计算机组成原理实验设计总结

第九组

肖坤朋 1911491　刘辰飞 1911434　徐佳培 1911499

刘旭源 1911441　赵世浩 1911522

1　实验名称

探究不同的 cache 替换算法，不同块大小，不同组相联度对 cache 存储性能的影响

2　实验目的

a) 熟悉 cache 的工作原理

b) 理解不同 cache 替换算法的算法思想

c) 理解不同 cache 的块大小，组相连度对 cache 性能的影响

3　实验环境

本实验的操作基于计算机程序设计语言 c++，要求在一台装有 C++ 编译器的计算机上进行实验。

4　实验要求

a) 做好预习：

1) 掌握 cache 存储的原理

2) 掌握 cache 不同替换算法的原理

3) 掌握衡量 cache 性能的方法

4) 在课前画好设计框图或实验原理图

5) 如果有兴趣可以在互联网上了解更多 cache 替换算法

b) 实验实施

1) 确定实验原理图的正确性

2) 编写实验代码

3) 完成不同替换算法的代码设计

4) 运行代码得出正确的分析结果，并将分析结果作为实验报告的一部分

c) 实验检查

1) 完成实验后，让指导老师或助教进行检查，机进行现场演示，可对演示结果进行拍照作为实验报告结果的一项材料

d) 实验报告撰写

1) 实验结束后，须按照规定的格式完成实验报告的撰写

5 实验内容

5.1 实验原理

本设计首先需要了解 cache 存储的原理，其原理图如下

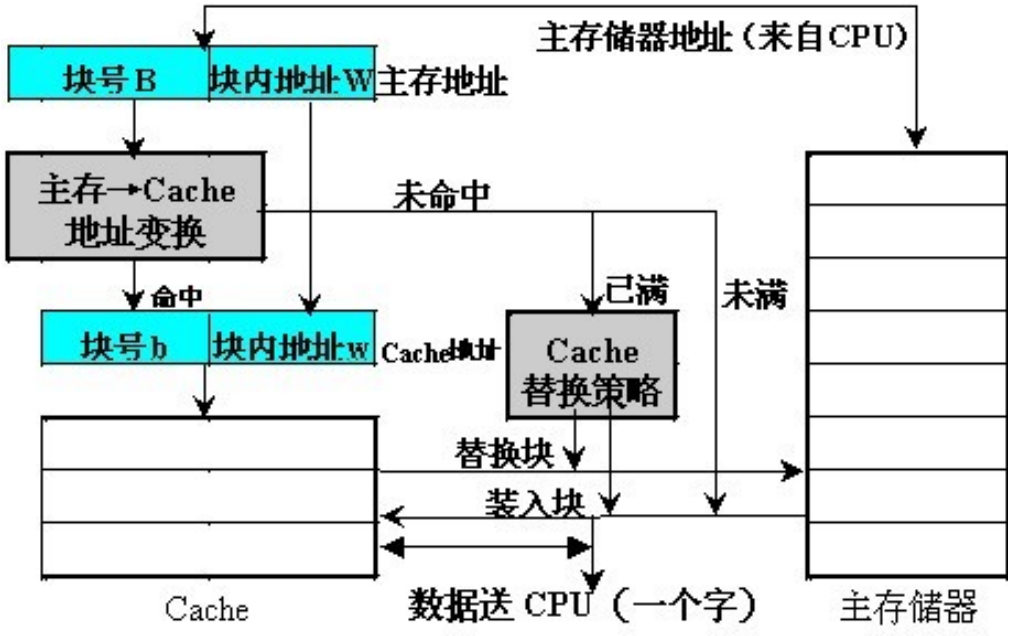


图 1: Cache 基本原理

当 cache 未命中时，如果 cache 相应的组已满，我们就需要执行相应的替换策略，从 cache 中剔除一个块来个新的数据腾出位置，对于直相联 cache, 每一个索引, 只有一个块可被替换; 对于全相联, 由于没有索引位, 所有的块都可被替换; 对于组相联, 则需要在组内选择块替换。最常用的替换策略有最久未使用和随机替换法。cache 的替换算法主要使用硬件实现, 而随着 cache 大小的提升, 各种算法的缺失率都会下降。而因为缺失代价较大, 所以应该尽可能地提升命中率; 因为缺失不

