DeepCPU: Serving RNN-based Deep Learning Models 10x Faster

综述报告

专业：计算机技术 班级：硕1807 姓名：张超 学号：M201872992

1. 关于论文

该论文的作者有：Minjia Zhang、Samyam Rajbhandari、Wenhan Wang、和 Yuxiong He。该论文于 2018 年 7 月份发表在会议 USENIX ATC 18 上。

1.1 期刊介绍

自 1975 年以来，高级计算系统协会 USENIX 汇集了在计算机世界前沿工作的工程师，系统管理员，科学家和技术人员。USENIX 会议已成为介绍和讨论有关计算系统各方面发展的最先进信息的重要会议基础。

USENIX 会议的主要任务有四点：

（1） 培养技术卓越和创新；

（2） 支持和传播最新的计算机系统结构研究；

（3） 为讨论技术问题提供了一个开放的论坛；

（4） 鼓励计算机社区的开展活动

USENIX 是一个非营利组织，致力于支持先进的计算系统社区并进一步扩展创新研究的范围。USENIX 以组织会议和出版研究而闻名，USENIX 最大的优势在于建立计算机系统结构社区。

该会议通常每届录用 30 篇左右论文，录取率在 15%左右，是操作系统、体系结构方面最好的会议之一。会议于 2018 年 7 月 11-13 日在美国波士顿举行。文章由 ERC Grant 339539 (AOC).部分支持。

2. 论文的工作

2.1 发现问题

2.1.1 RNN 服务的问题

深度学习是一个快速发展的领域，影响很大。传统的 FFNN 前馈神经网络输入输出是独立于彼此的，但是对于某些任务而言，不是很合适使用这种网络，例如，预测一个句子的下一个单词，需要了解上下文。RNN 可以解决这个问题，输出取决于先前的计算，有点类似于人类的学习，带点记忆性的。RNN 在很多问题比如自然语言处理，机器翻译，机器阅读理解上表现很出色。

有很多 RNN 模型可以用于在线服务，例如网络搜索、会话机器人等。这些服务都需要密级的计算，可能会导致服务延迟的情况。根据 SLA 服务等级协议，如果一个服务的延迟不超过几十毫秒，那么这个服务是用户可以接受的。如果让用户等待的时间再长，那么会影响用户体验。

传统的 RNN 在线服务就存在这种问题，因为需要大量的计算，导致服务的时间太长了，所以不得不放弃某些服务。

2.1.2 CPU 计算问题

Xeon E5-2650 型号的 CPU 是一种顶级的 CPU，它的计算峰值可以达到 1.69T flops（每秒钟计算的浮点数数量），但是当这种顶级的 CPU 用于 RNN 的计算时，它的计算性能仅仅达到 30G flops，不到它本身性能的 2%，也就是说在进行 RNN 计算的时候，CPU 的性能并没有发挥出来，这也是导致性能差的根本原因。

2.2 分析问题

2.2.1 数据重用性问题

无论是什么神经网络，它的内部都是大量的矩阵相乘的计算。从输入层到输出层，还是说隐藏层里面，都要做大量的矩阵相乘的计算。在矩阵相乘计算的过

程中，数据重用性差。什么是数据重用性，本文中指出，RNN 在进行矩阵相乘的计算过程中，需要把数据在 CPU 的三级缓存和二级缓存中进行移动？因为是多核计算，一个核算一部分，两个核通过二级缓存来交流，一个核的计算的中间结果要复制到缓存上，让另一个核来取。由于现在的 CPU 的计算能力很强，那么这个移动的效率就成了瓶颈，拖累了 CPU 的计算。

对于 RNN 来说，在批量很小的服务场景中，情况确实如此。考虑到一个矩阵相乘，我们假设开始计算时输入和输出都位于 L3 缓存，那么输入和输出必须从 L2 缓存到 L3 缓存至少读一次，输出的话必须从 L2 缓存到 L3 缓存至少存一次。因此在矩阵相乘中 L2 缓存最大可能数据重用。同样的，在 LSTM 中，以 B = 1 为例：基于测量的 250 GB / s L3 带宽，Xeon E5 - 2650 机器上 LSTM 的最佳可实现性能至多为 125G flops。这还不到机器峰值 1.69G flops 的 8 %。

