草 稿 区

软件学院本科生 2020—2021 学年第 二 学期《并行与分布式程序设计》课程期末考试试卷（B 卷）

专业： **年级：** **学号：** **姓名：** **成绩:**

得 分

一、单选及填空（本题共 28 分，每空 2 分，请把答案按题号汇总到表格中）

1	2	3	4	5	6	7	8	9	10	11	12(1)	12(2)	13

1. 在高性能领域，超级计算机的发展水平体现着一个国家的国家战略与科技水平，彰显其综合国力。从 2013 年发布到 2016 年 6 月期间，我国超算（ ）一直居于全球超级计算机 500 强榜首。
   A. 神威·太湖之光　　B. 天河三号　　C. 天河二号　　D. 富岳
2. 一个 SSE 寄存器可容纳（ ）个单精度浮点数。
   A.2　　B.4　　C.8　　D.16
3. 以下哪条不是推动并行计算发展的因素？（ ）
   A. 多核、众核具有巨大的功耗优势　　B.单 CPU 发展已能满足应用需求
   C.利用标准硬件构造并行机令升级容易　　D.编程环境标准化逐步发展
4. SSE intrinsics _mm_load_pd 命令的功能是（ ）。
   A.对齐向量读取单精度浮点数　　B.未对齐向量读取单精度浮点数
   C.对齐向量读取双精度浮点数　　D.未对齐向量读取双精度浮点数
5. 在下面问题中，SIMD 并行最不适合（ ）。
   A. 向量加法　　B. 向量中元素排序　　C. 矩阵向量乘法　　D. 矩阵加法
6. 主线程创建了 4 个从线程，对它们执行 pthread_join,然后打印一条信息，从线程打印各自的线程号，未使用任何同步，则主线程打印的消息和从线程打印的线程号的相对顺序（ ）。
   A.必然主线程前、从线程后　　B.必然从线程前、主线程后
   C.必然相互交织　　D.各种顺序皆有可能
7. OpenMP 编译指示的作用范围是_____。

A.其后一个语句　　B.其后连续语句　　C.其后直到函数结束　　D.整个函数

8. 对于多线程各自进行本地运算，然后由主线程汇总结果的模式，下面说法正确的是（　　）。

A.在同构核心上，线程运行速度一样，主线程无需等待，直接汇总结果即可

B.线程运行速度可能不一致，必须采用同步保证主线程汇总正确结果

C.太多本地运算，不能体现并行效果，不是好的模式

D.主线程汇总结果在性能上必然不如多线程并行汇总结果

9. 在 OpenMP 编程的循环调度中，假设有 15 个迭代任务和 3 个线程，若使用 stastic 调度类型，那么缺省的一次分配给一个线程的任务块大小为（　　）。

A. 5　　B. 3　　C. 1　　D. 指数级减小

10. 关于 MPI 是什么,以下说法错误的是（　　）。

A.一种消息传递编程模型标准　　B.一种共享内存编程模型标准

C.编程角度看是 C++/Fortran 等的库　　D.基于 SPMD 模型

11. 代码：“for (i=0; i<10; i++)　A[i] = A[i+1]+1; ”此循环（　　）数据依赖。

A.存在　　B.不存在　　C.不确定　　D.以上皆错

12. 若串行快速排序算法 30s，串行起泡排序算法时间 200s，一个 5 核并行起泡排序算法用时 60s，则该并行起泡排序算法的加速比是(1)_____，效率是(2)_____。

