系统结构

题型

选择题 10

判断 15

简答 25 共7道

上面都是送的 全都是书上的概念

应用 10 3道小计算

1. 流水线
2. 互联网络函数
3. 交叉开关

编程 15 一道大题 OpenMP

MPI基本函数要知道

综合题 25 共3道 考验研讨的东西? 书上东西的拓展

从存储角度如何提高速度? 从流水线角度如何提高速度? GPU如何提高并行性?

OpenMP和MPI

OpenMP 线程级别 多核共享存储

MPI 进程之间消息传递 多CPU之间

MPI

cumm_size函数什么意思 启动、获取进程数、获取进程号

MPI 多CPU的进程之间的消息传递

OpenMP

听起来像是从0开始写

编译指导语句 for 私有变量? 公有变量? fork-join过程要知道 critical 还是reduction 规约? 看一看PI头文件也要写 inlcude后面的不能少

编译命令、运行命令都要有 再来一点性能的实验分析 对于负载的一些讨论

局部性在哪里 开销在哪里

文件打开、文件读取 二进制文件的打开函数也要会写 CPP本身的功底

第一章 计算机系统结构导论

1. 计算机系统结构的定义 p8 方+雪

- 计算机系统结构是指对机器语言计算机的软、硬件功能分配和界面的定义。

2. 计算机系统结构的组成和实现的区别，以及他们的关系 *p9 p10* 方+雪 **可能考简答**

- 计算机系统结构是**计算机系统**的**概念性结构**和**功能特性**。
- 计算机组成是**计算机系统结构**的**逻辑实现**。
- 计算机实现是**计算机组成**的**物理实现**。
- 总而言之，系统结构、组成和实现之间的关系应符合下列原则：系统结构设计不要对组成、实现有过多和不合理限制；组成世纪应在系统结构指导下，以目前能实现的技术为基础；实现应该在组成的逻辑结构下，以目前器件技术为基础，以优化性能价格比为目标。

3. 模拟和仿真的概念 *p11* 方+雪

- 肯定是不兼容的才需要模拟和仿真。 **可能考判断**
- 用**机器语言程序**解释实现程序的移植方法称为模拟。
- 用**微程序**直接解释另一种机器的指令系统称为仿真。
- 仿真与模拟的主要区别在于解释用的语言。仿真使用**微程序**解释，器解释程序在微程序存储器；模拟是用**机器语言**程序解释，器解释程序在主存储器。

4. 软件和硬件的关系 *p12 + p14*
5. 从中间开始的设计方法 *p15*

判断题：计算机设计的方法从中间开始是比较合适的，从层次结构的软银见的洁面开始，合理的进行软件功能分配，合理的进行软件监控。✅

6. *Flynn*分类法 *p16* 方+雪

简答题：举出*Flynn*发的四种计算机系统结构，每种举出一个例子。

- *SISD*
    - 串行处理机
- *SIMD*
    - *CUDA* 阵列处理机 并行处理机 *MMX AVX SVE*
- *MISD*
- *MIMD*
    - 多处理机 多计算机

7. 透明性定义 *p8*

- 所谓“透明性”，一是指确实存在，二是指无法检测和设置。
- 在计算机系统中，低层的概念性结构和功能特性对高层来说是透明的。

8. 兼容性 *p11* 方+雪

- 系列机**系统软件**必须兼容，系列机软件兼容指同一个软件（目标程序）可以不加修改地运行于系统结构相同的各机器上，并且所得结果一致。
- 向上兼容：抵挡机器的目标程序不加修改可以运行与高档程序。 向前（后）兼容：系列机投放市场先后实现软件兼容。
- 一般采用，**向上兼容，向后兼容**。 一定是向后哈 **可能考判断，并且给出向前兼容的选项，这个时候选** ❌

9. 计算机是软件、硬件组成的 ❌ **为什么？？**

10. SIMT与SFMD的区别和联系 p18 **可能考简答题** 方+雪

- cx说GPU是SIMT lff说GPU是SIMD
- SIMT线程级别并行
- SIMD数据之间的并行
- 在SIMD上有很多处理单元做向量计算，想让他提高效率，加速，只能增加硬件或者处理机。
- **在SIMT上想提高效率，提高并行度，可以增加什么呢?**
- SIMT和SIMD之间有些许不同

