1. 各个并行计算模型
2. 单指令单数据（SISD）的CPU对加法指令译码后，执行部件先访问内存，取得第一个操作数；之后再一次访问内存，取得第二个操作数；随后才能进行求和运算。而在SIMD型的CPU中，指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。

(2)@@MIMD计算机具有多个异步和独立工作的处理器。在任何时钟周期内，不同的处理器可以在不同的数据片段上执行不同的指令，也即是同时执行多个指令流，而这些指令流分别对不同数据流进行操作。

@@SIMD是采用一个指令流处理多个数据流。（流水线？）

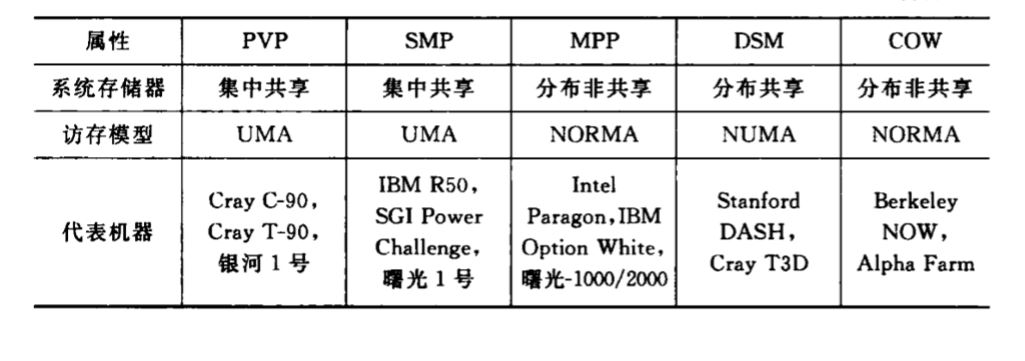
(3)PRAM（Parallel Random Access Machine，随机存取并行机器）模型，也称为共享存储的SIMD模型

LogP模型是一种分布存储的、点到点通信的多处理机模型



2.结构特性





UMA中央处理器，NUMA分布式处理器。（前两个都属于多处理器范畴） NORMA多计算机。

（1）并行向量处理机（PVP），对称多处理机（SMP），大规模并行处理机（MPP），分布共享存储处理机（DSM），工作站机器（COW），全高速缓存储存访问（COMA），CC-NUMA

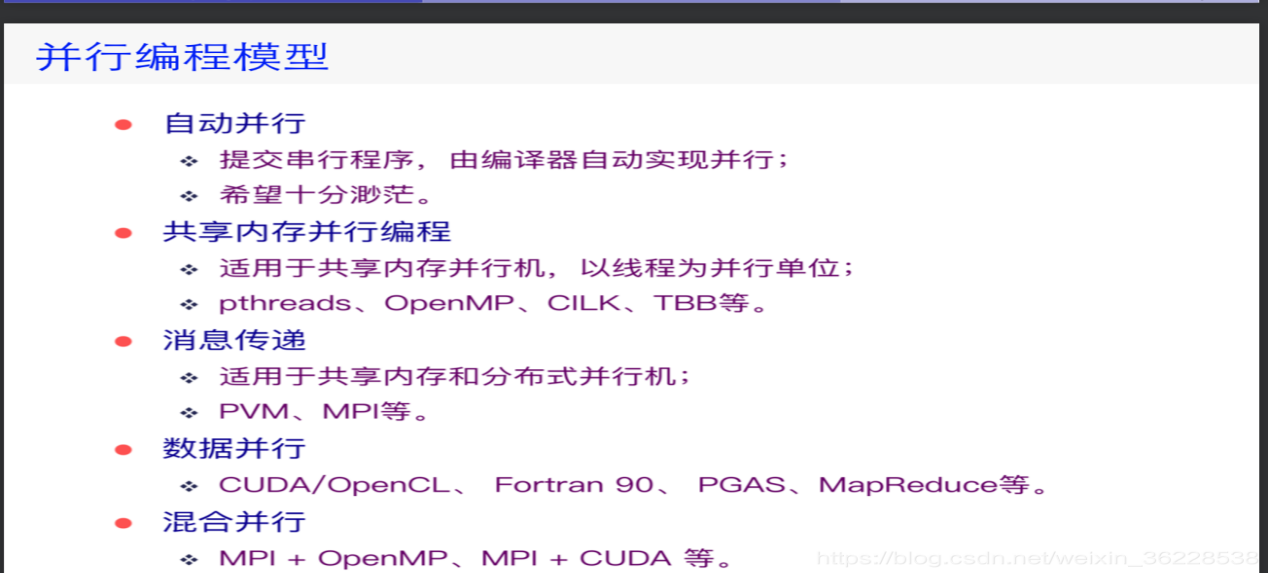
（2）对称多处理"（Symmetrical Multi-Processing）简称SMP。每个处理器可以等同地访问共享存储器，I/O设备和操作系统等服务。

