软件学院本科生 **2019—2020** 学年第二学期《并行与分布式程序设计》 课程期末考试试卷（A 卷）

专业： **年级：** **学号：** **姓名：** **成绩:**

草 稿 区

得 分

一、单选及填空（本题共 25 分，选择 1-11 题每题 2 分，12、13 题每空 1 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. 2018 年 7 月 22 日，我国“（ ）E 级原型机系统”已在国家超级计算天津中心完成研制部署，并顺利通过项目课题验收，将逐步进入开放应用阶段。预示着中国 E 级计算机将很快进入实质性研发阶段，该计算机正式发布后有可能助力中国超算机重新登顶 Top500 榜单。

   A. 天河二号 B. 天河三号 C. 神威・太湖之光 D. Summit

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. 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

4. 一个函数或库是“线程安全的”，意味着其（ ）。

   A. 多线程执行结果能正确执行 B.多线程执行能保护用户隐私数据

   C. 多线程执行能抵御网络攻击 D.以上皆错

5. 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

6. 静态任务划分相对于动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

草 稿 区

7. 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 dynamic 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

8. 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

9. Pthread 程序和 OpenMP 程序中线程获得自身编号的方式分别是（ ）。

   A.两者均为创建线程时传递参数　　B.前者创建线程时传递参数，后者通过特定 API

   C.两者均通过特定 API　　D.前者通过特定 API，后者创建线程时传递参数

10. 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

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___。

得 分

二、多项选择题（本题共 15 分，每小题 3 分，请把本题答案按题号汇总到表格中）

1	2	3	4	5

1. 以下选项关于阿姆达尔定律描述正确的是（ ）。

   A.只要一个串行程序中存在不可并行化的部分，它通过并行化的加速比就是受限的

   B.串行程序中如果有比例为 r 的部分不可并行化，则根据该定律，加速比就是 1/r

   C.该定律描述了一种加速比存在上限的情况，现实中并行化往往达不到该上限

   D.只要并行的核数目足够多、性能足够强，并行化加速比是可以无限增大的

2. Pthread 编程中可以用来实现路障的方式有（ ）。

   A.忙等待　　B.条件变量　　C. 信号量　　D.读写锁

3. SIMD 向量计算编程中，主要会遇到哪些额外开销 （ ）。

   A. 打包/解包开销　　B. 数据传输开销　　C. 对齐开销　　D. 控制流导致额外开销

4. 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

5. 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

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. 并行计算还主要用于国防、航天、工程研究等领域。（ ）
3. 并行计算领域，近些年硬件技术飞速发展，相比之下，软件生态环境发展缓慢。（ ）
4. 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。（ ）
5. 指令级并行让多个处理器部件或功能单元串行执行指令来提高计算性能。（ ）
6. 单指令多数据流（SIMD）的一种广泛应用是向量处理器，此外，GPU 也使用 SIMD 并行方式。（ ）
7. OpenMP 编译指示的作用范围只有其后一个语句。（ ）
8. 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。（ ）
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. 请简述推动并行计算发展的因素。（4 分）

草　稿　区

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)数据依赖：

(3)互斥量：

3. 请简述 Flynn 分类法对计算机系统的分类。 (4 分)

4.MPI 编程是一个消息传递接口函数库，提供了基础的点对点的通信接口，如 MPI_Send、MPI_Recv，此外还提供了多种组通信的方式。请写出不少于 6 个组通信原语，并分别简述其功能。(6 分)

得 分

五、按要求写出相应代码（本题共 30 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
                        t1 = _mm_loadu_ps(a[i] + k);
                        ②
                        ③
                        sum = _mm_add_ps(sum, t1);
                  }
                  ④
                  sum = _mm_hadd_ps(sum, sum);
                  _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
}
```

草 稿 区

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）

草　稿　区

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）

```
#include <stdio.h>
#include <stdlib.h>
##ifdef _OPENMP
#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  ①
   Hello();

   return 0;
}   /* main */
void Hello(void) {
   #  ②
   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 编程实现不同进程的进程号等字符串的输出。（4 分）

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]){
   int myid, numprocs;
   int namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   ①
   MPI_Get_processor_name(processor_name,&namelen);
   fprintf(stderr,"Hello World! Process %d of %d on %s\n",myid, numprocs, processor_name);
   ②
  return 0;
}
```

①:

②:

草 稿 区