计算机体系结构
实验三 存储层次分析及程序优化

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

实验目的

- 加深对Cache基本结构及Cache替换策略的理解
- 掌握使用程序测量Cache结构、替换算法、写策略的基本方法
- 掌握利用Cache结构和局部性原理对程序进行优化的基本方法

实验内容

1. 使用C/C++编写程序测量所使用机器的Cache结构参数：

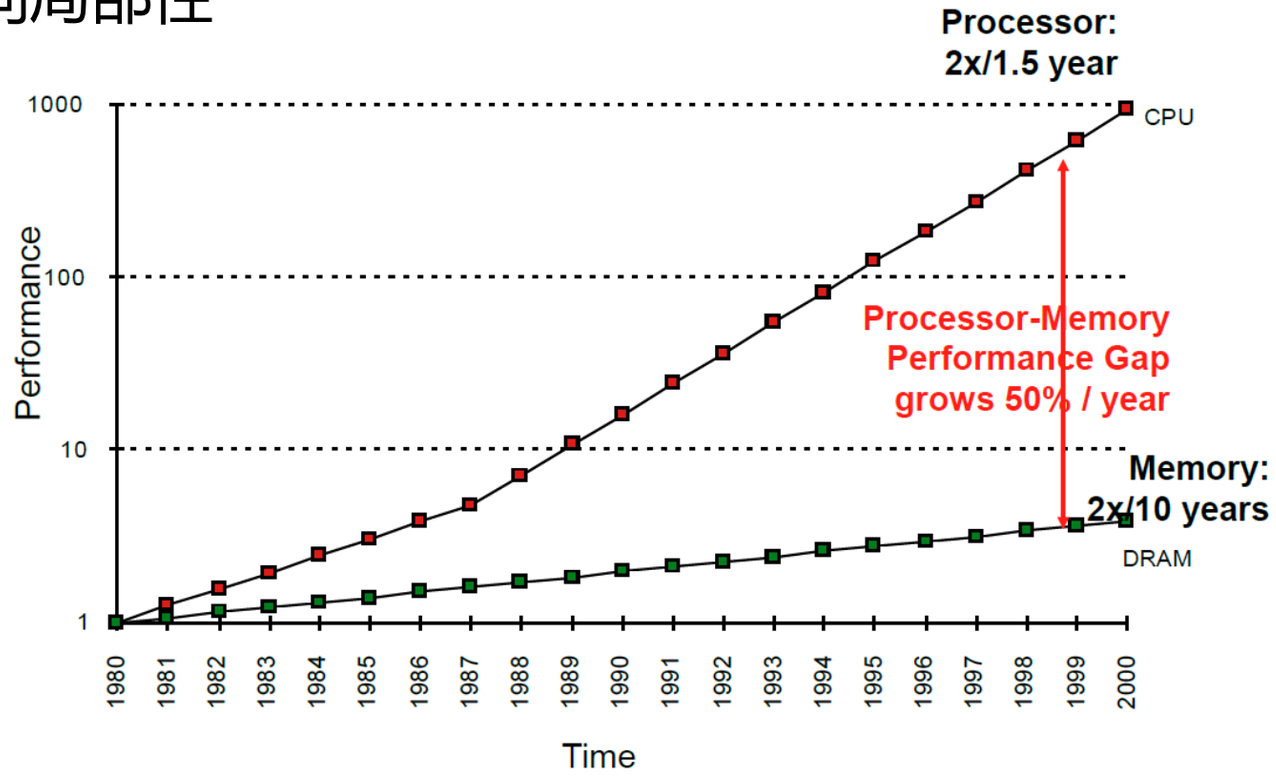
   测量目标机的L1 DCache与L2 Cache的结构

   - 测量各级Cache的容量、块大小、相联度
   - 测量Cache的写策略（写回法？写直达法？）
   - 测量Cache替换策略是否采用LRU算法，或发掘其他有意义的Cache参数