经常发生, 可以采用软件的方法来处理缺失, 精确找出最合适的块替换。常见的方法有 LRU、MRU、RAM 等等。

6 实验模拟过程

6.1 替换算法的模拟

在本实验中，用 C++ 模拟实现以下 cache 的替换算法

LFU

即最不经常使用页置换算法，要求在页置换时置换引用计数最小的页。

LRU

即最近最少使用替换算法，其选择最近最久未使用的页面予以进行替换。

MRU

即最近最多使用替换算法，其在 Cache 组中选择最近使用最多的块进行替换

RANDOM

即随机替换算法: 其在 Cache 组中随机选择一个块进行替换

NRU

一种近似于 LRU 算法的替换方法，通过标注 Not Recently Used 位，从中选择最近不常用的块进行替换

6.2 计算在特定情况下某种算法的命中率

1) 计算命中数，即 cache 利用该算法在本轮实验 cpu 请求中成功命中的总次数。

2) 计算缺失数，即 cache 利用该算法在本轮实验 cpu 请求中未能命中的总缺失数。

3) 计算命中率，即

$$Hit\,Ratio = \frac{Hit\,time}{Try\,Time} \tag{1}$$

6.3 计算在特定情况下某种算法的时间消耗

在特定情况下的时间消耗主要包括：

1) cache 组内寻找所需块的时间，依赖于组相联大小设置。

2) cache 缺失导致的访问内存的时间，即计算机访存时间，但总缺失时间依赖于算法所导致的缺失率。

3) cache 缺失时，针对不同块大小，其载入时间也有所区别

6.4 采用不同的实验数据、块大小、索引域大小和组相联大小多次试验

1) 以上四种参数对替换算法的命中率以及时间消耗起着决定性作用。

2) 对比不同情况下各个替换算法的性能，比较总结得出结论。

7 实验示例

在设计完成试验后，通过编写 C++ 代码，对实验进行了模拟操作，在实现过程中，通过设计了两个随机数来完成一次绝对地址与相对地址的模拟，在一定程度上模拟了程序的局部性特点，以下对实验基本条件，实验基本过程，结果与结论进行说明，其具体代码在附件中：

在这一系列实验中，模拟了一个大小为 4KB 的 cache，而其模拟的主存地址的范围为 0 到 2^{20}，其可能的偏移最大为 2^{14}, 每一次实验对其进行 10000 次模拟读取操作，每一组实验进行 20 次，记录数据并画出统计图。