为了在SIMD结构上提高效率和并行度，可以采取以下几个措施：

1. 扩展指令集：采用更为丰富的指令集，包括更多的运算符和特定的指令集，能够提升数据处理效率和运算速度。一般而言，增加指令集是最为直接有效的策略。

2. 使用更广的数据类型：通过扩大数据类型的位数，来处理更多的数据位，从而进一步提高并行度和运算效率。如使用32位和64位浮点数来处理更复杂的计算。

3. 针对特定应用进行优化：对于一些特定的应用，可以进行一些优化，例如针对矩阵运算的优化和改进，并通过SIMD指令集来实现矩阵乘法等操作，实现更快的速度和更高的性能。

4. 通过循环展开等技巧来充分利用SIMD: 在代码编写过程中，可以采用循环展开等技术，将多条数据处理指令合并成一个向量指令，充分利用SIMD结构。这样能够实现更高的效率和并行性。

5. 优化存储器访问: 为了避免由于数据访问延迟和带宽限制而导致明显的性能下降，可以采用数据重用等技术，减少数据访问次数。

以上是提高SIMD效率和并行度的几个方法，需要根据具体的应用场景和实际情况选择适当的方案。同时，也需要进行综合考虑，找到最终是否能够实现对性能瓶颈的处理，同时是否会带来更大的成本。

为了在SIMT结构上提高效率和并行度，可以采取以下几个措施：

1. 增加核心数量：增加核心数量可以扩大并行处理的规模，进而提高并行度。不过，这一策略实行起来比较困难，因为增加核心数量需要更大的面积和更大的功耗，同时还需要更多的存储器和带宽来满足运行需求。

2. 改善内存访问性能：由于SIMT架构的特殊性质，内存访问往往是最关键的性能瓶颈，可以采用更高效的内存访问机制（如缓存、预取、指令调度等）来改善内存访问性能。此外，使用更高带宽的存储器和总线来提高内存传输速率也是一种解决方案。

3. 优化指令调度：SIMT结构中每个核的指令都是按照一定的顺序执行，如果采用更高效的指令调度机制，就可以提高SIMT的并行度。可以使用更优化的编译器或者手动进行指令重排等操作来实现指令调度优化。

4. 采用更优化的算法和数据结构：优化算法和数据结构可以避免重复计算和数据依赖等问题，从而提高并行效率和减少数据冲突。在SIMT中，可以通过使用更高效的算法和数据结构来将任务分解成更小的部分，实现更高的并行度和更高的效率。

以上是提高SIMT的效率和并行度的几个方法，需要根据具体的应用场景和实际情况选择适当的方案。

SIMD和SIMT是两种不同的并行计算模型，其架构和应用场景不同，所以在提高效率和提高并行度时需要采取不同的措施。

SIMD适用于相同类型的数据集合上进行并行计算，采用向量指令实现操作。对于SIMD架构，可以通过扩展指令集、使用更广的数据类型、优化存储器访问等方法来提高效率和并行度。

SIMT则是通过并行度提高计算效率，针对不同计算单元上执行不同的程序指令。SIMT架构可以通过增加核心数量、改善内存访问性能、优化指令调度等方法来提高效率和并行度。