2. 根据机器的Cache结构，优化矩阵乘法算法

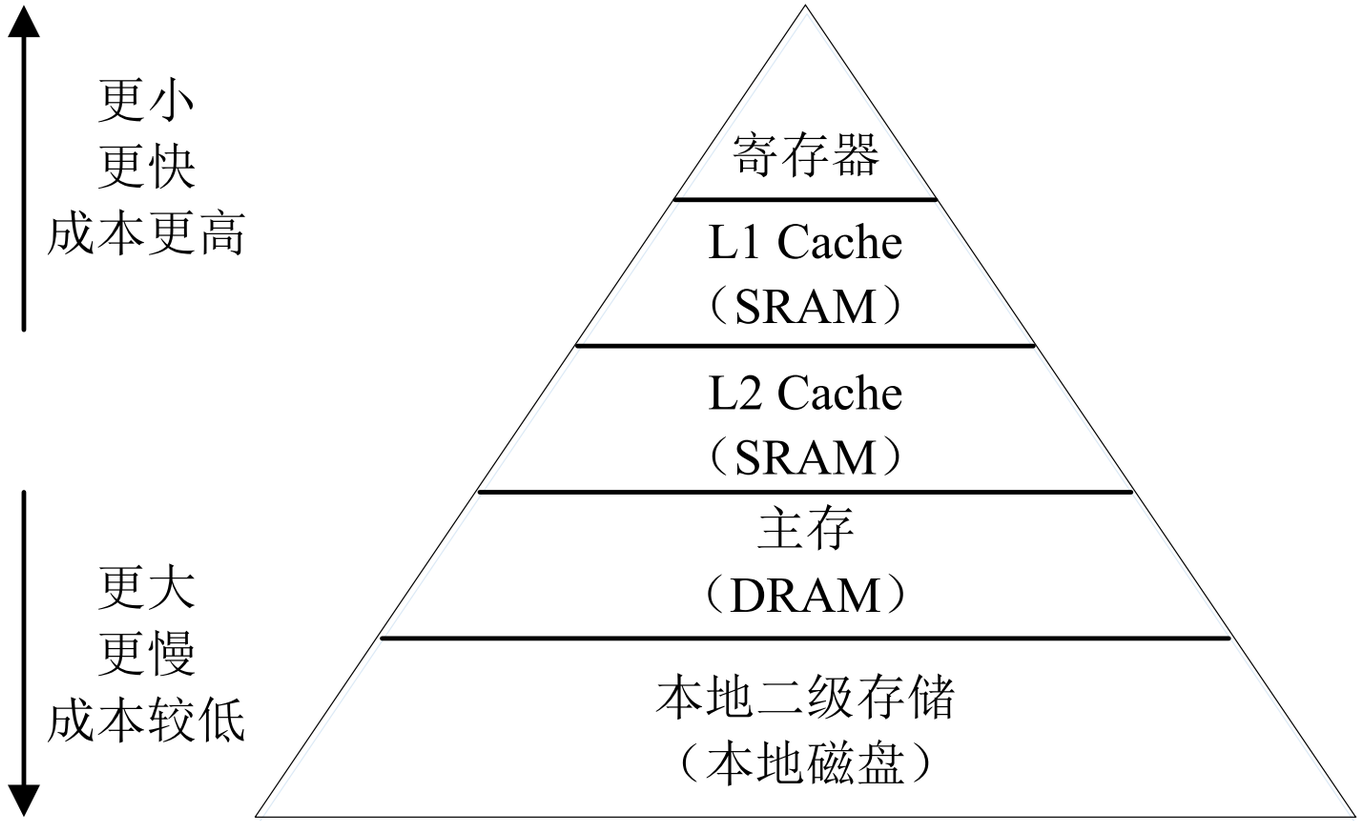
实验原理

- **1、背景**
  - **为什么要使用 Cache**
    - 计算机性能的瓶颈：内存访问的延迟远大于处理器的时钟周期
    - 局部性原理：时间局部性和空间局部性
  - **设计 Cache 时通常需要考虑的问题**
    - Cache 规模
    - Cache Line 规模
    - Cache 相联度
    - 写策略
    - 替换策略
    - 一致性 ……



