模拟器代码简介

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，假如你要在 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。

实现一个预取器

要实现你自己的预取器，假如L2层预取器，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。

实现缓存替换策略

要实现自定义的缓存替换策略，以LLC层为例，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。

模拟器代码获取和运行过程

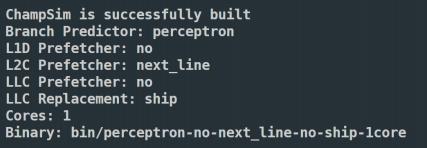
首先键入命令 clone repo:

$ 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。

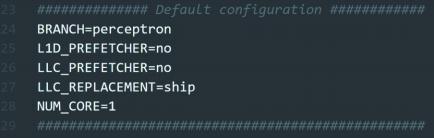
$ ./build\_champsim.sh next\_line

随后会显示：



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。



在编译成功后，需要下载相应的数据集进行测试。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。