因此，虽然SIMD和SIMT都是用于并行计算的架构，但它们的并行计算模型和应用场景不同，需要在实际应用时针对其特点采取不同的措施来提高效率和并行度。

11. 虚拟计算机的层次结构 p11

第二章 处理器及其相关技术

1. 缩短地址码的方法 → 哈夫曼编码 p27 **可能考填空**
2. 指令系统的设计要求 **可能考填空** 方+雪

规整性、高效性、兼容性 3个

3. CUDA编程基于**SIMD**编程模型，在GPU上执行的代码被称为**核函数** p48 方+雪
4. GPU相关概念 p47 **可能考简答**

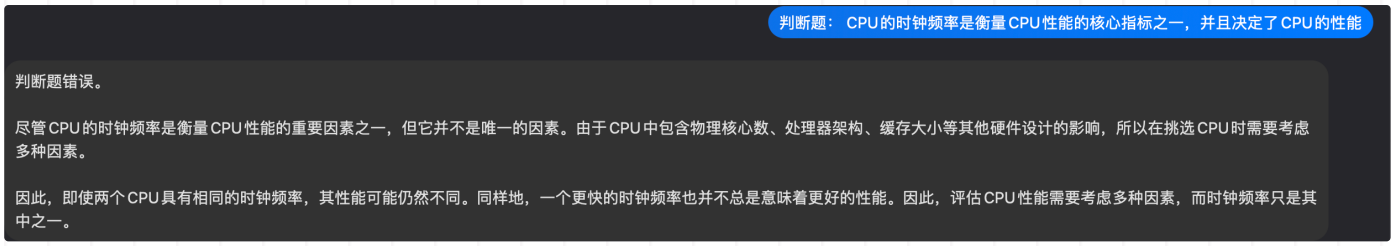
*GPU*是***SIMD***类型的系统结构，在*GPU*中基本的运算单元被称为**流处理器**。多个流处理器、缓存和指令控制单元组成**流多处理器**。*SM*是*GPU*执行指令的基本单位，在同一流多处理器内的流处理器具有**共享**的高速一级**缓存**。整个*GPU*则有多组流多处理器、共享二级缓存和显存控制器等组成。

5. 二八定律 *p29* **可能考判断，并且答案为** ✅

一个典型程序的运算过程所使用的*80%*指令只占处理器指令系统所有指令中的*20%*，事实上最频繁使用的指令是取数据、存数据和加法等最简单的指令。

6. 时钟频率 *p31* **可能考判断，并且答案为** ❌

*CPU*的时钟频率是衡量*CPU*性能的核心指标之一，并且决定了*CPU*的性能 ❌



7. *GPU*一定比*CPU*快 ❌

第三章 存储系统结构

1. 给你不同的存储介质，让你分析怎样去组织可以获得更好的性能。 **可能考简答**

物理材质本身

*cache*想要性能提升，需要怎么做？虚拟存储器想要性能变强的话，需要怎么做？

2. *cache*是在特殊位置使用的？ ❌ **可能考判断** 方+雪

*cache*以前只有单一的*cache*，现在可以做*1*级、*2*级、*3*级等多级，有的是做在*CPU*内，有的做在板上？不一定只是在特殊位置来使用的。 （没懂啥意思，这是*lff*原话）