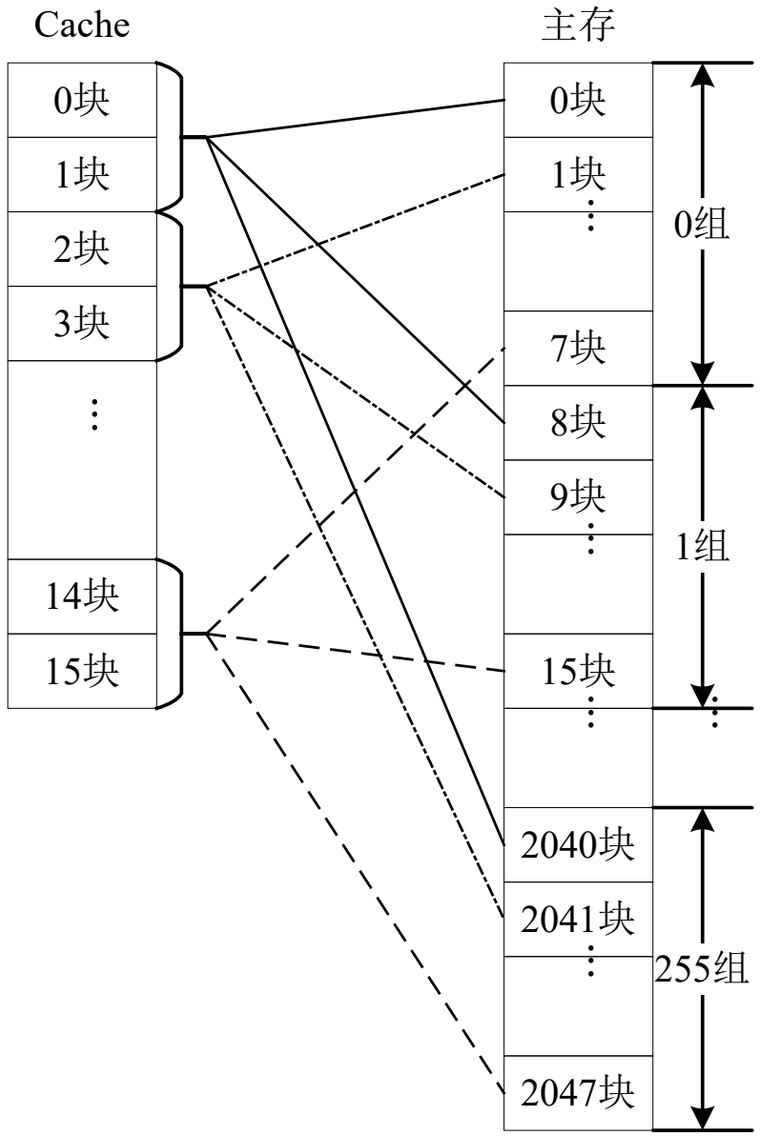
实验原理

- 2、Cache
  - 位于CPU和主存之间，规模较小，速度很高的存储器
  - 通常由SRAM组成



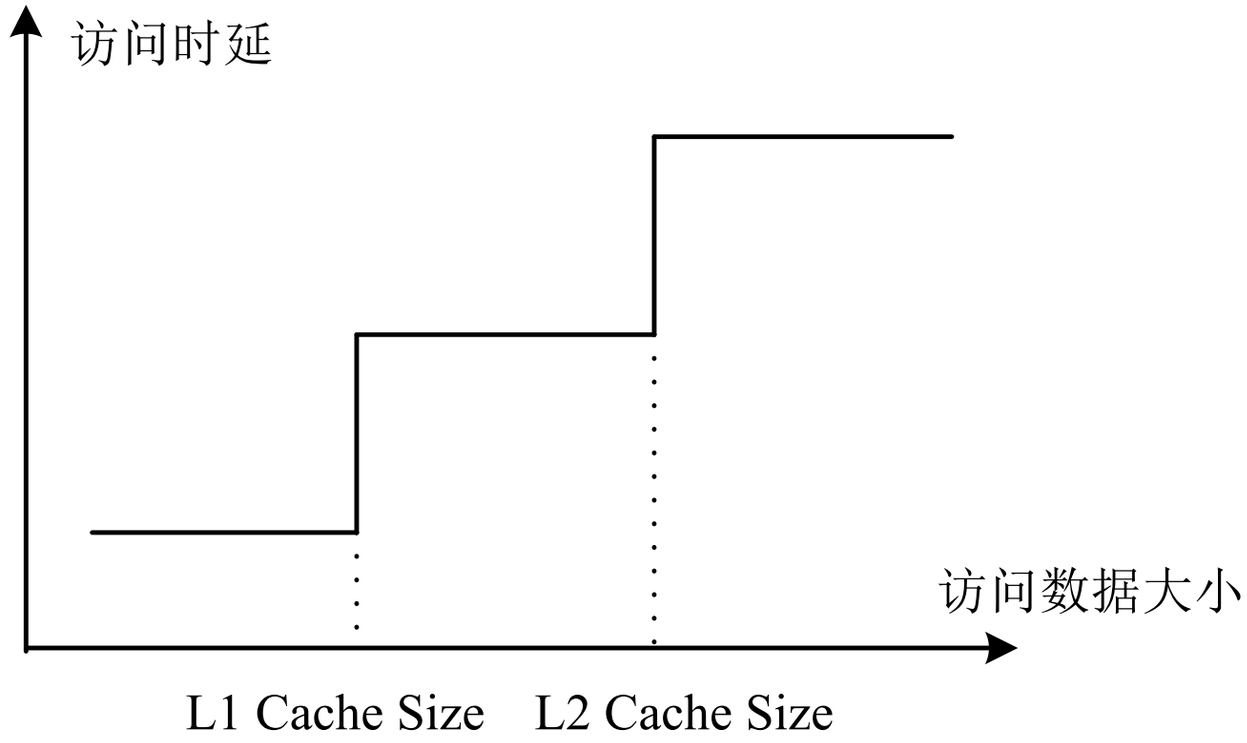
实验原理

- **Cache组相联映射**
  - 组相联映射实际上是组间采用直接映射，组内采用全相联映射。
  - 例如，主存分为256组，每组8块，Cache分为8组，每组2块。



实验原理

- 3、测量Cache容量
  - 当访问的数据块大小超过L1 Cache后，会出现Cache读缺失，平均读取速度会有一个突然增加的过程，L2 Cache亦然。

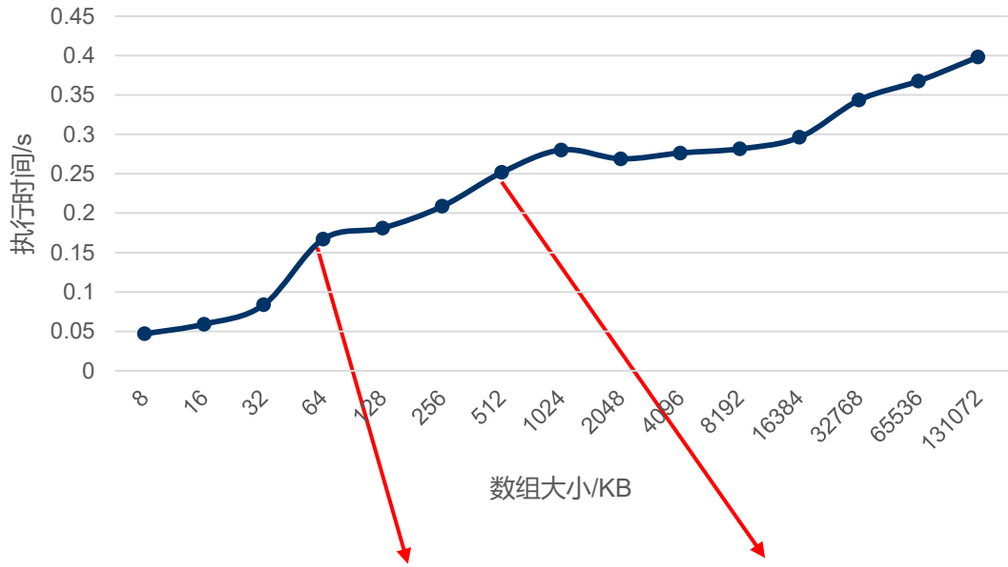


实验原理

- 3、测量Cache容量
  - 某台机器的某次实验结果示例

    (L1D Cache Size = 32 KiB，L2 Cache Size = 256 KiB)

```
test size = 8 KB, finish - start = 0.046856 s
test size = 16 KB, finish - start = 0.059107 s
test size = 32 KB, finish - start = 0.083651 s
test size = 64 KB, finish - start = 0.167041 s
test size = 128 KB, finish - start = 0.181165 s
test size = 256 KB, finish - start = 0.208699 s
test size = 512 KB, finish - start = 0.251703 s
test size = 1024 KB, finish - start = 0.280279 s
test size = 2048 KB, finish - start = 0.268928 s
test size = 4096 KB, finish - start = 0.276383 s
test size = 8192 KB, finish - start = 0.281609 s
test size = 16384 KB, finish - start = 0.296222 s
test size = 32768 KB, finish - start = 0.343518 s
test size = 65536 KB, finish - start = 0.367744 s
test size = 131072 KB, finish - start = 0.398132 s
```



