陈国良. 并行计算: 结构 • 算法 • 编程 [M]. 1999.

屈彬

2019 年 7 月 17 日

1 并行计算机系统及其结构模型

1.1 并行计算（2019 年 7 月 2 日）

1.1.1 并行计算与计算科学

定义 1.1 (应用需求) 应用需求分为三类，**计算密集**（*Compute-Intensive*）型应用，如大型科学工程计算与数值模拟；**数据密集**（*Data-Intensive*）型应用，如数字图书馆、数据仓库、数据挖掘和计算可视化等；**网络密集**（*Network-Intensive*）型应用，如协同工作、遥控和远程医疗诊断等。

定义 1.2 (高性能计算和通信) *High Performance Computing and Communication, HPCC*。

定义 1.3 (加速战略计算创新) *Accelerated Strategic Computing Initiative, ASCI*。

定义 1.4 (美国能源部三大高性能计算实验室) *Lawrence Livermore*、*Los Alamos*、*Sandia*。

1.2 并行计算系统互连（2019 年 7 月 3 日）

1.2.1 系统互连

定义 1.5 (机群网络分类) *1.* **系统域网络:**（*System Area Network, SAN*），连接短距离（*3 25m*）内的节点。

2. **局域网络：**（*Local Area Network, LAN*），连接企事业单位内（*500m 2km*）内的节点。

3. **节点内网络：**由处理器总线、局部（本地）总线、存储器总线构成。

4. **小型机系统接口：**（*Small Computer System Interface, SCSI*）由 ***I/O*** 总线、系统总线构成。

定义 1.6 (工作站机群) （*Cluster of Workstations, COW*），通过 *SAN/LAN* 互连的工作站组。

1.2.2 静态互连网络

定义 1.7 (对剖宽度) （*Bisection Width*）：将网络划分为两等分所必须剪去的最少边数。

1.2.3 动态互连网络

定义 1.8 (总线分类) 常用总线包括 *PCI*、*VME*、*Multibus*、*SBus*、*Microchannel* 和 *IEEE Futurebus*。

1. **本地总线：**（Local Bus），CPU 板级上的总线。
2. **存储器总线：**存储器板级上的总线，主要指内存与 CPU 互连的总线。
3. **数据总线：**I/O 与通信板级上的总线，例如硬盘与内存之间的总线，以及网卡。

1.2.4 标准互连网络分类

1. **光纤分布式数据接口**（Fiber Distributed Data Interface, FDDI），采用双向光纤令牌环提供 100 200Mb/s 的数据传输。
2. **快速以太网**主流的互连网络，第三代以太网数据传输速度可达 1Gb/s。
3. **myrinet**由 Myricom 公司生产的千兆位包开关网。
4. **高性能并行接口**（High Performance Parallel Interface, HiPPI），主要用于构筑异构计算机系统。
5. **异步传输模式**（Asynchronous Transfer Mode, ATM），在光纤通信基础上建立起来的一种新的宽带综合业务数字网（B-ISDN）的交换技术。
6. **无线带宽**（InfiniBand）。

1.3 并行计算机体系结构（2019 年 7 月 4 日）

1.3.1 并行计算机结构模型

大型并行机系统一般可分为六类机器：

1. **单指令多数据流：**（Single Instruction Multiple-Data, SIMD）。
2. **并行向量处理机：**（Parallel Vector Processor, PVP）。
3. **对称多处理机：**（Symmetric Multiprocessor, SMP）。
4. **大规模并行处理机：**（Massively Parallel Processor, MPP）。
5. **工作站机群：**（Cluster of Workstations, COW）。
6. **分布共享存储多处理机：**（Distributed Shared Memory, DSM）。