COW的每个结点都是一个完整的工作站，各节点通过一种低成本的商品网络相联。

PVP使用专门的高带宽交叉开关网络将VP（向量处理器）连向共享存储模块。

1. openmp和mpi

OpenMP是共享内存模型,mpi是分布式内存模型



*openmp:*

(1) #include <omp.h>

(2) #pragma omp parallel for

(3)避免数据依赖和竞争

当一个循环满足以上五个条件时，依然可能因为数据依赖而不能够合理的并行化。当两个不同的迭代之间的数据存在依赖关系时，就会发生这种情况。

// 假设数组已经初始化为1

#pragma omp parallel for

for (int i = 2; i < 10; i++) {

factorial[i] = i \* factorial[i-1];

}

编译器会把这个循环多线程化，但是并不能实现我们想要的加速效果，得出的数组含有错误的结构。因为每次迭代都依赖于另一个不同的迭代，这被称之为竞态条件。要解决这个问题只能够重写循环或者选择不同的算法。

(4)

int sum = 0;

#pragma omp parallel for reduction(+:sum)

for (int i = 0; i < 100; i++) {

sum += array[i];

}

内部实现中，OpenMP 为每个线程提供了私有的sum变量，当线程退出时，OpenMP 再把每个线程的部分和加在一起得到最终结果。

（5）CRITICAL指令指定一块同一时间只能被一条线程执行的代码区域

#pragma omp critical

（6）

使用private子句和critical部分并行化的程序

#include <omp.h>

static long num\_steps = 100000;

double step;

#define NUM\_THREADS 2

void main ()

{

int i;

double x, sum, pi=0.0;

step = 1.0/(double) num\_steps;

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel private (x, sum)

{

id = omp\_get\_thread\_num();

for (i=id,sum=0.0;i< num\_steps;i=i+NUM\_THREADS){

x = (i+0.5)\*step;

sum += 4.0/(1.0+x\*x);

}

#pragma omp critical

pi += sum\*step

}

}

#pragma omp parallel for reduction(+:sum) private(x)

for (i=0;i<num\_steps; i++){

x = (i+0.5)\*step;

sum = sum + 4.0/(1.0+x\*x);

}

*mpi:*

(1)MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD); //将n广播到所有进程中

(2)MPI\_Reduce(&sum, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

//利用归约操作(MPI\_SUM)将所有进程的sum累加到root进程(0)的sum当中得到结果

(3)MPI\_Init(&argc,&argv);

//初始化。并行的部分全部放在MPI\_Init(&argc,&argv)和MPI\_Finalize()内部。

MPI\_Finalize();

(4)MPI\_Comm\_size(MPI\_COMM\_WORLD,&num\_procs);

//MPI\_COMM\_WORLD是所有进程的集合，在执行完MPI\_Init后自动产生

(5)MPI\_Comm\_rank(MPI\_COMM\_WORLD,&my\_rank);

//得到本进程在通信空间中的rank值,即在组中的逻辑编号

//(该 rank值为0到p-1间的整数,相当于进程的ID。)

消息发送：MPI\_Send函数用于发送一个消息到目标进程。

int MPI\_Send(void \*buf, int count, MPI\_Datatype dataytpe, int dest, int tag, MPI\_Comm comm)

3.PCAM---设计方法学

划分(Partitioning)：分解成小的任务，开拓并发性

通讯(Communication)：确定诸任务间的数据交换，监测划分的合理性

组合(Agglomeration)：依据任务的局部性，组合成更大的任务

映射(Mapping)：将每个任务分配到处理器上，提高算法的性能

Warp：CUDA中每个线程块分为若干个组(称为warp)，每个warp包含32个线程，物

理上以SIMD方式并行

4.

