期末作业：cache 替换策略和数据预取

2022 年 12 月

1. 实验介绍

在本实验中，你将使用 ChampSim 模拟器(ChampSim 是一个基于 trace 的微体系结构模拟器)实现和评估不同的 L2 Cache 数据预取和 LLC cacheline 替换策略。

我们将为你提供一些程序的 trace，以及 ChampSim 的源代码。我们还将提供对应接口，你的工作是补全接口实现不同的预取算法和缓存替换策略，并评估其性能。

2. 实验

2.1 实验步骤

首先键入命令 clone repo:

$ git clone https://github.com/XDUFanYang/ChampSim

在进入文件夹后，你需要执行 build_champsim.sh，这是一个用于构建模拟器的 shell 脚本。该脚本只接受一个输入参数，即预取算法名称。我们提供了几个示例，以帮助你入门。使用以下命令编译 ChampSim。

$ ./build_champsim.sh next_line

随后会显示：

```
ChampSim is successfully built
Branch Predictor: perceptron
L1D Prefetcher: no
L2C Prefetcher: next_line
LLC Prefetcher: no
LLC Replacement: ship
Cores: 1
Binary: bin/perceptron-no-next_line-no-ship-1core
```

这说明编译完成，成功生成二进制文件。输出的日志显示模拟器使用 perceptron 分支预取器，在 L2Cache 中采用 next_line 预取器，在 LLC 上采用 ship 替换策略，其中模拟的核数为 1，生成的 bin 文件名为 perceptron-no-next_line-no-ship-1core。

如下图所示，我们可以在 build_champsim.sh 中灵活设置使用不同的预取策略和 cacheline 替换策略。你可以为你的策略起一个喜欢的名字，然后在该文件中进行设置，确保编译正确。

```
23  ############## Default configuration ############
24  BRANCH=perceptron
25  L1D_PREFETCHER=no
26  LLC_PREFETCHER=no
27  LLC_REPLACEMENT=ship
28  NUM_CORE=1
29  #################################################
```

在编译成功后，需要下载相应的数据集进行测试，我们提供了几个数据集 trace 方便同学们检测自己的模拟器效果。在准备好 trace 后，要检测 ChampSim 效果，请执行 run_champsim.sh 脚本。

$./run_champsim.sh <BINARY> <WARMUP INST> <SIM INST> <TRACE>

这个脚本有如下输入参数

1. BINARY: 编译好的模拟器 bin 文件的路径。
2. WARMUP INST: warm up 指令数 (默认为 100，使用 100M 条指令预热)。
3. SIM INST: 需要模拟的指令数 (默认为 500，模拟 500M 条指令)。
4. TRACE: 需要执行的 trace 文件的路径。

如果我们要评测 perceptron-no-next_line-no-ship-1core 在 436.cactusADM-1804B.champsimtrace.xz 数据集上的表现，命令行可以这么设置：

$./run_champsim.sh bin/perceptron-no-next_line-no-ship-1core 100 100 traces/436.cactusADM-1804B.champsimtrace.xz

在程序运行时，输出流会显示在终端上，你也可以重定向到某个文件。输出流包含了很多信息，包括模拟器的设置以及模拟过程中的性能数据，我们将 cumulative IPC 作为性能指标进行评分。

为了方便一次性检测多个数据集效果，我们提供了 score.py 脚本，这个脚本需要输入以下两个参数：

1. bin_path：编译成功的模拟器二进制文件。
2. trace_dir_path：trace 所在的文件夹路径。

脚本会统计每个数据集的 cumulative IPC，并且最终会给出平均 score，这将作为你模拟器性能的重要参考。

2.2 实验代码

下面将简要介绍 ChampSim 的结构。除非明确提到，请不要修改这些文件。

- inc: 此目录包含所有的头文件，编译过程中将自动包含这些头文件。**请不要更改这些文件中的任何参数。**
- src: 这个目录包含 core、cache 等微架构的文件。你可以浏览这些文件来更好地理解 ChampSim。
- prefetcher and replacement: 这是你要修改的地方。ChampSim 可以在多个缓存上实现预取器，实验中你只需在 L2 上实现预取器。L2 预取函数在 l2c prefetcher.cc 文件中定义。整个操作由五个关键函数组成:
  1. l2c_prefetcher_initialize: 初始化。
  2. l2c_prefetcher_final_stats: 这是在模拟结束时调用的函数，用于打印模拟的统计信息。
  3. l2c_prefetcher_operate: 每个 L2 查找操作都调用此函数。这意味着对二级缓存中可能命中或未命中的读写操作都会调用它。
  4. l2c_prefetcher_cache_fill: 每个二级缓存填充操作都会调用此函数。
  5. prefetch_line: **你不需要实现此功能**，但需要将预取请求发送到下一级内存层次结构时调用它。默认情况下，预取器在缓存级别 N 生成的预取请求首先查找第 N 级缓存。未命中时，预取请求查找下一个缓存级别的缓存（N+1、N+2 等），如果未命中 LLC，则最终进入主存。一旦内存提供

了相应的数据，数据将填充从 LLC 到第 N 级的所有缓存级别。

LLC 的 cache 替换策略在 llc_replacement.cc 文件中定义。其中关键函数有 4 个：

1. llc_initialize_replacement：初始化。
2. llc_find_victim：返回符合替换策略的 cacheline。
3. llc_update_replacement_state：进行 cacheline 替换。
4. llc_replacement_final_stats：打印模拟的统计信息。