数组大于 32KiB 后执行时间明显增大

数组大于 256KiB 后执行时间也有较明显增大

实验原理

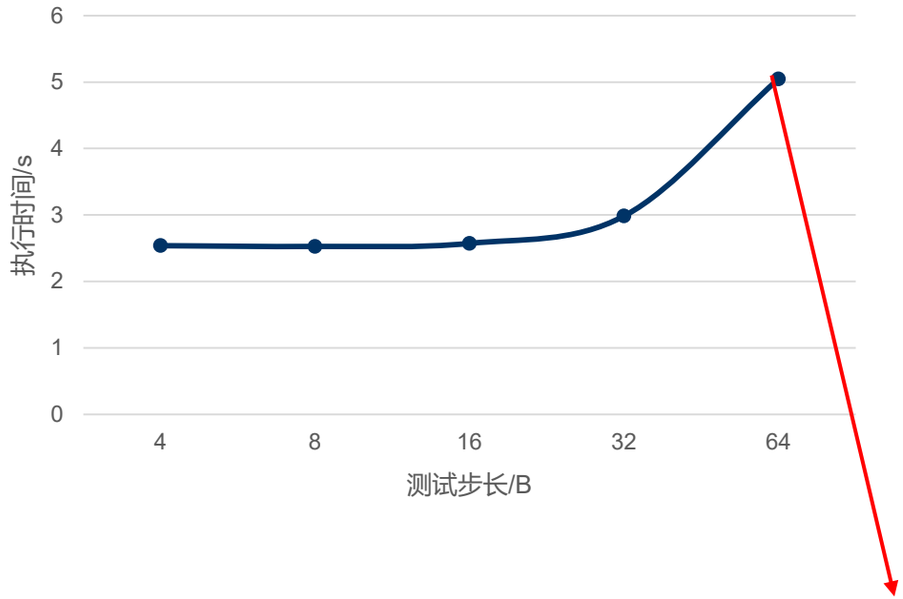
- **4、测量Cache块大小**
  - 对于数组访问，如果是连续访问，第一个字节缺失的话，一个块就被存放在了cache中，以后cache块顺序访问的字节就命中了，因此命中率就会很高。 如果访问是间断的，对数组间隔顺序访问，命中率就会降低，平均访问延迟增大。当某次间隔达到一定大小，即超过cache块大小，有可能造成每次都缺失的最坏情况 。

实验原理

- **4、测量Cache块大小**
  - 某台机器的某次实验结果示例

    (L1D Cache Line Size = 64B)

```
test block size = 4 B, finish - start = 2.538046 s
test block size = 8 B, finish - start = 2.526867 s
test block size = 16 B, finish - start = 2.572923 s
test block size = 32 B, finish - start = 2.984372 s
test block size = 64 B, finish - start = 5.047482 s
```



步长为64B时执行时间明显增大，说明 Cache 缺失率变大

实验原理

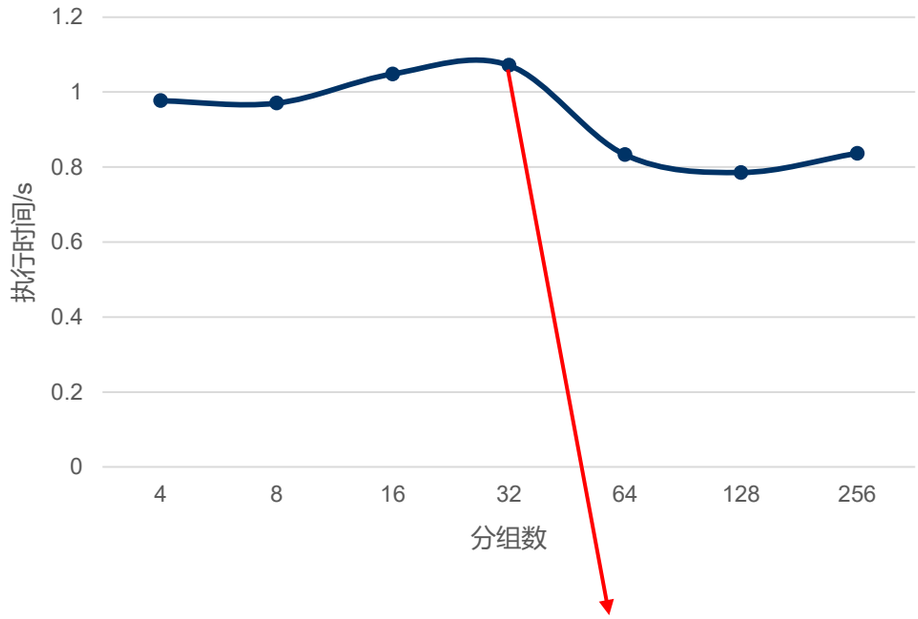
- **5、测量Cache相连度**
  - 用一个2倍Cache大小的数组。 比如如果是2路组相连的结构，
  - 如果将数组分为4块，访问数组的第一块和第三块，由于是2路组相连，第一块对应的Cache块和第三块对应的Cache块映射到同一个组里，可以同时被放到这个组里；
  - 如果将数组分为8块，访问数组的第一块，第三块，第五块，第七块，这时第一、三、五、七块对应的Cache块映射到同一个组里，但一个组只能容纳两个Cache块，因此不停的循环访问会造成Cache块不断的被替换出去，使得访存变得很慢。