不允许同时读和同时写的PRAM-EREW

允许同时读但不允许同时写的PRAM-CREW

允许同时读和同时写的PRAM-CRCW

5.MIMD机器上的PSRS排序算法：

(1)均匀划分:将n个元素A[1..n]均匀划分成p段，分配给p个处理器

(2)局部排序:pi调用串行排序算法对A[(i-1)n/p+1..in/p]排序

(3)选取样本:pi从其有序子序列A[(i-1)n/p+1..in/p]中选取p个样本元素

(4)样本排序:用一台处理器对p2个样本元素进行串行排序

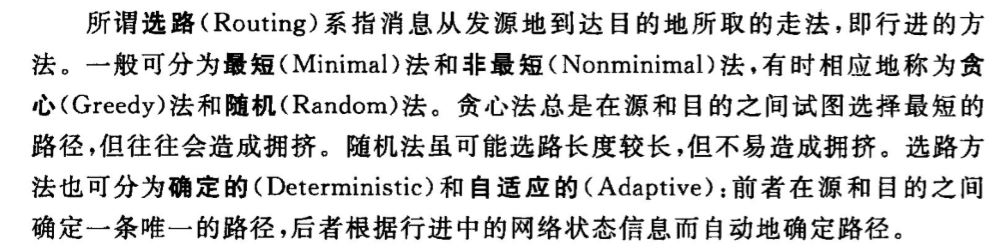
(5)选择主元:用一台处理器从排好序的样本序列中选取p-1个主元,并播送给其他pi

(6)主元划分:pi按主元将有序段A[(i-1)n/p+1..in/p]划分成p段

(7)全局交换:各处理器将其有序段按段号交换到对应的处理器中（一定保证均匀划分？？）

(8)归并排序:各处理器对接收到的元素进行归并排序

6.选路方法：



1. 加速比：n个处理器加速比n/[1+(n-1)\*f]

8.并行计算加速的基本原理是将一个算法中的可并行执行的部分放到多个处理器

上同时执行，而并行算法的加速比中的 *p* 是指它能够将问题分解成能够并行执行的任务

数量。因此对于一个具有良好可扩放性的并行算法，任务的规模（个数）应该要随着问

题的规模的增加而增加，这样才能充分利用更多的处理器，提高 *p*。

2019 年春季学期并行计算期末考试

Edited by Lyncien

2019.06.10

一、

填空题 10 \* 2%

1. 并行计算体系结构有 SIMD-SM/MIMD-SM/SIMD-DM/MIMD-DM，则 PRAM 模型是

（1） ，APRAM 模型是 （2） ，LogP 模型是 （3） 。

2. 令 W(n)是某并行算法 A 在运行时间 T(n)内所执行的运算量，则 A 使用 p 台处

理器可在 （4） 时间内执行完毕。

3. 对于求最大值的算法，SIMD-EREW 结构上使用 n/2 个处理器可在 （5） 时间

内完成，SIMD-CRCW 结构上使用 n^2 个处理器可在 （6） 时间内完成。

4. 高斯-赛德尔迭代法五点格式的 A 矩阵是 （7） 对角矩阵，并行化方法是

（8） 。

5. OpenMP 属于 （9） 并行编程模型，MPI 属于 （10） 并行编程模型

二、

简答题 4 \* 5%

1. 解释概念 SIMD，SPMD，SMP，PCAM，Warp

2. MPI 为什么要使用消息标签？

3. 稀疏方程组的求解为什么使用迭代法（如共轭梯度法）而不是直接法（如高斯

消元法）？

4. CUDA 中 CPU 与 GPU、线程块内、线程块间同步的方法与代码？

三、

综合题 4 \* 15%

1. 阅读代码

#include <stdio.h>

#include <\_\_\_\_\_\_>

int main ()

{

int i, n;

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

/\* Some initializations \*/

n = 100;

result = 0.0;

for (i=0; i < n; i++)

{

a[i] = i \* 1.0;

b[i] = i \* 2.0;