在传统的 parallel – GEMM 计算中，整个序列中都没有数据的重用。在服务期间，RNN 的权重矩阵在整个序列中保持不变，但是现有的解决方案没有利用这一点来优化数据重用。更确切地说，用于执行 MMs 的并行 GEMM 不知道这种在整个序列中的重用。在序列的每个步骤中，权重矩阵可以从 L3 缓存加载到 L2 缓存。然而，通过利用这种数据重用可以提高无线网络的性能。

2.2.2 矩阵相乘的优化问题

优化矩阵相乘的划分。传统的 parallel–GEMM 善于优化具有显著数据重用的大型计算的性能，它们利用循环拼接的方式隐藏了二级缓存到三级缓存的移动成本，相比之下，RNN 中重用量很小，对于大多数服务情况来说，通常是 1-10 之间的最小值，这不足以忽略数据移动的成本，及时矩阵相乘在 L3 缓存中。在没有大规模的重用的基础上，GEMM 的性能受到 L3 缓存到 L2 缓存移动成本的限制，GEMM 在最小化数据移动方面是次优的。

更具体的说，现代 CPU 上的 L3 缓存提供给多个 L2 缓存，这些缓存是每个内核专用的。在 RN 以及 N 计算期间，多个内核可能需要一些数据，导致来自 L3 缓存的同一数据的多次传输，因此 L3 和 L2 高速缓存之间的总数据移动取决于矩阵相乘计算的空间划分以及到内核的映射。例如我们将 MM 计算分成两个核，第一个核计算输出矩阵 C 的上半部分，第二个计算下半部分，则输出矩阵必

须复制到两个内核的 L2 缓存上，因为整个矩阵 B 需要计算矩阵 c 的两个半部分。或者，如果计算水平分割，则输入矩阵 A 必须复制到两个内核的 L2 缓存上。不同的分区显然会导致不同数量的数据重用。parallel - GEMM 并不总是产生最大化数据重用的部分。专为小矩阵而设计的库也是不够的，因为有些库只关注顺序执行，而其他库只关注小到足以容纳在 L1 缓存。

对 RNN 中矩阵相乘的优化可以使得数据在缓存之间的移动开销最小，从而提高系统整体的性能。

2.2.3 优化并行计算

由于 RNN 本身计算的特点，即下一次计算需要用到上一次计算的结果，整个计算流程具有上下文关联的特性，这个特性导致了 RNN 的计算无法很好的并行执行，才会导致计算的速度很慢。

2.4 挑战和策略

2.4.1 挑战

由于优化旋钮和时间表的空间很大，要找一种优化的 RNN 执行实现，最大限度的提高数据重用，同时使用低级的硬件资源，这是一种挑战。实际上，通过循环排列、循环融合、循环取消滚动、展开因子选择、循环平铺、切片大小选择、MM 重新排序、矢量化、寄存器平铺、登记切片大小选择、并行循环选择、并行粒度选择、线程到内核映射等，可以获得无限数量的有效选择。以及它们的组合。此外，启用这些优化旋钮并为所有选择创建一个时间表生成器是一项不平凡的工程任务。此外，最佳选择取决于硬件架构和 RNN 参数:单个解决方案不会适用于所有情况，优化的时间表需要以有效的方式逐个调整。所有这些都使得这个问题在实践中具有挑战性。