除 SIMD 外，其余五种皆为 MIMD。

1.3.2 并行计算机访存模型

1. **均匀存储访问：**（Uniform Memory Access, UMA），其特点包括：物理存储器被所有处理器均匀共享；所有处理器访问任何存储单元取相同的时间；每台处理器可带私有 cache；外围设备也可以一定形式共享。这种系统由于高度共享资源又被称为**紧耦合系统**（Tightly Coupled System）。多核同构 CPU 系统的存储模型属于典型的 UMA 模型。
2. **非均匀存储访问：**（Nonuniform Memory Access, NUMA）。
3. **全高速缓存存储访问：**（Cache-Only Memory Access, COMA）。
4. **高速缓存一致性非均匀存储访问：**（Coherent-Cache Nonuniform Memory Access, CC-NUMA）。
5. **非远程存储访问：**（No-Remote Memory Access, NORMA）。

1.3.3 并行计算机存储组织

定义 1.9 (层次存储技术) *利用程序局部性，设置多级存储系统。底层的存储器通常具有容量大、速度慢、成本低的特点，高层的存储器通常具有容量小、速度快、成本高的特点。层次存储技术出现的目的在于寻找性能与成本之间的平衡。*

定义 1.10 (高速缓存一致性问题) 当处理器将新数据写入高层存储器时，需保证其他层次中存储器的数据对应的位置具有相同的副本。

2 当代并行计算机系统介绍

跳过第 2 章，它的内容与第 1 章存在大量重复。

3 并行计算性能评测

3.1 并行计算机的一些基本性能指标（2019 年 7 月 5 日）

3.1.1 CPU 和存储器的某些基本性能指标

名称	符号	含意	单位
机器规模	n	处理器的数目	
时钟速率	f	时钟周期长度的倒数	MHz
工作负载	W	计算操作的数目	Mflop
顺序执行时间	T_1	程序在单处理机上的运行时间	s（秒）
并行执行时间	T_n	程序在并行机上的运行时间	s（秒）
速度	$R_n = W/T_n$	每秒百万次浮点运算	Mflops
加速	$S_n = T_1/T_n$	衡量并行机有多快	
效率	$E_n = S_n/n$	衡量处理器的利用率	
峰值速度	$R_{peak} = nR'_{peak}$	所有处理器峰值速度之积	Mflops
顺序峰值速度	R'_{peak}	单个处理器的峰值速度	Mflops
利用率	$U = R_n/R_{peak}$	可达速度与峰值速度之比	
通信延迟	t_0	传送 0-字节或单字的时间	μs
渐进带宽	$r_{\inf}$	传送长消息的通信速率	MD/s

3.1.2 通信开销测量方法

1. **乒 -乓方法：**（Ping-Pong Scheme）适用于一对节点之间通信开销的测量。
2. **热土豆法：**（Hot-Potato），也称作**救火队法**（Fire-Brigade），适用于测量多个节点（两个以上）以环形方式通信，数据包传递一圈的通信开销。

理论**点到点通信开销**$t(m)$ 是消息长度 m（字节）的线性函数：

$$t(m) = t_0 + m/r_{\inf}$$

其中 t_0 是通信启动时间（μs），r_{inf} 是渐进带宽（MB/s）。

3.1.3 机器的成本、价格与性能/价格比

内容比较零碎，跳过。

3.2 加速比性能定律（2019 年 7 月 6 日）

3.2.1 Amdahl 定律

3.2.2 Gustafson 定律

3.2.3 Sun 和 Ni 定律

4 并行算法的设计基础

出于 PAC 比赛的需求，暂时先跳过第三章。

4.1 并行算法的基础知识

4.2 并行计算模型（2019 年 7 月 7 日）

并行计算模型是硬件和软件之间的一种桥梁（大雾，什么叫作 “桥梁”），使用它能够设计分析算法，在其上高级语言能被有效地编译且能够用硬件实现。（我觉得作者没有把并行计算模型的用途讲清楚）

