一、选择题

1. 高性能计算的性能衡量的单位是 floaps，那 1Tfloaps 是___B___.

A. 每秒千万亿次的浮点运算

B. 每秒万亿的浮点运算

C. 每秒百亿亿的浮点运算

D. 每秒拾亿的浮点运算

2. 下列哪台高性能计算机是目前性能最高的超级计算机(C)。

A. 天河二号

B. Sunway TaihuLight

C. Summit

D. sierra

3. 下列计算机中哪个不属于分布式内存系统的计算机：（C）

A. 集群

B. 超级计算机

C. 冯诺依曼计算机

D. 数据中心

4. 下列关于 MPI 函数库中 MPI_Send()与 MPI_Recv()说法正确的是（B）

A. MPI 中消息发送函数 MPI_Send()总是阻塞的

B. MPI 中消息接收函数 MPI_Recv()总是阻塞的

C. MPI 中消息发送函数 MPI_Send()总是缓冲的

D. MPI 中消息接收函数 MPI_Recv()总是缓冲的

5. 下列关于 MPI 中函数或变量名不正确的是（C）。

A. MPI_SUCCESS

B. MPI_Init

C. MPI_Comm_world

D. MPI_Comm_rank

6. MPI 中接收函数 MPI_Recv()在不知道以下哪个信息的情况下不能接收到消息？（C）

A. 消息中的数据量

B. 消息中发送者

C. 消息的数据类型

D. 消息的标签

7. 在 Pthreads 中，下列哪一个不是用于保护临界区的方式。（A）

A. 锁机制

B. 忙等待

C. 信号量

D. 互斥量

8. 在 OpenMP 中，假设利用 2 个线程对具有 5 次迭代的 for 循环进行并行，下列哪种调度方式可能会使得线程 1 上执行迭代 0,1,4；线程 2 上执行迭代 2,3.（C）

A. schedule(static,3)

B. schedule(static,1)

C. schedule(static,2)

D. schedule(dynamic,1)

9. 在 MPI 中，如果要将一个具有 100 个分量的数组分发给通信子中的所有进程（10 个进程），能够采用如下哪个集合通信函数？（B）

A. MPI_Reduce(…)

B. MPI_Scatter(…)

C. MPI_Send(…)

D. MPI_Bcast(…)

10. 在 OpenMP 中，如果要进行显式路障的设置，应采用哪个指令（B）

A. reduction

B. Barrier

C. critical

D. atomic

二、填空题

1. 常用的 Linpack 基准测试程序有 HPL 和 HPCC。
2. MPI 的英文全称为 Message-Passing Interface ；
   集合通信的英文为 Collective communication 。
3. MPI 的通信分为 点对点通信 和集合通信。
4. 请写出在 3 个主机上启动 MPD 进程，形成 MPI 运行时的环境的命令：
   mpdboot -n 3 -f mpd.hosts 。（注目标主机列表由 $HOME/mpd.hosts 文件制定 ）
5. 只有接收进程可以使用通配符参数，常见的通配符参数包括 MPI_ANY_SOURCE 和 MPI_ANY_TAG。
6. OpenMP 适用于 共享内存 系统进行并行程序设计，其中#pragma omp critical 的功能是 保护临界区，使同一时间内只能有一个线程执行临界区代码 。
7. OpenMP 中，最主要是的编译指导语句，一条编译指导语句由 指令 和子句组成。
8. 在 OpenMP 中，并行 for 循环的指令包括：for 指令和_____。
9. MPI_Scatter(a,local_n,MPI_DOUBLE,local_a,local_n,MPI_DOUBLE,0,comm)这条语句执行的功能 将进程 0 中数组 a 的各元素按照 local_n 的块大小以块划分的方式发送给通信子 comm 内的所有进程。
10. #pragma omp parallel num_threads(thread_count) reduction(+:global_result) 这条编译性指导语句的作用是 启动 thread_count 个线程并行执行代码块，并将代码块中的 global_result 作为规约变量，其中规约操作为加法（注意规约变量为共享作用域，但在各进行内会创建其私有副本） 。