13. 某串行程序运行时间为 20s，可并行化比例为 95%，若用 p 个核将其并行化，其加速比的理论上限是___。

草　稿　区

得　分

二、多项选择题（本题共 10 分，每小题 2 分，请把本题答案按题号汇总到表格中）

1	2	3	4	5

1. 编译器编译 OpenMP 并行循环时，会自动生成一些代码,其中不包括（　　）。

A.创建和管理线程代码　　B.循环划分给线程的代码

C.找出数据依赖的代码　　D.线程同步的代码

2. 动态任务划分相对于静态任务划分的优点是（　　）。

A.确保负载均衡　　B.任务粒度细　　C.计算复杂度低　　D.并行效率高

3. 在 OpenMP 编程中，可以进行 parallel for 循环并行化的代码限制条件描述正确的是（　　）。

A.循环变量必须是带符号整数或指针　　B.需要有明确的、与循环变量相应的循环不变量

C.循环体中无进/出控制流　　D.这些限制主要是为了使系统在循环执行前确定执行迭代的步数

草 稿 区

4. 下列关于“冯•诺伊曼瓶颈”的表述错误的是（ ）。

A. CPU 计算能力发展迅速，CPU 与高速缓存间数据传输成为瓶颈　B. 与 CPU 计算能力不够有关

C. CPU 计算能力发展迅速，CPU 与内存数据传输成为瓶颈　D. 与 CPU 和内存分开设计有关

5. 下面做法中，人们普遍认为可以提高并行程序的运行效率的有（ ）。

A. 减少内存中数据的访问次数　B. 减少线程的创建、销毁操作

C. 尽量均衡地分配任务　D. 尽可能多的增加线程数量

得 分

三、判断题（本题共 10 分，每小题 1 分，请把本题答案按题号汇总到表格中）

1	2	3	4	5	6	7	8	9	10

1. 截止到 2019 年末，Top 500 的超算机中总计算性能最强国家是中国。（ ）
2. 在 OpenMP 编程中，能进行 parallel for 循环并行化的代码循环变量必须是带符号整数或指针。（ ）
3. OpenMP 编程可以实现自动并行化，并且可以确保并行部分实现加速。（ ）
4. Pthread 编程中，主线程创建了 4 个从线程，对它们执行 pthread_join,然后打印一条信息，从线程打印各自的线程号，未使用任何同步，则主线程打印的消息和从线程打印的线程号的相对顺序可能是任意的。（ ）
5. 选用 MPI 主从模型处理矩阵每行排序疑问,主进程每次向一个从进程发送 10 行作为一个使命相关于每次发送 1 行的长处是削减了通讯开销。（ ）
6. Pthread 程序和 OpenMP 程序中线程均可通过特定 API 获得自身编号。（ ）
7. n*n 的两个矩阵相乘,问题规模为 n^3。（ ）
8. 指令级并行让多个处理器部件或功能单元并行执行指令来提高计算性能。（ ）
9. OpenMP 编程只提供了隐式同步的方式，没有显示同步功能。（ ）
10. 一般情况下，MPI 程序中发送和接收消息的两个进程要求必须在同一个网段中。（ ）

得 分

四、简答题（本题共 20 分）

1. 请简述有了缓存一致性，为什么还需要多线程的同步策略。（4 分）

草 稿 区

2. 请对下列并行算法设计中的名词进行解释。（6 分）

(1)竞争条件：

(2)阿姆达尔定律：

时非一个串行程序的执行几个子部并行化，无论多个核，通过并行化产生的加速比都有限

(3)缓存一致性：

3. 请简述 Flynn 分类法对计算机系统的分类。 (4 分)

4. MPI 编程中提供了多种组通信的原语。请写出不少于 6 个组通信原语，并分别简述其功能。（6 分）

草 稿 区

得 分

五、按要求写出相应代码（本题共 32 分）

1. 补全下面程序代码，程序实现的是 SSE 版本的矩阵乘法,c=a*b。（10 分）

①:

②:

③:

④:

⑤:

```
#include <stdio.h>

#include <pmmintrin.h>
#include <stdlib.h>
#include <algorithm>

const int maxN =1024;                    // magnitude of matrix

int n;
float a[maxN][maxN];
float b[maxN][maxN];
float c[maxN][maxN];

void sse_mul(int n, float a[][maxN], float b[][maxN], float c[][maxN]){
__m128 t1, t2, sum;
      for (int i = 0; i < n; ++i)
            for (int j = 0; j < i; ++j)
                  __①__
      for (int i = 0; i < n; ++i){
            for (int j = 0; j < n; ++j){
                  c[i][j] = 0.0;
                  sum = _mm_setzero_ps();
                  for (int k = n - 4; k >= 0; k -= 4){          // mul and sum every 4 elements
                        __②__
                        t2 = _mm_loadu_ps(b[j] + k);
                        t1 = _mm_mul_ps(t1, t2);
                        __③__
                  }
                  sum = _mm_hadd_ps(sum, sum);
                  __④__
                  _mm_store_ss(c[i] + j, sum);
                       for (int k = (n % 4) - 1; k >= 0; --k){        // handle the last n%4 elements
                        __⑤__
                  }
            }
      }
      for (int i = 0; i < n; ++i) for (int j = 0; j < i; ++j)  __①__
}
```

2. 补全下面程序代码，程序是用 Pthread 实现子线程的线程号等字符串的输出。（10 分）

①:

②:

③:

④:

⑤:

```
#include <stdio.h>
#include <stdlib.h>
#__①__

/* Global variable: accessible to all threads */
int thread_count;
pthread_mutex_init(&mutex, NULL);
void* Hello(void* rank); /* Thread function */

int main(int argc, char* argv[]) {
   long thread; /* Use long in case of a 64-bit system */
   pthread_t* thread_handles;

   /* Get number of threads from command line */
   thread_count = strtol(argv[1], NULL, 10);
   thread_handles = malloc(thread_count*sizeof(pthread_t));
   for (__②__) /*Create threads*/
      __③__;
   printf("Hello from the main thread\n");
   for (thread = 0; thread < thread_count; thread++)
      __④__
   free(thread_handles);
   return 0;
} /* main */
void* Hello(void* rank) {
   long my_rank = (long) rank;    /* Use long in case of 64-bit system */
   __⑤__;
   printf("Hello from thread %ld of %d\n", my_rank, thread_count);
   pthread_mutex_unlock(&mutex);
   return NULL;
} /* Hello */
```

3. 补全下面程序代码，程序是用 OpenMP 实现子线程的线程号等字符串的输出。（6 分）

草 稿 区 区

```
#include <stdio.h>
#include <stdlib.h>
#__①__
#include <omp.h>
#endif

void Hello(void); /* Thread function */

int main(int argc, char* argv[]) {
   /* Get number of threads from command line */
   int thread_count = strtol(argv[1], NULL, 10);

# pragma omp __②__
   Hello();

   return 0;
}   /* main */
void Hello(void) {
   #__①__
   int my_rank = omp_get_thread_num();
   int thread_count = __③__
   # else
      int my_rank = 0;
      int thread_count = 1;
   # endif
   printf("Hello from thread %d of %d\n", my_rank, thread_count);

}   /* Hello */
```

①:

②:

③:

4. 补全下面程序代码，程序是用 MPI 编程实现 2 个进程的并行排序的代码。（6 分）

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char ** argv)
{
   int rank, a[1000], b[500];

   MPI_Init(&argc, &argv);
   __①__;;
   if (rank == 0) {
      MPI_Send(&a[500], 500, MPI_INT, 1, 0, MPI_COMM_WORLD);
      sort(a, 500);
      MPI_Recv(b, 500, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   /* Serial: Merge array b and sorted part of array a */
   }
   else if (rank == 1) {
      __②__
      sort(b, 500);
      __②__
   }
   MPI_Finalize();
   return 0;
}
```

①：

②：

③：