并行期末复习汇总

绪论

发展并行计算的原因/推动并行计算的因素

频率已经不是处理器发展的主角

功耗/散热的限制

性能上升放缓

多核、众核成为之后CPU的发展趋势

并行计算软件技术面临的挑战

并行程序设计的复杂性

数据移动（通信）代价很高

能耗挑战

伸缩性挑战

软件生态环境几乎停滞

并行计算的应用

气候模拟、蛋白质折叠、药物发现、能源研究、数据分析

科学仿真、全局气候建模、海洋建模、星系演化、生物信息学、天文学、强子对撞机

商用领域：运算能力更强、速度更快、价格更便宜

国防、超算

中国著名的超级计算机

2016年 无锡 神威·太湖之光 十亿亿次/秒

2018年 天津 天河三号 百亿亿次/秒

并行硬件和并行软件

冯诺依曼瓶颈的定义及原因

定义：CPU和主存之间的分离称为冯诺依曼瓶颈

原因：互连结构限制了指令和数据访问的速率。一条指令的执行过程中，CPU从主存中存取的过程耗时较大

cache相关概念

命中：在多级缓存中找到了相应的数据

缺失：没找到

cache一致性

写直达：一旦发生不一致，立即相应更改主存中的数据

写回：将缓存中不一致的数据标记为脏数据，待整个高速缓存行里的数据都执行完毕以后，再一次性写入主存中。

共享内存系统缓存一致性

监听协议：一个核更新副本x后在总线上传播，其他核包含x的整个cache行更新

目录协议：通过目录存储每个内存行的状态，当一个变量需要更新时，查询目录，将所有包含该变量的高速缓存行置为非法。

指令级并行

流水线：将功能单元分段安排

多发射：让多条指令同时启动

并行多线程相关概念

进程：运行着的计算机实体

多任务：单个处理器同时运行多个任务

线程：一个进程被划分成一些相互独立的任务，每个独立任务称作线程

Flynn分类法

SISD：单指令单数据流（冯诺依曼）

SIMD：单指令多数据流 对多个数据同时执行相同的命令，一个控制单元对应多个计算单元

MISD：多指令单数据流 多个控制单元对应单个计算单元

MIMD：多指令多数据流

SIMD系统有哪些

向量处理器

GPU

MIMD系统有哪些

共享内存系统

分布式内存系统

互连网络

共享式互连网络：总线（低成本、灵活，但速度慢）、交叉开关矩阵（比总线速度快，但设计复杂、灵活性差、成本高）

分布式内存网络（**要背各个类型的等分宽度！**）：

直接互连☆（每个交换器与一个处理器-内存对直接相连）：

等分宽度可以理解为去除最少的链路数，从而将节点分成两等份。

正方形二维环面：$p=q^2$个节点，等分宽度为2q=**2*根号p**

全相连网络：$p^2/4$

超立方体：p/2

间接互连（交换器不一定与处理器直接相连）

延迟和带宽

信息传输时间=l+n/b

l:延迟：发送-开始接收

n:数据长度（字节）

b：带宽：接收速度

并行算法设计相关概念

原子性：一组操作要么全部执行，要么全不执行。

临界区：一个更新共享资源的**代码段**，一次只能允许一个线程执行该代码段

竞争条件：多个进程/线程尝试**更新同一个共享资源**时，结果可能是无法预测的，则存在竞争条件。

数据依赖（data dependence）：两个**内存操作的序**，为了保证结果的正确性，必须保持这个序。

同步（synchronization）：在时间上强制使执各行进程/线程在某一点等待，确保各进程/线程的正常顺序和对共享可写数据的正确访问。

互斥：任何时刻都只有一个线程在执行

互斥量：互斥锁的简称，用来限制每次只有一个线程能进入临界区。保证了一个线程独享临界区。

障碍：阻碍线程继续执行，在此程序点等待，直到所有参与线程都达到障碍点才能继续执行

性能评价标准

加速比

S=T串/T并

这里注意T串是串行**最优算法**的运行时间

效率

E=S/p=T串/T并*p

阿姆达尔定律

除非一个串行程序的执行几乎全部并行化，否则无论多少可利用的核，通过并行化产生的加速比都会是受限的。

最大加速比S=1/1-a

a:可并行化比例

只要有不可并行的部分就永远无法达到

可扩展性

程序核数（线程数）增加，输入规模也以相应增长率增加的情况下，程序效率不变

强可扩展性：问题规模不变时，效率不随核数的增大而降低

弱可扩展性：问题规模需要以一定的速率增大，效率才会不随核数的增加而降低

SIMD

SIMD编程的额外开销

打包/解包数据的开销

对齐开销

控制流导致的额外开销

sse指令

格式：_mm_load_ps

__m128: float

__m128d: double

__m128i: integer

load：从内存加载到寄存器 loadu:不要求16字节对齐 loadl/h：存储到寄存器低/高位

store：从寄存器加载到内存

setzero:清零

add mul hadd: 横向计算

pd：两个双精度 ps：四个单精度 ss：标量

矩阵乘法代码示例

```
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)
        swap(b[i][j], b[j][i]); //矩阵转置
    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){ // sum every 4 elements
                t1 = _mm_loadu_ps(a[i] + k);
                t2 = _mm_loadu_ps(b[j] + k);
                t1 = _mm_mul_ps(t1, t2);
                sum = _mm_add_ps(sum, t1);
            }
            sum = _mm_hadd_ps(sum, sum); //两次横向计算，将四个结果都加到sum
            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
                c[i][j] += a[i][k] * b[j][k];
            }
        }
    }
    for (int i = 0; i < n; ++i) for (int j = 0; j < i; ++j)
        swap(b[i][j], b[j][i]);
}
```