2.4.2 策略

为了克服这些困难，本文明智的定义了搜索空间，并且确定了最重要的提高

数据准确度的技术。使得在一组选择优化和时间表中进行高效的搜索，以获得最佳的 RNN 性能，我们将整个的优化管道构建成了一个库，称之为 DeepCPU。

优化最重要的一个开端是定义一个简洁的搜索空间，我们在两个点上开发这个空间：

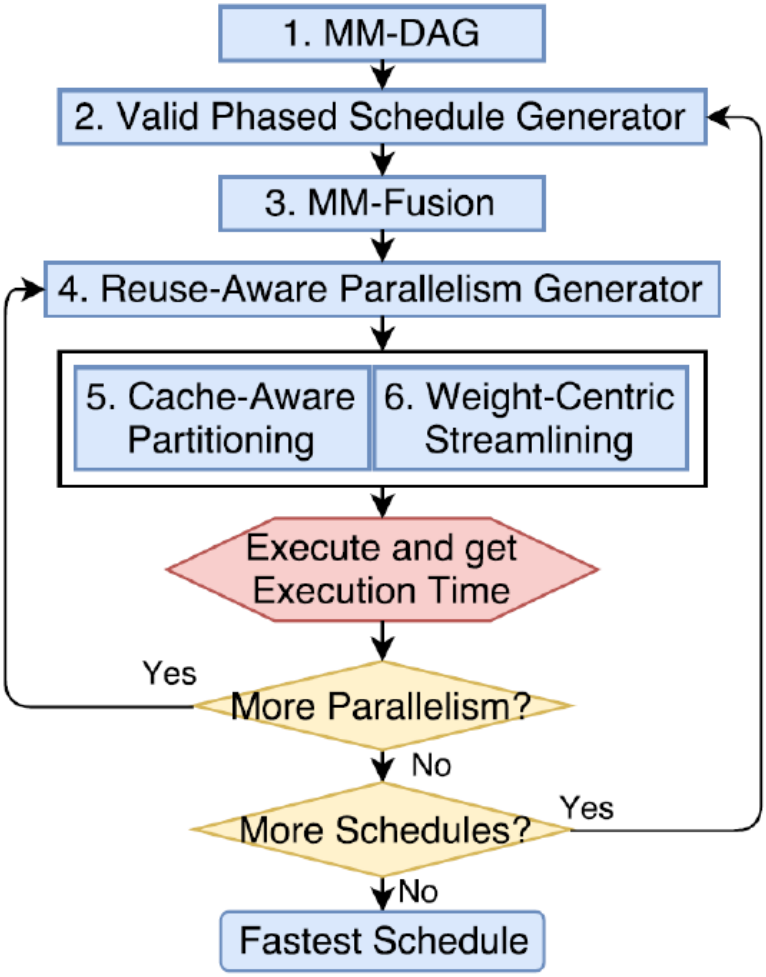
（1）我们识别最关键的矩阵相乘，并建立连接它们的计算图模型，以捕捉一阶的影响。我们可以通过构造一个矩阵乘法有向无环图（MM-DAG）来表示 RNN 计算，其中每个节点表示一个矩阵相乘，边表示它们之间的依赖关系。这个模型允许我们使用矩阵相乘作为基本模块来构建时间表，捕捉关键计算，同时抽象掉其它的低级细节。

（2）我们可以利用 RNN 的迭代特性和其它特性，修剪搜索空间中等效的重复数据，删除不能被跟踪的时间表（省去不需要计算的步骤），而不是检查 MM-DAG 的所有有效时间表。然后我们开发四种技术来有效的提升 RNN 的数据局部性（详细内容会在之后讲到）。

在 DeepCPU 的设计中，我们将性能分析（搜索空间）和经验校准（测量局部性和并行性的综合影响）结合起来，前者有效的减少了搜索空间，减少运行的时候的冗余；后者可靠的完成复杂的软件和硬件交互，这种结合增加了有效性和效率。

2.5 DeepCPU 的设计和实现

DeepCPU 的实现，核心就是这个流程图，流程 1~6 的优化过程，就是 DeepCPU 的全部优化过程。



2.5.1 MM-DAG

MM-DAG 即是矩阵相乘的有向无环图，DeepCPU 将 RNN 的计算建模成了一个矩阵计算的有向无环图。并优化计算的时间表，给了一个 MM-DAG，一个有效的调度可以满足所有依赖的执行顺序。

这些执行顺序的矩阵相乘可以分为两类：

（1）在一个时间步上有依赖关系的，称为依赖时间的阶段；

（2）如果不包含依赖的 MM，我们称为独立于时间的阶段。

2.5.2 Valid Phased

将整个 RNN 的计算抽象出有向无环图后，可以有效的减少最快计算矩阵相乘的搜索空间，我们提出三条规则来修剪搜索空间，删除次优的和重复的时间表：

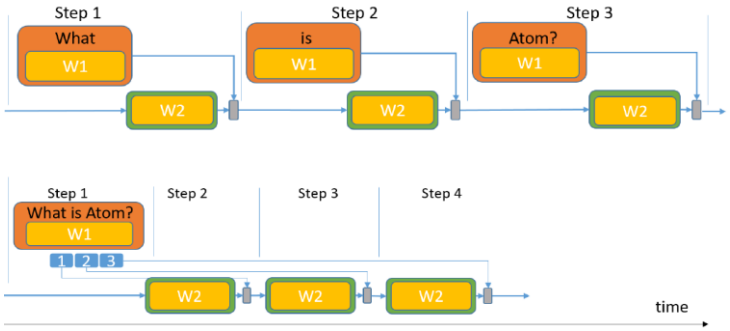
（1）依赖时间的阶段必须跨时间步长对称。由于 RNN 计算跨时间步是相同的，执行每个时间步的最快时间表也应该是相同的。