4.2.1 PRAM 模型

即**并行随机存储机器**（Parallel Random Access Machine, PRAM），是一种抽象的并行计算同步模型。在这种模型中，假定存在着一个容量无限大的共享存储器；有多个同构处理器；在任何时刻各处理器均可通过共享存储单元相互交换数据以实现同步。根据对处理器读写时序的限制，又可分类为：

1. **不允许同时读和同时写**（Exclusive=Read and Exclusive-Write）的 PRAM 模型，记作 **PRAM-EREW**。
2. **允许同时读不允许同时写**（Concurrent-Read and Exclusive-Write）的 PRAM 模型，记作 **PRAM-CREW**。
3. **允许同时读和同时写**（Concurrent-Read and Concurrent-Write）的 PRAM 模型，记作 **PRAM-CRCW**。

不过，允许同时写显然是不现实的，所以对 PRAM-CRCW 模型做了进一步的约定：

1. **CPRAM-CRCW**：公共的 PRAM-CRCW，只允许所有的处理器同时写相同的数。
2. **PPRAM-CRCW**：优先的 PRAM-CRCW，只允许最优先的处理器先写。
3. **APRAM-CRCW**：任意的 PRAM-CRCW，允许任意处理器自由写。

4.2.2 异步 PRAM 模型

记作 APRAM，它的特点是**分相**（Phase），每个处理器都有其私有存储（书里称作“局存”）、私有时钟（书里称作“局部时钟”）和局部程序；处理器之间通过共享全局存储器进行通信；无全局时钟，各处理器异步地独立执行各自的指令；若处理器之间存在任何时序依赖关系（书中称作“时间依赖关系”），须通过插入同步路障（Synchronization Barrier）以保持依赖源和目标的执行顺序。

APRAM 模型具有四种访存指令：

1. **全局读**：将全局存储单元中的内容读入私有存储单元中。
2. **局部操作**：对局存中的数执行操作，其结果是存入局存中。
3. **全局写**：将局存单元中的内容写入全局存储单元中。
4. **同步**：同步是计算中的一个逻辑点，在该点各处理器均需等待别的处理器到达后才能继续执行其局部程序。

4.2.3 BSP 模型

整体同步计算模型（Bulk Synchronous Parallel），字面含义是“大”同步模型，该模型使用了三个属性描述：模块（Components）、选路器（Router）和同步路障器执行时间 L（中文又称“超级步”）。BSP 是一种 MIMD-DM（DM：distributed memory）模型。

在一个超级步中，每个处理器执行各自的局部计算，通过选路器接收和发送消息；然后作一全局检查（同步），以确定当前超级步是否已由所有的处理器完成；然后前进到下一个超级步。

在分析 BSP 模型的性能时，假定局部操作可在一个超级步内完成。在每一超级步中，一个处理器至多发送或接收 h 条消息（称为 **h-relation**）。假定 s

是传输建立时间，g 定义为每秒处理器所能完成的局部计算数目与每秒选路器所能传输的数据量之比，那么传送 h 条消息的时间开销为 $gh + s$，若 $gh \geq 2s$，则超级步步长 $L \geq gh$（为什么？）。硬件可设置 L 的下限（例如宽的通信带宽减小 g），而软件可设置 L 的上限（例如并行粒度越大，L 越大）。

4.2.4 logP 模型

logP 模型是一种分布式存储的，点到点通信的多处理机模型，其中通信网络由一组参数来描述：

1. l（Latency）表示在网络中消息从源到目的地所遭到的**延迟**。
2. o（Overhead）表示处理器发送或接收一条消息所需的**额外开销**。
3. g（gap）表示处理器可连续进行消息发送或接收的最小时间间隔。
4. P（Processor）表示处理器/存储器模块数。

