M3：GPT-2 并行推理 (gpt.c)

April 18, 2025

2025

在人工智能时代，特别是大语言模型快速发展的今天，我们预见一个更加智能和互联的世界。AI 将深刻改变我们获取信息、进行沟通和解决问题的方式。那么，到底是什么藏在 AI 背后?

⏰Soft Deadline: 2025 年 5 月 6 日 23:59:59

你需要首先阅读实验须知，其中包含了代码获取方法、提交方法、如何查看提交结果等。在命令行中 `git pull origin M3` 下载框架代码。

⚠️请不要参考 llm.c 的实现

在 2024 学期开始前，jyy 就受到 llama.cpp 启发，有了实现 “gpt-2.cpp” 作为并行编程实验的想法。没想到 Andrej Karpathy 亲手下场实现了 llm.c，它使用了 OpenMP 进行并行。如果你 “偷看” 到应该如何做并行，本次实验的效果就有折扣。本次实验难度不高，请遵守学术诚信，独立完成实验。

1. 背景

GPT 这样的 “大语言模型” 本质上是一个函数 `f`，能够接收一个 “部分文本”，输出 “文本可能的下一个字符”。这个 “智能” 函数 `f` 具有惊人的应用场景，包括 “一切” 的 copilot (编程、操作系统使用、甚至是课堂学习)，和可以各种替代人类的智能 agent。而 `f` 训练好之后，就会部署在云端的推理服务，以 API的形式提供服务。我们预见，AI 推理服务将会和水、电、云一样成为人类社会的重要基础设施。

2. 实验描述

2.1 总览

gpt \ [token] \ ...

2.2 描述

使用 `gpt2_124M` 模型将输入的代表了一段文本的 token (整数) 序列进行补全，输出后续可能的 tokens (包含输入，总有 $n = 10$ 个 tokens)，作为概率意义上 “可能” 的后续文本。

2.3 解释

GPT-2 是一个基于 Transformer 的早期模型，但奠定了 OpenAI 的技术基础。并行化是提高大规模模型计算效率的关键技术之一。gpt.c 实现了生成式人工智能的 “文本补全”，框架代码已经提供了从 llm.c 中裁剪出的完整的神经网络推理 (inference) 实现。下载 `gpt2_124M.bin` 到实验目录即可正常工作，实现文本的补全。与它交互，你可以感受到自然语言处理的飞速发展 (GPT-2 是 2019 年的模型，我们使用的是 124M 的小版本模型，能力与今天的模型无法匹敌，但依然能看到它的确可以生成 “合法” 的句子)。

⚠️你需要手动下载模型

模型文件较大 (~500MB)，因此你需要手动下载到实验目录。这个文件不会被 git 追踪，也不会上传到服务器。

我们提供了一个 Python 的脚本 `chat.py`，可以直接输入一段文本，完成 tokenize，并且调用 gpt 命令行工具进行补全，没错，你真的在实验中使用了一个能生成文本的大语言模型！代码已经全部给大家准备好，并且可以正常工作了！在下面的例子里，数字序列 “31373 612” 就是 “Ladies and”。可以看到，语言模型的确为我们生成了可阅读的文本：

```
$ ./
```

没错，即便这么小的模型，单线程的实现在 CPU 上推理都非常吃力，更不必说之后 “超大” 的 GPT-3 (1750亿参数)。神经网络推理优化是一个非常复杂的主题；在这里我们试着做出并行化的第一步：

> 实验要求：将 gpt.c 并行化s
> 我们的 gpt.c 是串行的代码 (但功能未变)，只能利用单个处理器。你需要找到代码中可并行 (并且有收益) 的部分，改造成并行，从而获得相应的性能提升。

在这个实验中，我们可以使用课堂上的线程库 (thread.h 和 thread-sync.h)，其中包括线程的创建和回收、互斥锁、信号量和条件变量。你可以选择你喜欢的机制进行同步和互斥。

3. 正确性标准

> 正确性 & Scalability
> 你并行化后的**程序行为** 应当与我们给出的串行程序保持严格一致 (如果你希望动手 “玩一玩”，可以直接移步原 repo)。例如，我们输入 `+` 输出总共 $n = 10$ 个 tokens。你应该保持这个行为不变。
> 并行化带来的计算顺序调整可能会轻微影响整个神经网络的激活函数函数值，但只要最终输出的 token 序列 (理解为文本) 一致，我们就认为正确。相比于串行程序，在一个 k 个处理器的计算机上，除去模型加载时间，你应当在有较多轮推理时得到近似线性 (k倍) 的加速比。

Online Judge 评测时，$k \leq 4$。对于真实的神经网络训练/推导系统，GPU 这类 SIMT 的大规模并行处理器在能耗比上相比 CPU 有巨大的优势。回顾课程中讲解 SIMT 时，一个线程束共享一个 Program Counter，控制多个线程 “同步” 执行指令。此时，针对大矩阵、向量的 load/store，一个线程束就会生成一个非常长的内存 load (coalesced memory access)，相比 CPU “碾碎了” 的内存-缓存系统，具有高得多的电路比和能效比。