三、简答题

1. 在linux系统中，对文件名为hello.c的MPI源程序、pthreads源程序和OpenMP源程序分别写出采用3个进程或线程的MPI、Pthreads和OpenMP的编译和运行的命令。

答：

MPI的编译和运行的命令分别为：

mpicc -g -Wall -o mpi_hello hello.c

mpiexec -n 3 mpi_hello

Pthreads的编译和运行的命令分别为：

gcc −g −Wall −o pth_hello hello . c −lpthread

./ pth_hello 3

OpenMP的编译和运行的命令分别为：

gcc −g −Wall −fopenmp −o omp_hello hello.c

./ omp_example 3

2. 考虑下面的循环

```
a[0]=0;
for(i=1;i<n;i++)
    a[i]=a[i-1]+2i;
```

在这个循环中显然有循环依赖，因为在计算a[i]前必须先算a[i-1]的值。请找一种方法消除循环依赖，并且并行化这个循环。

答：

观察如下的规律：

a[0] = 0

a[1] = a[0] + 2 = 0 + 2

a[2] = a[1] + 4 = 0 + 2 + 4

a[3] = a[2] + 6 = 0 + 2 + 4 + 6

a[4] = a[3] + 8 = 0 + 2 + 4 + 6 + 8

由此可得，$a[i]=2\sum_{j=0}^{i} j=i(i+1)$；

所以我们能够重新写代码为

```
for(i=0;i<n;i++)
    a[i]=i*(i+1);
```

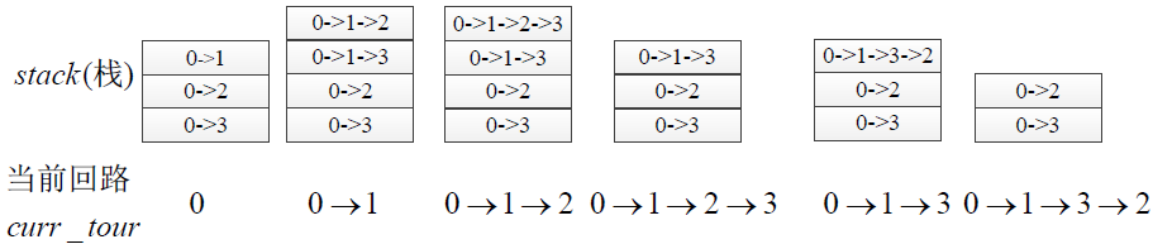
在此循环中，任何迭代的结果都不会再次使用。因此，代码可以与指令的并行化：

```
# pragma omp parallel for num_threads(thread_count) \
    default(none) private(i) shared(a, n)
for (i = 0; i < n; i++)
     a[i] = i*(i+1);
```

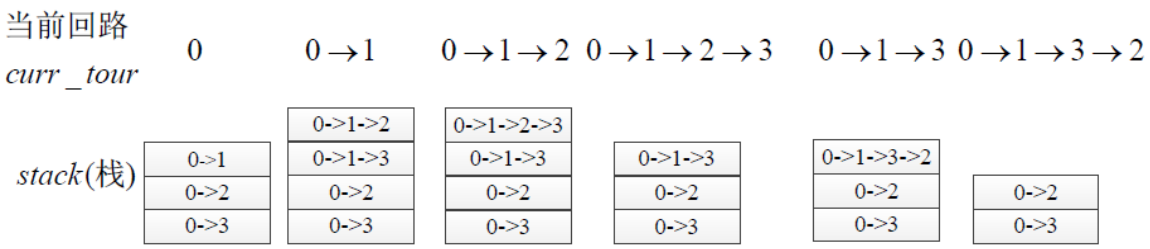
3. 下述代码是非递归深度优先搜索的代码片段，并假设Feasible(curr_tour, nbr)函数始终可行（为真）。现考虑4个城市的TSP问题，请画出栈和当前回路的运行状态，直到当前回路curr_tour为$0\to1\to3\to2$；请找出下述并行代码的临界区，并采用OpenMP的方式解决临界区问题。

```
Push_copy(stack,tour);//将回路压入栈中
while(!Empty(stack)){
    curr_tour=Pop(stack);//将回路弹出栈中
    if(City_count(curr_tour)==n){
        if(Best_tour(curr_tour))
            Update_best_tour(curr_tour);
    }else{
        for (nbr = n-1; nbr >= 1; nbr--)
            if (Feasible(curr_tour, nbr)) {
                Add_city(curr_tour, nbr);
                Push_copy(stack, curr_tour, avail);
                Remove_last_city(curr_tour);
            }
    }
    Free_tour(curr_tour);
}
```