5 并行算法的一般设计策略

5.1 串行算法的直接并行化

6 并行算法的基本设计技术

6.1 划分设计技术

6.1.1 均匀划分技术

6.1.2 方根划分技术

7 并行算法的一般设计过程

8 基本通信操作

9 稠密矩阵运算

10 线性方程组的求解

10.1 稀疏线性方程组求解

10.1.1 共轭梯度法

（Conjugate Gradient），也叫**共轭斜量法**。其基本思想共轭梯度法用于求解类似于 $\mathbf{Ax}=\mathbf{b}$ 这样的一次方程组。通过对二次型函数

$$f(\mathbf{x})=\frac{1}{2}\mathbf{x}^T\mathbf{Ax}-\mathbf{b}^T\mathbf{x}+c$$

的变元 $\mathbf{x}$ 求导，得

$$f^{'}(\mathbf{x})=\mathbf{Ax}-\mathbf{b}$$

因此解方程组 $\mathbf{Ax}=\mathbf{b}$ 等价于求出当 $f^{'}(\mathbf{x})=\mathbf{Ax}-\mathbf{b}=0$ 时对应的 $\mathbf{x}$（若 $\mathbf{A}$ 为实二次型，则 $f^{'}(\mathbf{x})=0$ 意味着 $f(\mathbf{x})$ 取得最小值）。

要想充分理解共轭梯度法，我们首先要了解之前的用于求解大规模线性方程组的同类方法。

最速下降法（梯度下降法）

从任意一点 $\mathbf{x}_{(0)}$ 出发，每次都朝梯度方向（下降速度最快的方向）下降，从而得到解序列 $\mathbf{x}_{(1)},\mathbf{x}_{(2)},...$，直到两次下降的误差小于给定的误差上限。记第

i 次迭代的误差为 $\mathbf{e}_{(i)} = \mathbf{x}_{(i)} - \mathbf{x}_{(i-1)}$，残差为 $\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$，其中 $\mathbf{x}$ 表示方程组 $\mathbf{A}\mathbf{x} = \mathbf{b}$ 的解。于是

$$\mathbf{r}_{(i)} = -f^{'}(\mathbf{x}_i)$$

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{a}_{(i)}\mathbf{r}_{(i)}$$

其中 $\mathbf{a}_{(i)}$ 为第 i 步沿着残差的方向 $\mathbf{r}_{(i)}$ 的步长。步长 $\mathbf{a}_{(i)}$ 选取的基本原则是尽量让下一次迭代中 $f(\mathbf{x}_{i+1})$ 的值更小，这样才能更快地到达 $f(\mathbf{x})$ 的全局最小值。把 $f(\mathbf{x}_{(i+1)})$ 看作是 $\mathbf{a}_{(i)}$ 的函数，要使得 $f(\mathbf{x})$ 最小，也就是求

$$\frac{df(\mathbf{x}_{(i+1)})}{d\mathbf{a}_{(i)}} = 0$$

时的 $\mathbf{a}_{(i)}$ 值。根据链式法则，有

$$\frac{df(\mathbf{x}_{(i+1)})}{\mathbf{a}_{(i)}} = \frac{df(\mathbf{x}_{(i+1)})}{d\mathbf{x}_{(i+1)}}\frac{d\mathbf{x}_{(i+1)}}{d\mathbf{a}_{(i)}} = f^{'}(\mathbf{x}_{(i+1)})^T\frac{d\mathbf{x}_{(i+1)}}{d\mathbf{a}_{(i)}} = -\mathbf{r}_{(i+1)}^T\mathbf{r}_{(i)}$$

当 $\mathbf{r}_{(i)}$ 与 $\mathbf{r}_{(i+1)}$ 正交时，$\mathbf{r}_{(i+1)}^T\mathbf{r}_{(i)} = 0$，那么可以求出迭代步 $\mathbf{a}_{(i)}$

$$\mathbf{r}_{(i+1)}^T\mathbf{r}_{(i)} = 0$$

$$(\mathbf{b} - \mathbf{A}\mathbf{x}_{(i+1)})^T\mathbf{r}_{(i)} = 0$$

$$(\mathbf{b} - \mathbf{A}(\mathbf{x}_{(i)} + \mathbf{a}_{(i)}\mathbf{r}_{(i)}))^T\mathbf{r}_{(i)} = 0$$