（2）如果两个连续的阶段是相同类型的，那么这两个阶段之间必须有相关性。如果不存在依赖性，那么这个时间表相当于另一个时间表，其中一个阶段包括两个阶段中的所有 MM。

（3）我们计算所有相关相位之前的时间无关相位，找出具有连续顺序相同类型的相位，增加了重复使用率。

2.5.3 MM-Fusion

矩阵相乘的合并，如果两个矩阵共享了一个公共的输入矩阵，那么可以合并成一个矩阵。矩阵的融合提高了数据重用，可以利用它来提高性能和可扩展性。



2.5.4 Parallelism Generator

重用感知的并行生成器，这一部分提高了并行的计算能力，但是也增加了数据的移动。这一部分讨论局部性和并行性的关系，以及我们的并行策略。如何并行化单个的MM呢？执行具有最大可用并行度的MM并不总是性能的最佳选择。随着并行度的增加，输入或输出必须跨多个 L2 专用缓存复制，从而增加了总体数据移动。一旦并行度达到某个阈值，性能就会受到数据移动而不是计算吞吐量的限制。在某些并行之后，MM 性能下降。找到最佳并行级别而不是应用使用所有可用内核的常识是至关重要的。

如何并行化并发 MM？一个阶段中的多个 MM 没有任何依赖性。DeepCPU 以并行 gems 并行方式执行它们，其中多个 MMs 并行执行，每个 MM 并行执行。

例如，为了在 P 核上计算两个独立的 MMs，M1 和 M2，我们并行运行 M1 和 M2，每个都使用 P/2 核。

这与序列中的并行 gems 形成对比，在序列中，我们首先使用 P 核运行 M1，然后使用 M2。跨多个内核并行化 MM 会增加从 L3 缓存到 L2 缓存的数据移动。相比之下，在多个划分的内核组中并行执行多个 MMs 允许每个组在一个唯一的 MM 上工作，而不需要跨它们进行数据复制，从而在保持相同并行级别的同时提高了数据重用。

2.5.5 Cache Partitioning

私有缓存感知分区（PCP）我们开发了 PCP，这是一种新的私有缓存感知分区策略，用于跨多协议执行 MMs，以优化阶段内和阶段间的 L2 重用。PCP 为优化数据移动提供了一种有原则的方法：

对于具有平行度 P 的给定 MM，我们显示 PCP 产生计算空间的 P 分割，从而使 L3 和 L2 缓存之间的总数据移动最小化。DeepCPU 采用 PCP 为每个并行配置生成一个局部优化调度，而不需要根据经验校准不同分区并测量它们的性能，从而最大限度的从 L2 缓存中重用数据。

2.5.6 Weight-Centric Streamlining

以权重量为中心的简化 WCS 是我们的实现方式，它支持完整的 PCP，支持在整个 TDP 中重用重量矩阵。对于给定的并行度，PCP 产生分区，使得计算分区所需的权重适合于单个内核的 L2 高速缓存，允许权重在 TDPs 上从 L2 高速缓存重用，而不被驱逐。然而，为了确保这种重用，计算必须在权重所在的位置进行，即并行分区和执行它们的内核之间的映射不能跨 TDPs 改变。

为此，我们使用 OpenMP 创建一个跨越整个 RNN 计算序列的并行区域。并行度等于计划中所有阶段的最大并行度。并行区域中的每个线程负责在每个阶段执行至多一个并行分区。在并行度小于线程数的阶段，一些线程可能会保持空闲。每个线程 ID 都映射到一个唯一的分区 ID，并且这个映射在 TDPs 上是相同的。

本质上，我们交替顺序循环和平行区域，使得顺序循环在平行区域内的主要

优点在两点：

（1）将每个矩阵相乘放在一个核上。我们在整个计算过程中创建了本地线程 ID 和全局线程 ID 之间的唯一映射，这确保了在整个序列中，MM 分区总是在同一内核上执行。

