MPI_Minloc得到最小的Y对应的进程号，使用MPI_Reduce();

使用该进程号广播最小的Y对应的 x。 或Allreduce();

安全的MPI程序

判断我可能死锁的MPI程序，不安全；否则是安全的

e.g. 同时先收后发，如果死锁



同时先发后收，可能死锁（取决于buffer是否够大）

```
A                    B
MPI_Send();          MPI_Send();
Recv();              Recv();
```

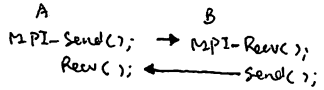
e.g. 真实案例：

用的SAN都是 InfiniBand 或 EtherNet，旧时代集群网络都是 MyNet.

供货商要求：先给钱，再发货

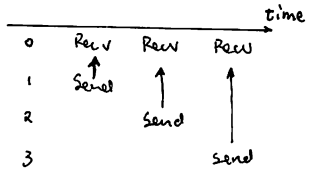
收件人要求：先收货，再给钱

e.g. 收发匹配，不会死锁（是安全的编程模式）



例如：大数据会等待，但是不会死锁

```
if(rank != 0) MPI_Send( dest = 0);
else{
    for(i=1; i<=n-1; ++i)
        MPI_Recv(); from process i
};
```



获取的方法是使用 MPI_Recv(tag=ANY_TAG, src=ANY_SOURCE);

解决死锁的一种方法：MPI_SENDRECV();

MPI自己优化，且绝不死锁

组调用语句可以写成的形式

```
for(){
    计算语句B;
    if(B) break;        // 组内所有进程都必须执行组调用
    组调用语句;}        若有多个进程写 Break 不一致，则会导致死锁
```

- 对顺序执行程序改来的并行程序，有这样一种情况，e.g.

```
for(){
    B = (ε < Δ);
    if(B) break;
    组调用;
    大量一起算;}
```

- 解决方案统一退出标识，将来收到为1的一个进程，然后广播退出指令

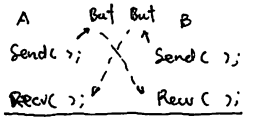
MPI的4种通信模式

标准通信模式 MPI_Send();

系统决定是否缓存数据

编程者可以认为：调用已经结束了，标准通信就已完成（不一定真的到了对方的进程）

e.g.



两个Send都将数据放入Buf，然后退出调用；这样不会死锁；

若Buf不够多，则也会死锁。

缓存通信模式 MPI_BSEND();

使用用户提供的缓冲区 MPI_Buffer_Attach(buff, size);

DEATTACH(buff, size);

那么，Bsend()将一定放入用户缓冲区，然后退出调用；（只要Buff够大，一定不会死锁）

使用缓很少.

同步通信模式 MPI_SSEND();

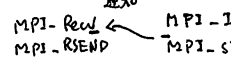
发送语句返回时，相应的接收语句一定已经接收

就绪模式 MPI_RSEND();

以极100%的把握，接收的buff已经准备好；

- 当种用法可以减少中间的通信次数，提高速度
- 但是使用起来有些危险，需由用户保证Recv已经Ready

常见的场景

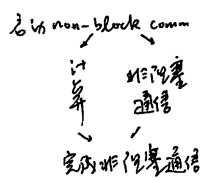


MPI的非阻塞通信模式：非阻塞发送与接收

调用一返回不意味着通信的实现（与之前的所有模式不同），只是给下一个发送/接收一指令

作用是：计算与通信重叠（阻塞模式中，必须等待通信完成才能计算）

图示：



需考虑：非阻塞~发送，接收 如何书写？如何知道通信是否完成？

非阻塞发送

```
MPI_Isend (buf, count, datatype, dest, tag, comm, request);
```

怎么写非阻塞发送

同步：已开始接收，才返回

缓存：用户提供buffer

就绪：接收已开始，才发送

非阻塞接收

```
MPI_Irecv (buf, count, datatype, src, tag, comm, request);
```

测试非阻塞通信是否完成

用request：非阻塞通信对象来描述，管理非阻塞通信

启动非阻塞通信时，创建request.

完成通信时，销毁request.