$$(\mathbf{b} - \mathbf{A}\mathbf{x}_{(i)})^T\mathbf{r}_{(i)} - \mathbf{a}_{(i)}(\mathbf{A}\mathbf{r}_{(i)})^T\mathbf{r}_{(i)} = 0$$

$$(\mathbf{b} - \mathbf{A}\mathbf{x}_{(i)})^T = \mathbf{a}_{(i)}(\mathbf{A}\mathbf{r}_{(i)})$$

$$\mathbf{a}_{(i)} = \frac{\mathbf{r}_{(i)}^T\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^T\mathbf{A}\mathbf{r}_{(i)}}$$

综合，最速下降法就是

$$\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

$$\mathbf{a}_{(i)} = \frac{\mathbf{r}_{(i)}^T\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^T\mathbf{A}\mathbf{r}_{(i)}}$$

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{a}_{(i)}\mathbf{r}_{(i)}$$

为了减小复杂度，也可以采用以下方法求 $\mathbf{r}_{(i+1)}$

$$\mathbf{b}-\mathbf{A}\mathbf{x}_{(i+1)}=\mathbf{b}-\mathbf{A}(\mathbf{x}_{(i)}+\mathbf{a}_{(i)}\mathbf{r}_{(i)})$$

$$\mathbf{r}_{(i+1)}=\mathbf{r}_{(i)}-\mathbf{A}\mathbf{a}_{(i)}\mathbf{r}_{(i)}$$

梯度下降法在迭代过程中迭代方向都与上一次迭代方向正交。

雅各比迭代法

学高数时似乎接触过。雅各比迭代要求线性方程组 $\mathbf{A}\mathbf{x}=\mathbf{b}$ 中的矩阵 $\mathbf{A}$ 为非奇异矩阵，对于一般矩阵 $\mathbf{A}$，我们可以把 $\mathbf{A}$ 拆分为矩阵 $\mathbf{D}$ 和 $\mathbf{E}$（$\mathbf{A}=\mathbf{D}+\mathbf{E}$），其中 $\mathbf{D}$ 是对角阵（可逆），$\mathbf{E}$ 是剩下的部分，那么

$$\mathbf{A}\mathbf{x}=\mathbf{b}$$

$$(\mathbf{D}+\mathbf{E})\mathbf{x}=\mathbf{b}$$

$$\mathbf{D}\mathbf{x}=\mathbf{b}-\mathbf{E}\mathbf{x}$$

$$\mathbf{x}=\mathbf{D}^{-1}(\mathbf{b}-\mathbf{E}\mathbf{x})$$

为看得舒服，将 $\mathbf{x}=\mathbf{D}^{-1}(\mathbf{b}-\mathbf{E}\mathbf{x})$ 写作

$$\mathbf{x}_{(i+1)}=\mathbf{B}\mathbf{x}_{(i)}+\mathbf{z}$$

这就是雅各比迭代的迭代格式。其中 $\mathbf{B}=-\mathbf{D}^{-1}\mathbf{E}$，$\mathbf{z}=\mathbf{D}^{-1}\mathbf{b}$。关于迭代格式的构造，知乎的答案https://zhuanlan.zhihu.com/p/30965284讲得很细。如果矩阵 $\mathbf{B}$ 的谱半径（最大特征值）小于 1，那么迭代是可以收敛的，反之则不能收敛。

梯度下降法与雅各比迭代法公共的主要缺陷是：为了收敛到解附近，迭代方向可能变了很多次，造成迭代次数增多，从而增加计算复杂度。

共轭梯度法

如果能够选取一系列线性无关的方向向量 $\mathbf{d}_{(0)},\mathbf{d}_{(1)},(\mathbf{d})_{(2)},...,\mathbf{d}_{(n-1)}$，沿着每个方向只迭代一次，最后就能到达解 $\mathbf{x}$ 处，迭代的次数最多为 n 次，可以解决梯度下降法和雅各比迭代法的问题。