在这个实验中，你需要静态分配好线程 (例如 4 个 workers)，然后由这些线程完成计算任务。Online Judge 测试时的线程库与同学们实验代码中的完全一样。**我们只会复制你的单个文件 gpt.c 进行评测** 。

4. 实验指南

4.1 GPT-2 工作原理

首先， `gpt.c` 的源代码就是最好的老师！它实现了对神经网络的 “真正数学严格” 的描述——如果我们看 PyTorch的代码，其中会涉及许多内置的算子，你对其中的实现其实并不 100% 理解——但对于 C 代码这种语义 “扁平”的语言来说，你真的可以完全理解它！此外，我们给一个外链，也还推荐 Understanding Deep Learning。

4.2 寻找并行的机会

如何优化你的代码？在开始任何优化之前，首先看 D. E. Knuth 的名言：

> Premature optimization is the root of all evil.

“Premature” 不仅指你不应该在实现代码的过程中做 “任意” 的优化，它们可能收效甚微，但大幅破坏了你代码的可读性、可维护性等。“Premature” 同样指你不应该随意地去动你的代码，然后去做一些想当然的优化。同样，这些优化可能破坏可读性，而且很可能没有什么效果。

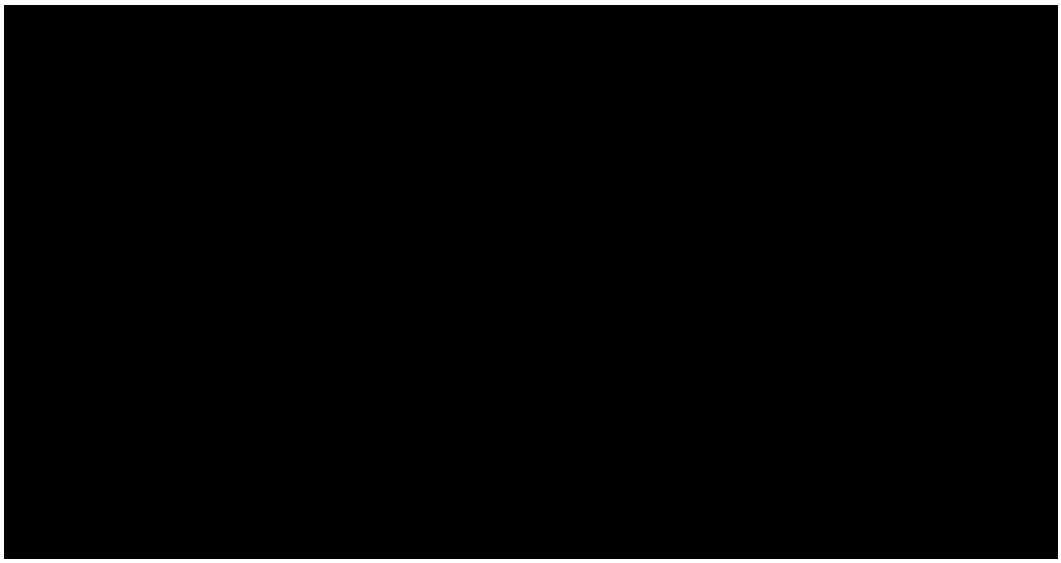
> 💡怎么做出正确的优化?
> 给大家一个提示：Everything is a state machine. 如果顺着这个思路出发，我们应该怎么考虑性能优化这个问题?

答案是：我们应该观察状态机的执行。根据二八定律，绝大部分的时间应该是消耗在少部分操作上的。因此，你首先需要理解程序的哪个部分消耗了最多的时间——针对性地，我们也应当试图优化 (并行化) 这些代码——你对占用 1% 时间的代码，即便优化到极致使运行时间完全为 0，你的收益也近乎为零。

当然，你需要工具：profiler，用于采样状态机的执行，“密集” 的采样就能得到程序消耗时间的序列。

> In software engineering, **profiling** (“program profiling”, “software profiling”) is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.

例如，Linux perf 工具是一个强大的性能分析工具，它提供了一系列用于分析程序性能和跟踪系统事件的功能。这个工具是内核中的一部分，利用了 Linux 内核中的性能计数器子系统。Perf 可以帮助开发者和系统管理员监控整个系统的性能，包括硬件和软件层面。Perf 工具能够测量软件事件如函数调用和程序执行时间，并且能够监控硬件事件比如 CPU 周期数、指令数以及缓存命中和未命中的情况。这使得它非常适合于性能调优和寻找瓶颈。此外，perf 支持事件的实时追踪，能够记录和报告系统运行中的各种事件，如上下文切换、系统调用、页错误等。这些功能使得 perf 成为 Linux 系统性能分析和问题诊断的重要工具。



4.3 使用线程库实现并行

如果你不知道怎么实现并行，可以从我们课堂的代码入手，例如生产者-消费者问题。

我们已经提到，生产者-消费者可以解决大家大部分并行编程问题——这个自然不例外。