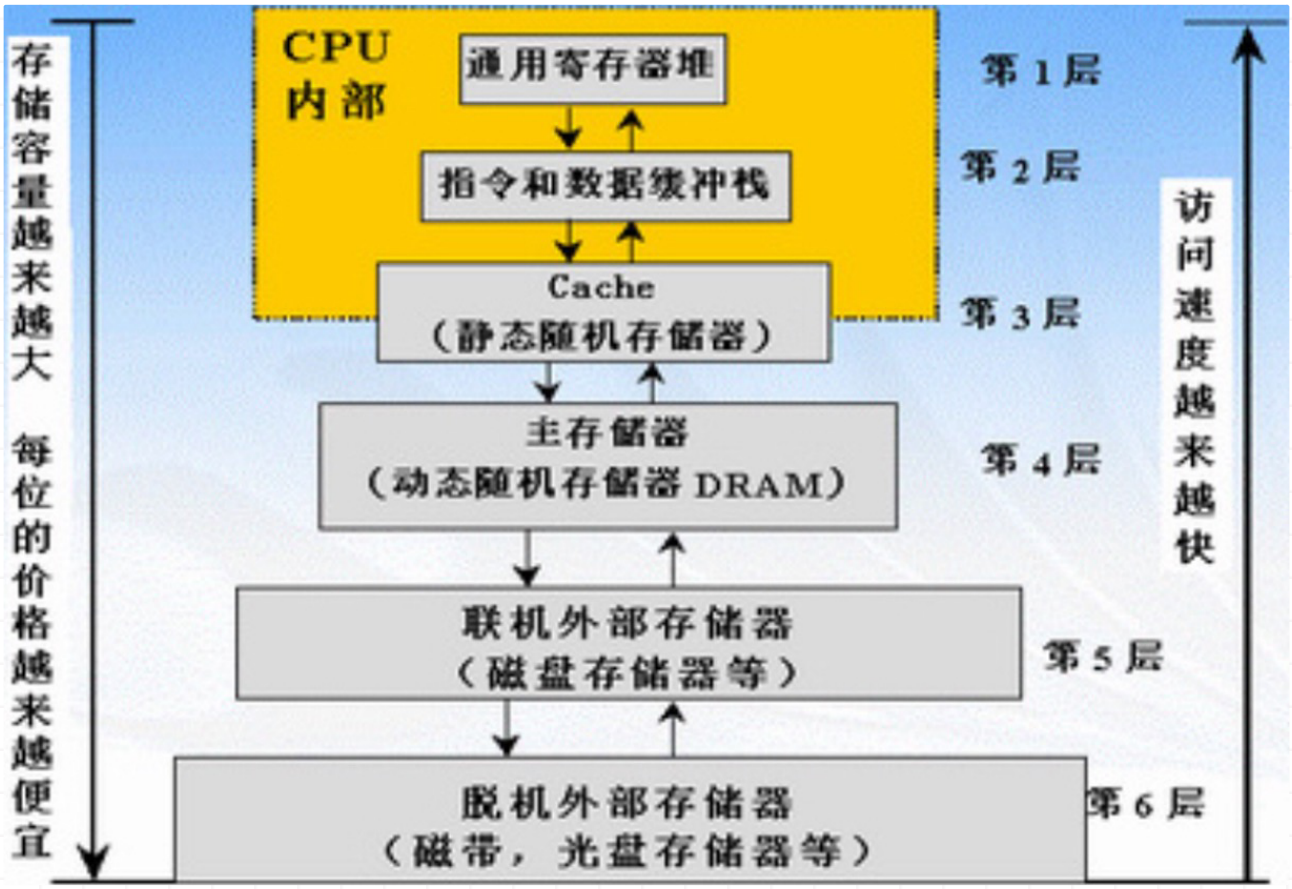
3. 置换算法的堆栈 *p72-p74* **可能考简答** 方+雪

对于置换算法进行分类，具有堆栈型特点的置换算法，我们认为是好的。*FIFO*、*LRU*、*OPT*中，哪些是堆栈型？原因是什么？

4. *cache*透明性（一致性） *p87* **一定考简答** 方+雪

*cache*不一致的原因是什么？写回法和写直达法。是什么？各自的优缺点？针对单处理器的*cache*，不讨论多处理器。**写回法中，*CPU*中的数据只写入*cache*，不写入主存。***cx***说这句话很重要。

5. 存储系统的层次结构？为什么存储结构可以提高速度？**可能考简答**



- 提高*cache*命中率？如何提高命中率？**预取**，把内存的一大堆东西一个块，提前取到。程序和数据都有局部性原理。为啥存储系统比单个的存储器强？
- 并行。平铺提高速度的方法，低位交叉。流水线方式？**低位交叉存储体是采用流水线方式工作的并行存储系统。**

第四章 流水线结构」

1. 什么是先行控制结构？ *p120 p122* **可能考简答** 方+雪

把处理机中原来一个集中的控制器，分解为存储控制器（存控）、指令控制器（指控）、运算控制器（运控）三个部分。

先行控制技术实际上是**缓冲技术**和**预处理技术**相结合的产物。

图*4-9*

2. 流水线原理 *p123 4.1.3* **可能考简答**

流水线的特点，或者说是实现流水线的方式。

3. 加速比的计算 *p122* 方+雪

4. 非线性流水线的调度 预约表等 *p130* **计算题** 方+雪

5. 超级流水处理机性能比较 *p140* 表 *4-1* **可能考填空** 方+雪

6. 超级流水处理机的评价指标 *p143* 方+雪

记住式子 *S(m,1) S(1,n) S(m,n)*

7. 数据相关 *p148* 第*4*题这一类 方+雪

写写相关、写读相关、读写相关 补考调度

第五章 并行处理机与多处理机系统

1. 提高并行性的措施 p151 方+雪

- **时间重叠、资源重复、资源共享**
- 普通流水线为时间重叠
- 超标量既有时间重叠，又有资源重复 空间换时间
- 超流水线 时间换空间
- 并行处理机 资源重复，并非时间重叠