7.1 块大小对于命中的影响

对于 cache 大小一定的情况，随着块大小的增大，块的数目也将会减少，这也会使得没进行一次读取，会有更大范围地址内的数据被装入 cache，从而影响命中率。

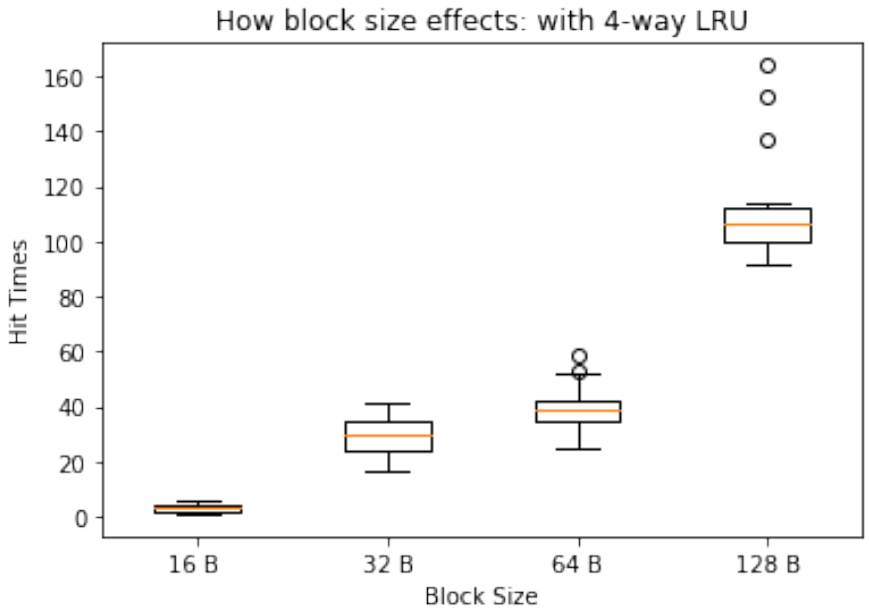


图 2: 块大小对于命中的影响

在这一实验中，可以发现，随着块大小的增大，其命中率有着显著提高，这是符合一般认知的。但与此同时，块缺失惩罚也会加重，每次缺失会需要更多的时间来从主存装入数据，基于此，选择合适的块大小可以直接影响 cache 的性能。

7.2 组相联对于命中的影响

一般而言，可以通过组相联的方式降低冲突缺失以提高 cache 的整体性能：

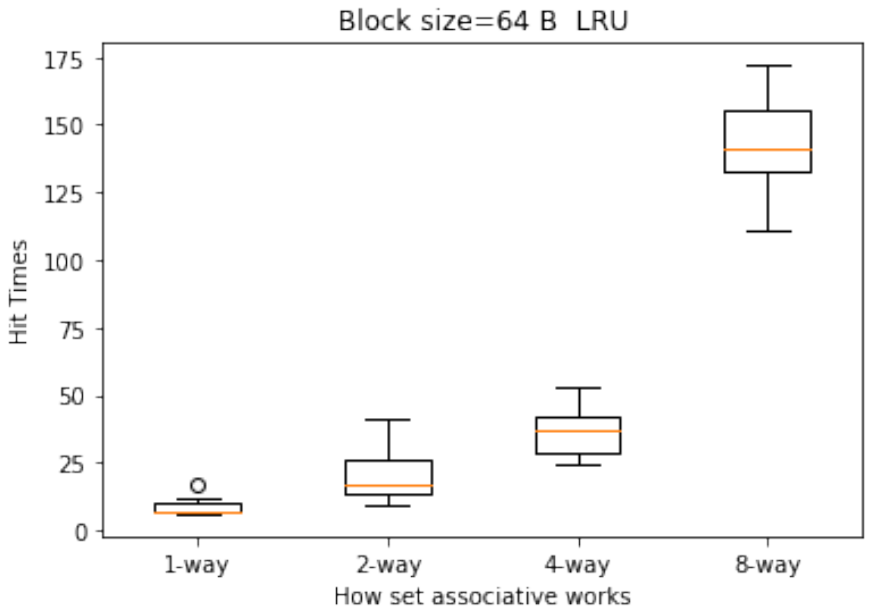


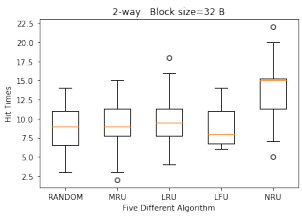
图 3: 组相联对于命中的影响

在这一实验中，可以发现，随着组相联程度的增大，其命中率有着显著提高，但与此同时，对块进行读取操作时时间消耗也会加大，其电路复杂程度也会提高，在这样的情况下，需要在性能与成本之间作出合适的 trade-off，选择合适的组相联程度。