答：



或者



并行代码的临界区为：

Update_best_tour(curr_tour);

采用 OpenMP 的方式保护临界区：

#pragma omp critical

Update_best_tour(curr_tour);

4. 判断下列程序是否能够安全通信，且说明原因。若如果不安全，请说明原因并给出相应的解决方案。

```
comm=MPI_COMM_WORLD;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
if(rank==0)
{    MPI_Recv(x2 , 3 , MPI_INT,1,tag,comm,&status);//A 语句
     MPI_Send(x1 , 3 , MPI_INT,1,tag,comm);//B 语句   }
if(rank==1)
{    MPI_Recv(x1, 3 ,MPI_INT,0,tag,comm,&status);//C 语句
     MPI_Send(x2, 3 ,MPI_INT,0,tag,comm);//D 语句   }
……
```

答案：不能够安全通信。

发送消息可能是阻塞或缓冲，而接收消息始终是阻塞。由于进程 0 和进程 1 都是先接收消息，后发送消息，所以进程 0 和 1 都会阻塞等待，此时没有任何进行发送消息。因此，上述代码会发送死锁，不能够安全通信。

解决方案为：将 A 语句与 B 语句进行调换顺序；或者 C 语句与 D 语句进行调换顺序。

5. 请按自己的理解说明并行计算和分布式计算的异同点。

并行计算(parallel computing)是指一个程序通过多个任务紧密协作来解决某一个问题。分布式计算(distributed computing)是指一个程序需要与其它程序协作来解决某个问题。

并行计算与分布式计算的区别：

1）问题的来源与应用领域不同，并行计算主要来自于科学计算领域，而分布式计算主要来自于商业领域。

2）系统架构不同：并行计算主要是指在许多核或处理器上进行求解一个问题；而分布式计算更强调的是跨系统、跨区域进行的协同工作来解决一个问题。

3）分布式系统强调的是资源的共享，和并发。

四、程序阅读题

1.写出下述程序的运行结果。

```
int k=100;
#pragma omp parallel for private(k)
for(k=0;k<2;k++)
{

    printf("k=%d\n",k);
}
printf("last k=%d\n",k);
```

运行结果为：

k=0
k=1
last k=100 。

2. 写出以下程序段的运行结果：

```
j=0, ThreadId=0
j=1, ThreadId=1
```

，若将#pragma omp parallel num_threads(2){}

删除，程序段的运行结果：

```
j=0, ThreadId=0
j=1, ThreadId=0
```

。

```
int j=0;
#pragma omp parallel num_threads(2)
```

```
{
#pragma omp for
    for(j=0;j<2;j++)
    printf("j=%d, ThreadId=%d\n",j,omp_get_thread_num());}
```