2.3 实现一个预取器

要实现你自己的预取器，请以<prefetcher_name>.l2c_pref 为名创建一个新文件，并使用 C++语法在规定的 4 个函数中补充自定义逻辑。如果你不需要某个函数，直接将函数体清空就好，请不要删除函数。请不要修改 l2c_prefetcher.cc 文件。对应脚本将你的 lc_pref 文件自动生成为 l2c_prefetcher.cc 文件。

为了帮助你了解整个过程，我们提供了两种简单的预取器实现：（1）next-line prefetcher 和（2）table-based IP-stride prefetcher。这些文件可以帮你简要了解如何实现预取器。我们还提供了一个空的 L2 预取器，以模拟没有任何预取器的系统。

2.4 实验缓存替换策略

要实现自定义的缓存替换策略，请使用<replacemet_name>.llc_repl 为文件名创建一个新文件，并使用 C++语法在规定的 4 个 API 中补充自定义逻辑。如果你不需要某个函数，直接将函数体清空就好，请不要删除函数。

为了帮助你了解缓存替换的整个过程，我们提供了两种简单的实现：（1）ship replacement policy 和（2）srrip replacement policy。

3. prefetch 基本知识

预取是一种预测技术，用于预测未来的内存请求，并在处理器实际需要之前将其提取到缓存中。这种预测可以使程序运行更快。在具有三级缓存层次结构的处理器核中，可以在任何缓存层次上使用预取程序。现在有许多工作提出了预取算法，以准确预测程序未来的内存访问。

4. cache 基本知识

在计算机领域，缓存可以加速处理器对于内存的访问。内存请求在缓存中找到数据时比没找到时处理地要快。其中影响缓存性能的一个关键因素是缓存的替换策略，替换策略指定当一个新的缓存行插入到缓存中时，集合中的哪个缓存行被替换。以 LRU（Least Recently Used）为例，LRU 使用了一个替换策略来替换最近使用最少的行。

5. 实现一个预取算法

这里推荐实现的是基于 GHB 的步长预取算法。

你的目标是通过函数在 L2 缓存上实现一个基于全局历史缓冲区(Global History Buffer)的步长预取器。 预取器记录 L1 数据缓存中丢失的负载的地址的访问模式，并将预测的数据预取到 L2 缓存中。它由两个主要结构组成：

•索引表(Index Table，IT)：一个通过程序属性(例如程序计数器 PC)进行索引的表，它存储指向 GHB 条目的指针。

•全局历史缓冲区(GHB)：一个循环队列，存储 L2 缓存观察到的缓存行地址序列。每个 GHB 条目存储一个指针(这里称为 prev_ptr)，该指针指向具有相同 IT 索引的最后一个缓存行地址。通过遍历 prev_ptr，可以获得指向同一 IT 条目的缓存行地址的时间序列。

算法过程：对于每个二级缓存访问(命中和未命中)，该算法使用访问的 PC 索引到 IT 中，并将缓存行地址(例如 A)插入 GHB。该算法使用 PC 和 GHB 条目中的链接指针，检索访问二级缓存对应条目中的最后 3 个地址。步长是通过取序列中 3 个连续地址之间的差来计算的。如果两个步长相等(步长为 d)，预取器只向缓存行 A+ld、A+(l+1)d、A+(l+2)d、…、A+(l+n)d 发出预取请求，其中 l 是事先设定好的预取 look-ahead，n 是度。对于你的设计，你可以将 l 和 n 静态设置。还请调整 IT 和 GHB 的大小，使其为 256 个条目。有关基于 GHB 的步长预取器实现的更多详细信息，请阅读 GHB 论文[1]。

6. 实现一个 cache 替换算法

这里推荐在 LLC 上实现 LFU 算法。

LFU 缓存算法维护一个关于缓存中缓存行访问次数的计数器。它将删除访问频率最低的项，以便添加新项。每次进行缓存读写时，缓存行中的访问次数状态都要有所变化。

7. 性能评测

你的系统将在多个数据集上进行实验，在性能评测时，你可能需要编辑 build_champsim.sh 来编译你的模拟器，你也可能使用 score.py 来得到模拟器统计数据。模拟器输出的 IPC 数据将表示你的模拟器在该数据集上的性能。当然，你也可以用更多的指标来检查你的模拟器。同样，我们鼓励你能有更多的可能性，比如进行更多预取策略的尝试比较、更系统的策略组合、结合 ai 进行策略制定。同时，我们希望你的报告格式符合学术标准，这里有几篇参考文章[2][3]。这些都是我们给你实验评分的依据，期待你的发挥。

注意：你不能修改除了这两项以外的任何设置，你需要保持 l2c_size 等参数为初始状态。

8. 作业要求

最低要求：正确编译 ChampSim，并且完成最少一个预取器和一个 cache 替换策略的添加。这里推荐在 L2 cache 上实现 GHB 步长预取器，在 LLC 上实现 LFU cacheline 替换策略。你还需要完成一个详尽的实验报告，里面要包括你模拟器的具体实现和实验效果分析。

进阶要求：实现多种预取器和 cache 替换策略并且进行性能比较，采用多项性能指标来解释不同策略的优缺点。实现更多复杂策略的组合，并且详细分析引入的代价和性能提升效果。

参考文献：

[1]Nesbit, K. J., & Smith, J. E. (2004, February). Data cache prefetching using a global history buffer. In 10th International Symposium on High Performance Computer Architecture (HPCA'04) (pp. 96-96). IEEE.

[2] Chaudhuri, M., & Deshmukh, N. (2019). Sangam: A multi-component core cache prefetcher. 3rd Data Prefetching Championship.

[3]Sethumurugan, S., Yin, J., & Sartori, J. (2021, February). Designing a cost-effective cache replacement policy using machine learning. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 291-303). IEEE.