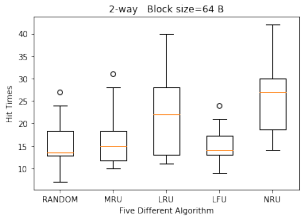
7.3 替换算法对于命中的影响

对于替换算法的选择，应该基于合适的应用场景，也就是说，应当根据硬件所确定的块大小与组相联度数确定替换算法。

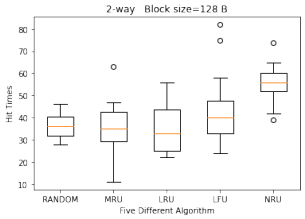
块大小对算法的影响 在这一场景下，选择二路组相联这一相联度，进行实验，其结果如下：



(a) 块大小为 32B



(b) 块大小为 64B

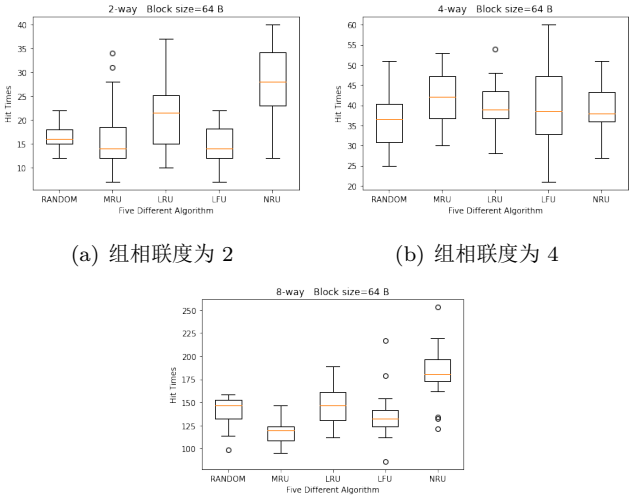


(c) 块大小为 128B

图 4: 块大小对算法选择影响

由此结果可知，在二路组相联的情况下，NRU 算法始终有着较为良好的效果，而与与其相似的 LRU 算法在块较小时效果较好，这为块大小影响下算法的选择提供了一定的参考。

组相连度对算法的影响 在这一场景下，选择 64B 作为块大小，对不同的相联度的选择进行实验，其结果如下：



(a) 组相联度为 2　　(b) 组相联度为 4

(c) 组相联度为 8

图 5: 组相连相度对算法选择影响

基于此结果，可以得知：在块大小为 64B 的情况下，对于不同组相联度，NRU 与 LRU 的效果较好，这也是由它们算法的特点比较契合此次实验所设计的随机数机制。在不同情况下，应该通过更加良好的方式进行模拟与比较。

在以上几个情境下，NRU 与 LRU 的效果显著好于其它算法，而 RANDOM 与 MRU 没有较大区别，这符合实验预期，即模拟地址时随机使用一个较小偏移来模拟局部性，但对于一些可能的较大范围的跳转指令并没有进行良好的模拟，使得 MRU 类型的算法无法发挥其优势。

基于时间代价对 cache 性能量化分析　由于本次实验未能查找到比较准确，全面的关于不同块大小，不同相联度对时间的影响，因此没有进行量化的时间代价比较，而主要从

命中次数出发进行实验。
对于这一问题，模拟代码中已经预留了函数进行计算，后续只需更改具体的时间参数即可。

8 实验的改进

本次实验的不足主要在以下几个方面：

1) 尽管在一定程度上通过改进生成随机数的机制模拟了局部性，但并没有能够模拟出更加精确的，具体的程序实际情况，其偏移参数相比 cache 大小也过大，这也是造成命中率较低的主要原因。

2) 模拟实验的代码鲁棒性仍有所欠缺，需要进行一些优化。

3) 需要更加精确的数据支持对时间代价的模拟，本例中的数据基于一系列不完整资料，因此仍存在不精确等问题。

在此基础上，应该对实验进行后续改进以达到更好的模拟效果。