（2）它减少了创建平行区域的开销。不是在 RNN 序列的每个步骤中打开和关闭平行区域，我们只为整个计算创建一次平行区域。

本文比较了对于不同大小的 MMs，运行具有/不具有 WCS 的并行 GEMM 和 PCP 序列的性能。后两者始终优于前两者，但只有与 WCS 一起使用，PCP 的全部优化功能才得以实现。

2.5.7 小结

通过第一步到第六步的优化，CPU 可以大大的提高，并且这个优化是循环执行的。在实测中，DeepCPU 只需要几百次的校准运行就能找到最佳的执行时间表。而且这个优化和校准的过程通常在模型构建的时候被调用一次，然后在以后的服务中，可以被重复利用。

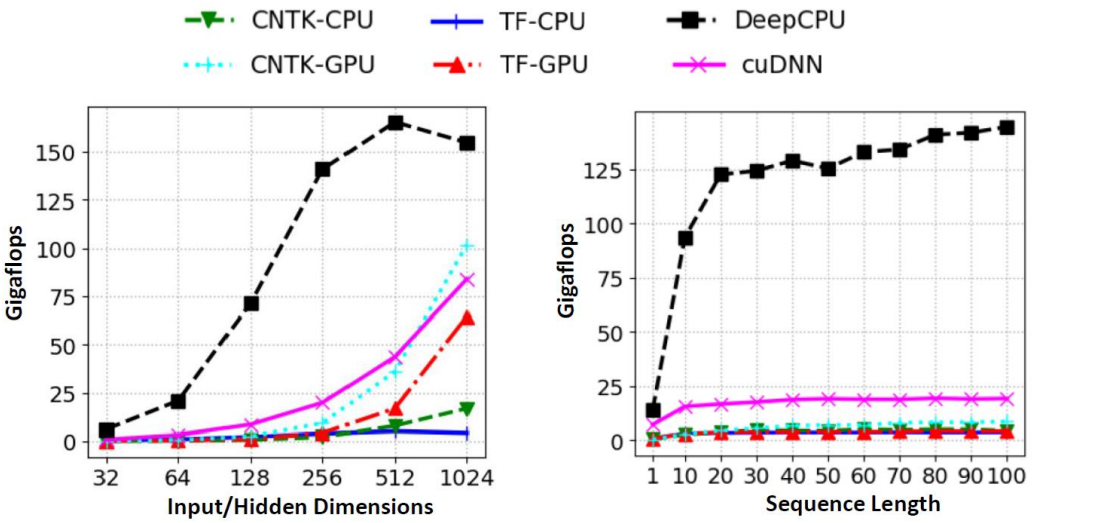
2.6 测试

2.6.1 DeepCPU PK CPU

Model parameters			LSTM speedup		GRU speedup	
input	hidden	seq. len.	TF	CNTK	TF	CNTK
64	64	100	26	81	11	36
256	64	100	34	93	17	45
1024	64	100	45	60	23	39
64	256	100	34	37	22	38
64	1024	100	28	4.6	17	5.8
1024	1024	100	42	3	23	4.8
256	256	1	14	16	17	19
256	256	10	21	18	21	24
256	256	100	38	28	24	28

在 LSTM 网络上，DeepCPU 平均加速 20~30 倍；在 GRU 网络上，DeepCPU 平均加速 15~25 倍。

2.6.2 DeepCPU PK GPU



Batch Size = 1, Sequence Length = 100

Batch Size = 1, Input/Hidden Dimension = 256

在性能上，DeepCPU 也远远高于 CPU。

3. 论文的贡献

（1） 使 RNN 在 CPU 上的服务速度提高了 10x，进而使得在线提供某些 RNN 服务由不可能变成可能。

（2）使 CPU 在 RNN 服务上打败 GPU。

（3）节约了机器成本，相同的服务，最初要使用 10k 台机器，现在减少到几百个，节约了数百万美金的成本。

4. 总结

对于一个普遍的计算问题，在计算性能上，GPU 肯定是强于 CPU 的，但是，GPU 的计算性能不可能在所有的方面都优于 CPU，在某些问题上，GPU 的计算也有不足之处。

论文的研究不具有普遍性，但是具有很强的针对性。论文的作者正是抓住了 GPU 在 RNN 计算上的不足之处，结合 RNN 本身的特性，对 CPU 进行优化，优化的结果可以使得 CPU 在 RNN 的计算上可以打败 GPU。