# 课堂练习-1

1. 请写出存储受限条件下的加速比公式，并推导出其他约束条件下的加速比公式。
2. 为什么说计算π的并行算法是有效的？
3. 对于一个在给定并行体系结构上解决给定问题的并行算法，若下面的条件变化时，并行效率是增加还是减少？若其他独立参数固定。
   - ✓ 处理器数目增加
   - ✓ 处理器的计算速度增加
   - ✓ 问题规模增加
   - ✓ 通讯量增加
   - ✓ 通讯带宽增加
   - ✓ 通讯延迟增加
   - ✓ 通讯端口增加
4. 什么是成本最优的并行算法？
5. 概要说明并行计算模型PRAM、BSP和logP的体系结构及主要参数。

# 课堂练习-2

1. 并行算法设计的一般过程PCAM是指什么？
2. Cannon算法与简单分块并行算法相比，优点是什么？
3. DNS算法的时间步长是多少？为什么可以达到？

4. 什么是Fork-Join模型？
5. 什么是共享存储编程模型？共享存储编程模型的主要特征？

# 课堂练习-3

1. OpenMP并行编程模型的本质是什么?
2. OpenMP的三个组成部分?
3. 说明private、firstprivate lastprivate、和threadprivate的区别。
4. 循环for并行化的调度方式都有哪些?
5. 什么是组合的并行共享任务结构?
6. 在OpenMP中，如何实现对共享变量的并发读写?
7. 以计算π为例说明OpenMP编程的多种并行化结构。

# 使用并行域并行化

```
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
void main ()
{    int i;
     double x, pi, sum[NUM_THREADS];
     step = 1.0/(double) num_steps;
     omp_set_num_threads(NUM_THREADS)
    #pragma omp parallel
    {
            double x;
            int id;
            id = omp_get_thraead_num();
            for (i=id, sum[id]=0.0;i< num_steps; i=i+NUM_THREADS){
                        x = (i+0.5)*step;
                        sum[id] += 4.0/(1.0+x*x);
            }
    }
     for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

# 使用共享任务结构并行化

```
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
void main ()
{
      int i;
      double x, pi, sum[NUM_THREADS];
      step = 1.0/(double) num_steps;
      omp_set_num_threads(NUM_THREADS)
      #pragma omp parallel
      {
                  double x;
                  int id;
                  id = omp_get_thraead_num();
                  sum[id] = 0;
                  #pragma omp for
                  for (i=0;i< num_steps; i++){
                                    x = (i+0.5)*step;
                                    sum[id] += 4.0/(1.0+x*x);
                  }
      }
for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

# 使用private和critical并行化

```
#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
    }
}
```

# 使用并行归约

```
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
void main ()
{    int i;
     double x, pi, sum = 0.0;
     step = 1.0/(double) num_steps;
     omp_set_num_threads(NUM_THREADS)
     #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);
     }
     pi = step * sum;
}
```

# 课堂练习-4

1. 什么是MPI的消息、数据类型、通信域？
2. MPI的扩展数据类型有什么应用？
3. MPI的阻塞通信和非阻塞通信？
4. 点到点通信模式有哪些？
5. 有哪些方法解决MPI通信的死锁问题？
6. MPI群集通信模式有哪些？

# 课堂练习-5

1. MapReduce是适合什么体系架构计算机的一种编程模式？举例说明其主要应用领域有哪些。
2. 请给出Word Count的MapReduce实现的伪代码描述。
3. Shuffle、Combiner和Partioner的作用是什么？
4. MapReduce采用哪些机制来实现容错？