实验原理

- **5、测量Cache相连度**
  - 某台机器的某次实验结果示例

    (L1D Cache 相联度为 8)

```
test number of groups = 4, finish - start = 0.976846 s
test number of groups = 8, finish - start = 0.970444 s
test number of groups = 16, finish - start = 1.048350 s
test number of groups = 32, finish - start = 1.071333 s
test number of groups = 64, finish - start = 0.832903 s
test number of groups = 128, finish - start = 0.785498 s
test number of groups = 256, finish - start = 0.836780 s
```



n = 5，即分成32组时访存时间相对较大，
所以可估计相联度为2^{n-2} = 8

实验原理

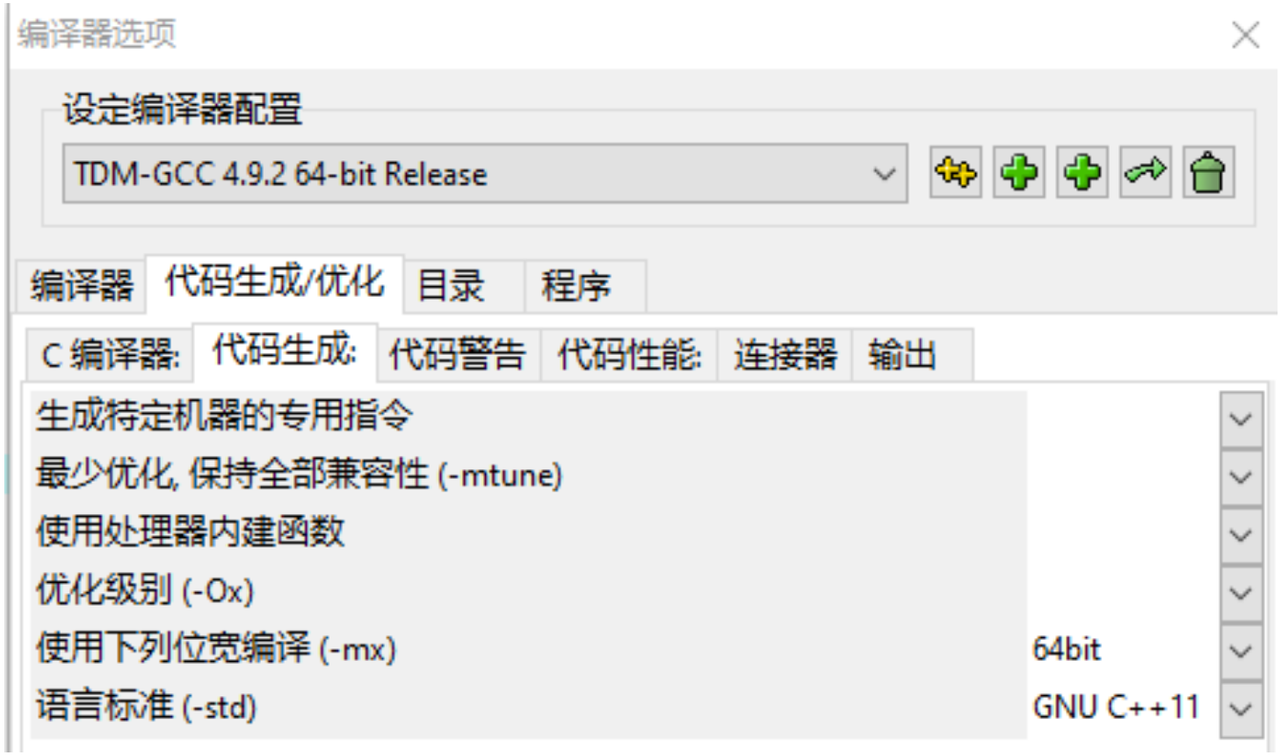
- **6、优化矩阵乘法**
  - 空间局部性
    - 基本算法按照i,j,k的访问顺序，使用公式c[i][j] += a[i][k] * b[k][j]
    - 虽然按内存地址顺序访问了数组a、c，但是对于数组b却是按列访问的，这破坏了访问内存的空间局部性，造成更大的Cache替换浪费。
  - Cache容量
    - 从Cache容量考虑，最大限度地利用Cache容量，同时降低缺失率，达到访问时间最快。