共轭 给定一个正定矩阵 $\mathbf{A}$，如果两个向量 $\mathbf{u}$ 和 $\mathbf{v}$ 满足 $\mathbf{u}^T\mathbf{A}\mathbf{v}=0$，则称 $\mathbf{u}$ 和 $\mathbf{v}$ 是关于矩阵 $\mathbf{A}$ 共轭的。

现在我们来找一组关于线性方程组 $\mathbf{A}\mathbf{x}=\mathbf{b}$ 中矩阵 $\mathbf{A}$ 两两共轭的基向量 $\mathbf{d}$。问题在于如何寻找这样的 $\mathbf{d}$。

我们先看一下残差向量 $\mathbf{r}$。对于第 $i+1$ 次迭代中的残差 $\mathbf{r}_{(i+1)}$，根据最速下降法中的公式，有

$$\mathbf{r}_{(i+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i+1)}$$

$$\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{a}_{(i)}\mathbf{r}_{(i)}$$

哎呀不管了，目前没找到有靠谱证明过程的资料，直接上共轭梯度法的公式吧

$$\mathbf{d}_{(0)} = \mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

$$\mathbf{a}_{(i)} = -\frac{\mathbf{d}_{(i)}^T\mathbf{r}_{(i)}}{\mathbf{d}_{(i)}^T\mathbf{A}\mathbf{d}_{(i)}}$$

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{a}_{(i)}\mathbf{d}_{(i)}$$

$$\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \mathbf{a}_{(i)}\mathbf{A}\mathbf{d}_{(i)}$$

$$\mathbf{d}_{i+1} = \mathbf{r}_{(i+1)} + \frac{\mathbf{r}_{(i+1)}^T\mathbf{r}_{(i+1)}}{\mathbf{r}_{(i)}^T\mathbf{r}_{(i)}}\mathbf{d}_{(i)}$$

11 快速傅里叶变换

12 并行程序设计基础

13 共享存储系统并行编程

14 分布存储系统并行编程

14.1 基于消息传递的并行编程（2019 年 7 月 10 日）

所谓基于消息传递的并行编程，是指用户必须显示地通过发送和接收消息来实现处理器之间的数据交换。根据问题分解的两种形式，消息传递并行性的开发也分为两种模式

1. **域分解：**将一个大的问题区域分解成若干个较小的问题区域，每个区域执行同样的指令，但处理的数据不同。通常采用 SPMD（Single Process Multiple Data）编程模式实现。

2. **函数分解：**又称**功能分解**，将一个大问题分解成若干个子问题，每个子问题执行不同的指令。通常采用 MPMD（Multiple Process Multiple Data）编程模式实现。

14.1.1 SPMD 并行程序

SPMD 按照节点的地位可分为两种结构

1. **主机/节点结构：**节点分为 master 和 slave。
2. **无主机结构：**每个节点都是对等的。

14.1.2 MPMD 并行程序

实现子问题之间相互作用方式有两种

1. **数据流方式：**（Dataflow）根据子问题数据的相关性来组织并行，按**数据驱动**（Data-Driven）的方式决定并行执行的先后次序。
2. **客户/服务器方式：**就是常见的 C/S 方式。

14.2 MPI 并行编程（2019 年 7 月 10 日）

（Message Passing Interface）。MPI 进程是**重量级**、**单线程**的进程。

14.2.1 最基本的 MPI

MPI 标准包含 6 个基本函数

函数名	功能
MPI_INIT	启动 MPI 计算
MPI_FINALIZE	结束 MPI 计算
MPI_COMM_SIZE	确定进程数
MPI_COMM_RANK	确定自己的进程标识符
MPI_SEND	发送一条消息
MPI_RECV	接受一条消息

下面以 MPICH2 为例说明个函数的详细信息。

MPI_INIT(int *argc, char **argv)