- MPI_Wait(request, status); 等待完成，释放对象 (e.g. MPI_Isend() + MPI_Wait() 等同于 MPI_Send() 阻塞通信)
  - 发送完成，buf可重用
  - 或接收完成，buf已经包括了msg
- MPI_Test(request, status, flag);
  - true，则等同于MPI_Wait()的结果
  - false，则相当于空操作

第i non-block comm 的完成与测试

相应阻塞通信 阻塞等待

第i non-block comm 的完成与测试

任何一个对象完成即返回

```
MPI_Waitany (count, requests, index, status);
```

完成的对象 void

```
MPI_Waitall (count, requests, statuses);
```

全都完成，才返

```
MPI_Waitsome (incount, requests, outcount, array_of_indices, array_of_status);
```

完成的对象 ids.

≥0个对象完成即返回

同样也有 Testany (count, requests, index, flag, status);

Testall (count, requests, flag, statuses);

全完成

Testsome (incount, requests, outcount, indices, statuses);

≥0个对象完成即返回

如何使得计算与通信重叠？尽量将通信向前写

e.g. 先发送已经有，但之后还会需要的数据

然后进行下次发送需要的数据的计算

检查上次发送用发送缓冲区

e.g. 先接收因为不用的data

进行其他的计算.

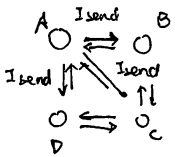
检查接收，进行为接收相关的计算.

例子：

```
A                          B
MPI_Isend(&rq);            MPI_Isend(&rq);
MPI_Recv();                MPI_Recv();      ---> 由于已经标记发送，所以Recv一定能完成
MPI_Wait(&rq);             MPI_Wait(&rq);
```

但是对前一个Isend不一定完成可以接收的，所以还要wait. 且wait必须要有.

例子：



进行相邻的其他所有进程发送信息.

从其他所有进程接收消息;

然后 wait 或 waitall.

非阻塞通信的重叠优化

发送方循环内，利用如通过 Isend/Irecv 创建 request 对象，然后 wait 清除 request 对象。

可否创建一个指向非阻塞通信对象，然后多次使用，最后释放一次。（可以减少通信代价）

重复非阻塞通信

1. 初始化（四种类型）通信连接的对象，通信并未开始执行。
2. 启动通信
3. 发送通信，等待数据到达
4. 显式释放对象

MPI_Startall(count, requests);
MPI_Start(request);

Test与Wait，与非阻塞通信一样

MPI_Send_Init (buf, count, datatype, dest, tag, comm, request);
ssend
ibsend
Rsend
或者Recv的四种

MPI_Request_free(&request);

举例：

```
MPI_Recv_Init(buf,100,MPI_INT,left,tag,comm,req2);
MPI_Send_Init(buf,100,MPI_INT,right,tag,comm,req1);
for(i=1;i<n;++i){
    MPI_Startall(2,array_of_requests);
    // 计算
    MPI_Waitall(2,array_of_requests);}
MPI_Request_free(req2); MPI_Request_free(req1);   // 将非阻塞通信对象释放
```

适合采用的形式

```
for(){
    MPI_Isend();
    MPI_IRecv();
    // 计算
    MPI_Waitall();}
```

消息到达的检查

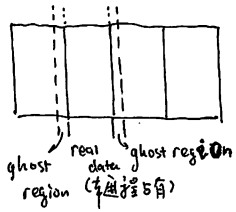
阻塞 MPI_Probe(src, tag, comm, Status); 等待匹配的消息到达

非阻塞 MPI_Iprobe(src, tag, comm, flag, status); 检测是否有消息到达

例如：

```
while(1){
    MPI_Iprobe(~ &flag ~);
    if(flag){
        MPI_Recv();
        // 计算}
    //计算总是要的 };
```

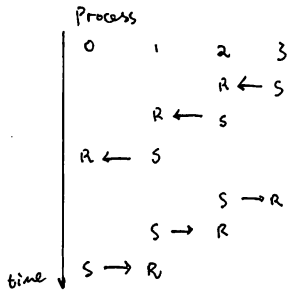
例3：Jacobi 迭代的数据通信模式



从每个进程来看，必须从相邻进程获取 ghost region，并将自己的边界信息传给相邻进程

然后完成一步迭代计算。

```
if(rank<3)   recv(rank+1);
if(rank>0)   send(rank-1);
if(rank<3)   send(rank+1);
if(rank>0)   recv(rank-1);
```



SRSR → 通信无论怎样改顺序，实际都是串行一个个传递

RSSR → 可以并行一部分

作业：创建一个大的数组，将计算并行到到到分；

每个进程都与上下左右四个进程交互数据，每次都仅发送一次数据量比较，最好一步比较。

做法：虽然可用 MPI_PROC_NULL，最好统一的格式；或用if判断；

可以用自定义的数据类型来做。