2. ILLIAC 是阵列机 是SIMD 是资源重复，并非时间重叠 p153 **可能考判断题** 方+雪
3. 并行处理机（SIMD计算机）的特点、原理 p151 p160 **可能考简答题** 方+雪

分为两种，一种是CPU、一种是GPU 两者对比？ ff圣经2，27分钟的时候说的

4. 混连函数 p163 **计算题** 方+雪

混洗、方体、PM2I的表达式子

5. 动态互联网络 p174 **计算题**

要不是多个cube 要不是多个shuffle 一定是 2功能的控制信号 看清楚标1交换还是标0交换

5. 并行性的概念 p150

   同时性 和 并发性

   判断题： 普通的一条流水线，只有并发性，没有同时性。 ✅
6. 并行性的执行角度和处理数据的角度 p150 方+雪
7. 互联网络特性 cx说不用看 p165
8. 多处理机的概念 p182

优缺点?

第六章 集群、网格和云计算

> 方方直接没有说这章的内容，🥹

1. 共享存储和分布式存储 **可能考判断**

集群的性能只由机器cpu决定。 ❌ 和通信网络有关，和上层的硬件调度有关。

2. MPI

P221 前六个函数的各种参数要搞清楚。

编程题

OPENMP

- 流文件或者二进制文件之中读取数据，读完之后要打开、关闭文件。

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    string filename = "data.txt";

    ifstream fin;
    fin.open(filename); // 打开文件

    string line;
    while (getline(fin, line)) {
        cout << line << endl; // 输出到终端
    }

    fin.close(); // 关闭文件

    ofstream fout;
    fout.open("output.txt"); // 打开

    fout << "This is added text." << endl; // 向文件中输出

    fout.close(); // 关闭文件

    return 0;
}
```

- 会告诉当前的机器配置是什么，比如有几个核，有几个核就用几个核
- include哪些头文件？ 头文件好像共6个，共3分，错1个扣1分，扣完为止。

```cpp
#include<fstream>
#include<iostream>
#include<omp.h>
#include<time.h>

omp_get_therad_num(); // 返回线程号
omp_set_num_threads(); // 设置后续并行域中的线程个数
omp_get_num_threads(); // 返回当前并行域中的线程数
omp_get_num_procs(); // 返回系统中处理器的个数
```

```cpp
// 用reduction
#include <stdio.h>
#include <omp.h>

int main() {
    int a[] = {1, 2, 3, 4, 5};
    int n = sizeof(a) / sizeof(a[0]);
    int product = 1;

    omp_set_num_threads(16); // 设置线程数量为16
#pragma omp parallel for reduction(*:product)
    for (int i = 0; i < n; i++) {
```

```
        product *= a[i];
    }

    printf("the product is: %d\n", product);
    return 0;
}
```

```
// 不用reduction
#include <stdio.h>
#include <omp.h>

int main() {
    int a[] = {1, 2, 3, 4, 5};
    int n = sizeof(a) / sizeof(a[0]);
    int product = 1;

    omp_set_num_threads(16); // 设置线程数量为16

#pragma omp parallel
{
    int local_product = 1;
#pragma omp for
    for (int i = 0; i < n; i++) {
        local_product *= a[i];
    }
#pragma omp critical
    {
        product *= local_product;
    }
}

    printf("the product is: %d\n", product);
    return 0;
}
```

- 写出编译命令和运行命令

```
g++ -fopenmp main.cpp -o main -lomp # 编译
./main # 运行
```

- 私有变量和公有变量
- 描述*fork-join*的过程?
- 已经完全的利用算力的情况下，如何提高速度?

负载均衡。负载均衡对并行程序的性能至关重要，因为在并行运算中，有些线程可能会比其他线程运行的更快或者更慢，如果不将任务正确地分配给每个线程，会导致有些线程空闲，而其他线程处于过载状态，并且整个并行程序得不到最优的性能。

动态负载均衡。 `#pragma omp parallel for schedule(dynamic, chunk_size)` ，其中，*chunk_size*是每个线程从任务池中获取任务的块大小。将任务块放入任务池中，每个线程获取池中的一块任务并执行。执行完成后，线程将结果返回池中并继续获取下一个任务块。通过这种方式，任务块可以动态分配给空闲线程，以达到更好的负载均衡效果。

扩展指令集?