启动 MPI 计算，argc、argv 都是 main 函数的参数，将主函数参数传递给其他

节点。

MPI_FINALIZE()

结束 MPI 计算。

MPI_COMM_SIZE(comm, *size)

输入输出类型	参数名	解释
IN	comm	通信体（handle）
OUT	size	通信体 comm 中进程的个数（integer）

MPI_COMM_RANK(comm, *pid)

输入输出类型	参数名	解释
IN	comm	通信体（handle）
OUT	pid	通信体 comm 中当前进程的标识符（integer）

MPI_SEND(*buf, count, datatype, dest, tag, comm)

输入输出类型	参数名	解释
IN	buf	待传输数据的内存首地址（可选参数）
IN	count	待传输数据的长度（integer）
IN	datatype	待传输数据的类型（handle）
IN	dest	目标进程的 pid（integer）
IN	tag	消息标识符（integer）
IN	comm	通信体（handle）

MPI_RECV(*buf, count, datatype, source, tag, comm, status)

MPI（C Binding）	C
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

输入输出类型	参数名	解释
OUT	buf	数据存放位置的首地址（可选参数）
IN	count	数据的长度（integer）
IN	datatype	待传输数据的类型（handle）
IN	source	源进程的 pid（integer）
IN	tag	消息标识符（integer）
IN	comm	通信体（handle）
OUT	status	状态对象（statuse）（这是什么？）

MPI 数据类型与 C 语言数据类型对应关系如下

14.2.2 一个基于 MPICH 实现多节点并行的例子

本节使用的例子详见 “MPI_demo/mat_inc” 目录。该程序的主要功能是给一片连续的内存中不同的位置赋不同的值。

我有两台刀片服务器 node02(内网地址 192.168.1.101)、node03(内网地址 192.168.1.103)，采用粗粒度的域分解方式实现任务划分，两台机器分别处理内存的前半段和后半段。两台机器的 MPI 软件版本必须相同，我用的是 MPICH3。基本的 MPICH 软件包只包含 mpirun、mpiexec，用于执行 MPI 程序；编译采用 MPI 程序还需要安装 MPICH-devel 开发者软件包，包含 mpicc 编译器。两台刀片服务器的操作系统皆为 CentOS（具体版本未知），互连方式为 infiniteband。

以下是步骤：

1. 首先确保 node02 与 node03 之间可以免密 ssh 登录。使用 “ssh-keygen

-t [rsa] [target host]" 命令生成 ssh-key，储存在 id_rsa、id_rsa.pub 文件中。

2. 将用 mpicc 编译器编译好的程序分别拷贝到两个节点的同一相对目录下（mpirun 命令会携带主节点的环境参数到从节点），注意代码中需要有根据进程号进行域分解的代码。

3. 在主节点（我以 node02 为主节点）中通过 "mpirun -np [主节点进程数] [可执行文件] : -host [从节点地址] -np [从节点进程数] [可执行文件]" 执行程序。也可以通过 machinelist 添加更多的节点。

以下是通过执行命令 "mpirun -np 1 ./mat_inc_mpi.out : -host 192.168.1.103 -np 1 ./mat_inc_mpi.out" 产的生输出结果，该结果与对应串行程序的输出相同，验证了并行程序的正确性。

```
[root@ceph-mon QuBin]# mpirun -np 1 ./mat_inc_mpi.out : -host 192.168.1.103 -np 1 ./mat_inc_mpi.out

0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      1
8       19      20      21      22      23      24      25      26      27      28      29      30      31      32      33      34      35      3
6       37      38      39      40      41      42      43      44      45      46      47      48      49      50      51      52      53      5
4       55      56      57      58      59      60      61      62      63      64      65      66      67      68      69      70      71      7
2       73      74      75      76      77      78      79      80      81      82      83      84      85      86      87      88      89      9
0       91      92      93      94      95      96      97      98      99
```

14.2.3 群体通信