五、程序题

1. 请将以下程序改写为 10 个线程执行的 OpenMP 并行程序，以实现使用如下的公式对 π 进行估计的功能

$$\pi = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
    long long n, i;
    double factor;
    double sum = 0.0;
    n=1000;
    for (i = 0; i < n; i++) {
        sum += factor/(2*i+1);
        factor=-factor;
    }
    sum = 4.0*sum;
    printf("Our estimate of pi = %.14f\n", sum);
    return 0;
    }   /* main */
```

答：

```
int main(int argc, char* argv[]) {
    long long n, i;
    int thread_count=10;
    double factor;
    double sum = 0.0;
    n=1000;
#   pragma omp parallel for num_threads(thread_count) reduction(+: sum) private(factor)
```

```
    for (i = 0; i < n; i++) {
        factor = (i % 2 == 0) ? 1.0 : -1.0;
        sum += factor/(2*i+1);
    }
    sum = 4.0*sum;
    printf("With n = %lld terms and %d threads,\n", n, thread_count);
    printf("      Our estimate of pi = %.14f\n", sum);
    return 0;
}   /* main */
```

2. 采用 MPI 实现如下的梯形积分，即估计函数 $y = f(x)$ 在区间 $[a,b]$ 上与 x 所围阴影部分的面积，如下图所示。

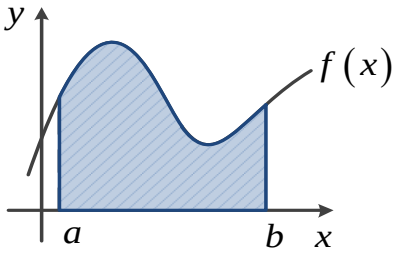


图 1 梯形积分

完成如下要求：

1) 主函数：采用**点对点通信**，即 0 号进程接收消息并打印，其它进程发送消息；

2) **接收用户输入函数 Get_input**：采用广播 MPI_Bcast 函数实现。

注意：采用如下已编写的梯形积分函数 Trap(double left_endpt, double right_endpt, int trap_count, double base_len)。

```
double Trap(
        double left_endpt    /* in */,
        double right_endpt /* in */,
        int       trap_count    /* in */,
        double base_len       /* in */) {
    double estimate, x;
    int i;
    estimate = (f(left_endpt) + f(right_endpt))/2.0;
    for (i = 1; i <= trap_count-1; i++) {
        x = left_endpt + i*base_len;
        estimate += f(x);
    }
    estimate = estimate*base_len;
    return estimate;
} /*    Trap    */
```

答 1）

程序 1 main 函数

```
int main(void) {
    int my_rank, comm_sz, n, local_n;
    double a, b, h, local_a, local_b;
    double local_int, total_int;
    int source;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    Get_input(my_rank, comm_sz, &a, &b, &n);
    h = (b-a)/n;                /* h is the same for all processes */
    local_n = n/comm_sz;    /* So is the number of trapezoids    */

    local_a = a + my_rank*local_n*h;
    local_b = local_a + local_n*h;
    local_int = Trap(local_a, local_b, local_n, h);

    if (my_rank != 0)
        MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
                MPI_COMM_WORLD);
    else {
        total_int = local_int;
        for (source = 1; source < comm_sz; source++) {
            MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            total_int += local_int;
        }
    }
    if (my_rank == 0) {
        printf("With n = %d trapezoids, our estimate\n", n);
        printf("of the integral from %f to %f = %.15e\n",
             a, b, total_int);
    }
    MPI_Finalize();
    return 0;
} /*    main    */
```

程序 2 Get_input 函数

```
void Get_input(int my_rank, int comm_sz, double* a_p, double* b_p,
        int* n_p) {
    int dest;
```

```
    if (my_rank == 0) {
        printf("Enter a, b, and n\n");
        scanf("%lf %lf %d", a_p, b_p, n_p);
        for (dest = 1; dest < comm_sz; dest++) {
            MPI_Send(a_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
            MPI_Send(b_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
            MPI_Send(n_p, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
        }
    } else { /* my_rank != 0 */
        MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
    }
}   /* Get_input */
```