实验步骤

- **实验环境**

  Dev-C++ 5.11，使用 C/C++ 编程

  使用C++ 11标准



实验步骤

- **1、查询本机器的Cache结构**

运行cpu-z，查看本机的Cache结构，以比对实验结果：

实验步骤

- **2、编写程序**
  - 将给定的事例程序cache_test.cpp拷贝到自己的运行目录；
  - 在add your own code处补全代码

```
int L1_DCache_Size() {
        cout << "L1_Data_Cache_Test" << endl;
        //add your own code
}

int L2_Cache_Size() {
        cout << "L2_Data_Cache_Test" << endl;
        //add your own code
}

int L1_DCache_Block() {
        cout << "L1_DCache_Block_Test" << endl;
        //add your own code
}

int L2_Cache_Block() {
        cout << "L2_Cache_Block_Test" << endl;
        //add your own code
}

int L1_DCache_Way_Count() {
        cout << "L1_DCache_Way_Count" << endl;
        //add your own code
```

实验步骤

- 3、运行程序并查看结果
  - 运行编译程序；
  - 使用clock()计时

```
#include <time.h>

clock_t start = clock();

/* access pattern */

clock_t finish = clock();
```

```
L1_Data_Cache_Test
Test_Array_Size = 8KB   Average access time = 26.375ms
Test_Array_Size = 16KB  Average access time = 25.75ms
Test_Array_Size = 32KB  Average access time = 25.6562ms
Test_Array_Size = 64KB  Average access time = 29.0156ms
Test_Array_Size = 128KB Average access time = 28.9375ms
L1_DCache_Size is 32KB
**************************************************
L2_Data_Cache_Test
Test_Array_Size = 64KB  Average access time = 58ms
Test_Array_Size = 128KB Average access time = 57.7891ms
Test_Array_Size = 256KB Average access time = 58.1758ms
Test_Array_Size = 512KB Average access time = 61.1777ms
Test_Array_Size = 1024KB        Average access time = 61.3584ms
L2_Cache_Size is 256KB
--------------------------------------------------
```

附加题

- **1、测量Cache写回策略（+1分）**
  - 写直达：数据在写到Cache块的同时，还要写入存储器
  - 写回法：数据仅写入Cache；仅当Cache的数据块被替换时，才写回到主存中
- **2、测量Cache替换策略（+1分）**
  - 根据测得的相联度，设计不同的访问序列，产生不同次数的缺失，以验证Cache所使用的替换策略（如LRU、LFU等）
- **3、矩阵乘法优化（ +1~2分）**
  - 根据矩阵乘法的访存特性，采取指导书以外的策略优化矩阵乘法算法

实验报告要求

- 题目分析
- 设计思路
- 关键实现
  - 可用图表描述
- 测试结果及分析
  - 需绘制测量测参数时，Cache的访问时间折线图

实验检查与提交

- 检查指令依赖距离分布图
- 将**源码**、**实验报告**打包提交
  - 命名规则：**学号_姓名_ARCH实验3.zip**
  - 提交方法：https://hitsz-cslab.gitee.io/arch/ojguide
  - Deadline：下周同一上课时间前
- 附加题：将设计思路、关键代码等写入报告，与源码一起打包提交（**+1~2**分）

开始实验