}

pragma omp \_\_\_\_\_\_

for (i=0; i < n; i++)

{

pragma omp \_\_\_\_\_\_

result = \_\_\_\_\_\_ + (a[i] \* b[i]);

}

printf("Final result= %f\n",result);

}

（1） 补全代码并说明程序的功能

（2） 使用另一种方式实现 15-20 行的求和

2. 给出环上收集（all-to-one）的选路（CT）的算法，作出示意图，分析时间。

3. 对于 PRAM 下求 n 个数前缀和的算法（课本算法 7.9）

（1） 是否是并行成本最优？是 EREW/CREW/CRCW 中的哪一种？

（2） 给出使之并行成本最优改进方法的伪代码，并分析成本最优性。

4. 对于离散傅里叶变换

（1） 给出蝶式 FFT 算法的时间复杂度，可以使用哪种并行算法设计技术使之并行

化？

（2） SIMD-BF 上的 FFT 算法，蝶形网络上每个处理器的 w 权因子有两种计算方

法，比较分析它们的计算工作量。

个人答案（可能有误，仅供参考）

一、

1.（1）SIMD-SM （2）MIMD-SM （3）MIMD-DM

2.（4）O(W(n)/p+T(n))

3.（5）O(nlogn) （6）O(1)

4.（7）三 （8）红黑着色并行算法

5.（9）共享变量 （10）信息传递

二、

1.SIMD：单指令多数据流

SPMD：单线程多数据流

SMP：对称多处理机

PCMA：设计并行算法的四个阶段：划分(Partitioning)，通讯(Communication)，组合

(Agglomeration)，映射(Mapping)的首字母

Warp：CUDA中每个线程块分为若干个组(称为warp)，每个warp包含32个线程，物

理上以SIMD方式并行

2. 当发送者连续发送两个相同类型消息给同一个接收者，如果没有消息标签，接收

者将无法区分这两个消息。添加标签使得服务进程可以对两个不同的用户进程分别

处理，提高灵活性。

3.对于大型、稀疏线性方程组，迭代法比直接法简单、占用存储空间小；对于在有限

步内无法得到问题的解时，迭代法可以在有限的迭代步数后，停止运算而得到足够好

的近似解。

4. CPU与GPU：如果CPU在接下来的操作中需要用到GPU的计算结果，则CPU必须

阻塞等待GPU执行完毕。可在kernel后添加一条同步语句cudaThreadSynchronize ()实

现。

线程块内：\_\_syncthreads()只有当同一个块内的所有线程都到达函数\_\_syncthreads()时

才会继续往下执行

线程块间：同一个grid中的不同线程块之间不能同步，即CUDA运行时库中没有提供

此类函数。 但可以通过终止一个kernel来实现同步

三、

1.（1）

omp.h

parallel for

critical

result

功能：向量点积（2）并行归约方法

#pragma omp parallel for default(shared) private(i) reduction(+:result)

for (i=0; i < n; i++)

{

result = result + (a[i] \* b[i]);

}

2.

先发送至距离为 1 的处理器；

发送距离为 2 的处理器；

…

发送距离为 2^i 的处理器

时间：

(+ 2+ 2ℎ

= + ℎ)

3.（1）不是并行成本最优，并行 c(n) = O(nlogn)，串行为 O(n)

CREW

（2）采用级联技术：先小范围串行后并行

处理器个数 p(n) = n / logn，每个处理器负责 n / (n / logn) = O(logn)

算法分为两阶段

阶段一：每个处理器串行求解 O(logn)个元素的和，花费时间 O(logn)

阶段二：再对 n / logn 个和应用平衡树求前缀和方法，

花费时间 O(log(n / logn)) = O(logn)

总时间为 O(logn)

并行成本 c(n) = p(n)t(n) = O(n/logn)O(logn) = O(n) = 串行成本，是最优

4.（1）T(n) = 2T(n/2) + O(n) => T(n) = O(nlogn)

可以用分治设计技术

（2）方法一：各处理器独立计算权因子

方法二：最后一行先计算权因子，然后各列处理器各自平方后即为上一行的对

应列的权因子，logn 步后完成所有计算，除了最后一行，其余处理器权因子计

算只需要一次乘法。因此比第一种方法更好。