Pthread

特性

可移植但比较慢、可同步

相对低层：

程序员控制线程管理和协调

程序员分辨并行任务并管理任务调度

适用于系统代码开发

支持：

创建并发执行

同步

非显式通信

并行程序设计的复杂性

足够的并发度（Amdahl定律）

并发粒度

局部性

负载均衡

协调和同步

线程安全/同步

（客观题）

对于一个函数或库，若多线程并行调用能“正确”执行，则称它是线程安全的

pthread基础API

phread

pthread_create(&thread，线程属性（NULL)，执行函数，函数参数);

pthread_join(&thread，NULL);

pthread_exit(&thread);

pthread_cancel(thread);

mutex

pthread_mutex_t amutex;

pthread_mutex_init(&amutex, NULL);

pthread_mutex_lock(&amutex);

pthread_mutex_unlock(&amutex);

pthread_mutex_destroy(&amutex);

semaphore

#include<semaphore.h>

sem_t sem;

sem_init(&sem, 0, origin);

sem_wait(&sem);

sem_post(&sem);

sem_destroy(&sem);

barrier

pthread_barrier_t b;

pthread_barrier_init(&b, NULL, num_threads);

pthread_barrier_wait(&b);

pthread_barrier_destroy(&b);

*条件变量(考的概率小)

pthread_cond_t condition;

pthread_cond_init(&condition, NULL);

pthread_cond_signal(&condition); //唤醒在条件变量上等待的线程

pthread_cond_wait(&condition, &mutex); //unlock，阻塞，lock 因此在执行此条语句之前必须在同线程上锁

pthread_cond_destroy(&condition);

pthread编程示例

helloworld

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* Global variable: accessible to all threads */
int thread_count;
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 = (pthread_t*) malloc(thread_count*sizeof(pthread_t));
    for (thread = 0; thread < thread_count; thread++)
        pthread_create(&thread_handles[thread], NULL, Hello, (void*) thread); //
创建线程
    printf("Hello from the main thread\n");
    for (thread = 0; thread < thread_count; thread++)
        pthread_join(thread_handles[thread], NULL); //等待子线程运行结束
    free(thread_handles);
    return 0;
} /* main */
void* Hello(void* rank) {
    long my_rank = (long) rank; /* Use long in case of 64-bit system */
    printf("Hello from thread %ld of %d\n", my_rank, thread_count);
    return NULL;
} /* Hello */
```

同步的几种方式

忙等待+互斥

信号量

障碍

条件变量

OpenMP

特性

可移植、同步、可扩展

相对高层：

程序员指出并行方式和线程特性，并指导任务调度

系统负责实际的并行分解和调度管理

适用于共享内存架构上的科学计算

支持：隐藏栈式管理

不能：自动化并行、确保加速、避免数据竞争

openmp基础API

openmp

#ifdef _OPENMP

int size = omp_get_num_threads();

int rank = omp_get_thread_num();

#else

int rank = 0;

int size = 1;

#endif

pragma omp parallel num_threads(size);

critical

是一个显式同步

#pragma omp critical

for+reduction

#pragma omp parallel **for** num_threads(size) \

reduction(+:approx)

for使用的限制条件以及目的

1.并非所有“元素级”循环都能并行化

a. 循环变量应为带符号整数

b. 终止检测只能为<,<=,>,>=以及循环不变量

c. 每步迭代递增/递减一个循环不变量

d. 对<,<=向上计数；对>,>=向下计数

e. 循环体无进出控制流

2.创建线程，在线程间分配循环步骤要求迭代次数可预测

a. 不检查依赖性

b. 不支持while/do-while

循环调度方式

static

dynamic：chunk缺省为1

guided ： Sk=Rk/2N chunk:最小划分大小

MPI

MPI基本原语

#include<mpi.h>

MPI_Init();

MPI_Comm_size(MPI_COMM_WORLD, &size);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

MPI_Finalize();

MPI_Send(message, num, type, dest, tag, MPI_COMM_WORLD);

MPI_Recv(message, num, type, resource, tag, MPI_COMM_WORLD, &status/MPI_STATUS_IGNORE);

主从通信

这次考试还考了主从通信的一些客观题，我没有复习到，大家自己整理吧。

组通信

功能：通信、同步和计算

MPI编程的组通信原语（至少6个）

MPI_Bcast 广播

MPI_Reduce 归约

MPI_Allreduce 每个进程都作为ROOT执行一次相同的归约操作

MPI_Scatter 数据散发 源节点向其他每个节点发送不同数据

MPI_Reduce_scatter 归约后再执行一次散发

MPI_Gather 从每个进程收集一次数据

MPI_Allgather 组内每个进程都进行一次收集 相当于All-to-All广播

MPI_Scan 每个进程都对排在它前面的进程进行归约操作

MPI_Barrier 只有当所有语句都执行了该调用后才一起向下执行