只改一行代码
在第四代至强® 可扩展平台上
高效微调优化 ChatGLM-6B

作者：英特尔公司 夏磊

大语言模型的应用与微调优化必要性

ChatGPT 的横空出世开启了大语言模型（LLM）的普及元年，BERT、GPT-4、ChatGLM 等模型的非凡能力则展现出类似通用人工智能（AI）的巨大潜力，也因此得到了多行业、多领域的广泛关注。为加速这些大模型与特定领域的深度融合，以及更好地适应特定任务，基于任务特性对这些模型进行定制化微调至关重要。然而，它们庞大的参数使得用传统方式对大模型进行调优面临诸多挑战，不仅要求相关人员熟练掌握微调技巧，还需要付出巨大的训练成本。近年来，出现了参数高效微调（Parameter-Efficient Fine-Tuning, PEFT）和提示微调（Prompt-tuning）技术。这些技术因其成本更低、应用方式更简单便捷，正在逐渐取代大模型传统调优方法。

本文结合目前在中文应用场景中具有出色表现的开源预训练大模型 ChatGLM-6B，介绍如何通过对其开源 Prompt-tuning 代码进行极少量的修改，并结合第四代英特尔® 至强® 可扩展处理器[1]的全新内置 AI 加速引擎——英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extension，简称英特尔® AMX）及配套的软件工具，来实现高效、低成本的大模型微调。

基于英特尔® 架构硬件的微调优化方案

本文通过以下三个方面实现了基于第四代英特尔® 至强® 可扩展处理器的 ChatGLM 高效微调优化：

1. **借助英特尔® AMX，大幅提升模型微调计算速度** AMX 是内置于第四代英特尔® 至强® 可扩展处理器中的矩阵乘法加速器，能够更快速地处理 BFloat16（BF16）或 INT8 数据类型的矩阵乘加运算，从而显著提升模型训练和推理的性能。

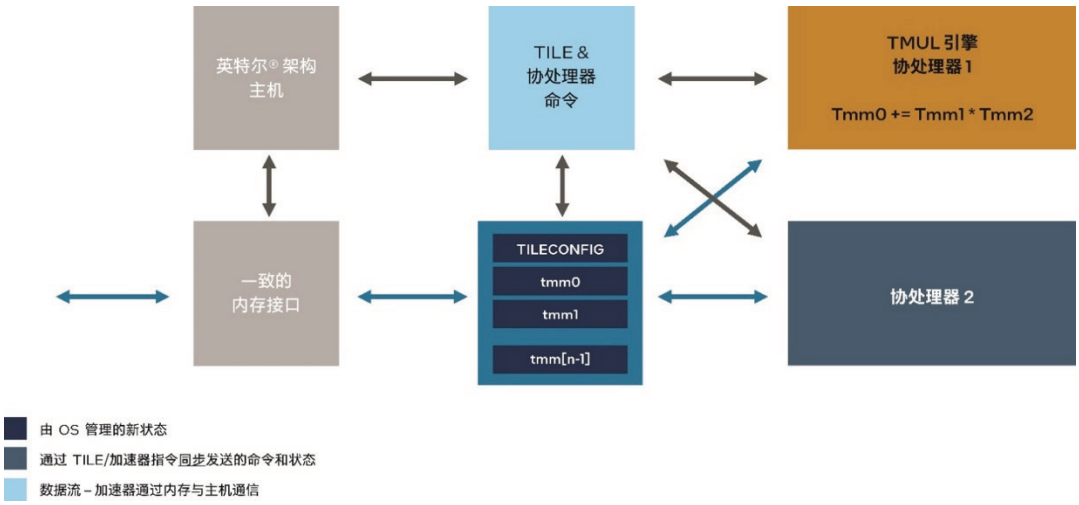

图 1. 英特尔® AMX 技术架构

目前，现行的 PyTorch 框架中，已经可以通过具备 BF16 自动混合精度功能自动实现对 AMX 加速器的利用。

就 ChatGLM-6B 而言，其开源微调代码的 autocast_smart_context_manager() 函数，也已具备对 CPU 自动混合精度的支持。因此，只需在启动微调时加入 CPU 自动混合精度的使能参数即可直接利用英特尔® AMX 带来的优势。

```
2601.     def autocast_smart_context_manager(self, cache_enabled: Optional[bool] = True):
2602.     """
2603.     A helper wrapper that creates an appropriate context manager for `autocast` while feeding it the desired
2604.     arguments, depending on the situation.
2605.     """
2606.     #print(self.use_cpu_amp)
2607.     if self.use_cuda_amp or self.use_cpu_amp:
2608.       if is_torch_greater_or_equal_than_1_10:
2609.         ctx_manager = (
2610.           torch.cpu.amp.autocast(cache_enabled=cache_enabled, dtype=self.amp_dtype)
2611.     if self.use_cpu_amp
2612.           else torch.cuda.amp.autocast(cache_enabled=cache_enabled, dtype=self.amp_dtype)
2613.         )
2614.       else:
2615.         ctx_manager = torch.cuda.amp.autocast()
2616.     else:
2617.       ctx_manager = contextlib.nullcontext() if sys.version_info >= (3, 7) else contextlib.suppress()
2618.
2619.     return ctx_manager
```

图 2. 通过 trainer.py 中的 autocast_smart_context_manager() 函数，
在 ChatGLM-6B 开源 prompt-tuning 目录下实现对 CPU 和 GPU 的自动混合精度支持

具体方法是在启动微调的 train.sh 脚本时做如下修改：

```
python3 main.py \
 --do_train \
 ...
 --half_precision_backend cpu_amp \
 --bf16
```

2. 结合英特尔® MPI 库充分利用处理器架构特点和多核配置，发挥 CPU 的整体效率

第四代英特尔® 至强® 可扩展处理器最多可拥有 60 个内核。这些内核通过 4 个集群（cluster）的方式进行内部组织。理论上，当多个处理器内核并行处理一个计算任务并需要共享或交换数据时，同一个集群内的内核之间的通信时延较低。因此，在使用 PyTorch 框架进行模型微调时，我们可以将同一个集群上的内核资源分配给同一个 PyTorch 实例，从而为单个实例提供更理想的计算效率。此外，通过利用 PyTorch 的分布式数据并行（Distributed Data Parallel, DDP）功能，还可将两个 CPU 上的 8 个集群的内核资源汇集在一起，充分发挥整体效率。

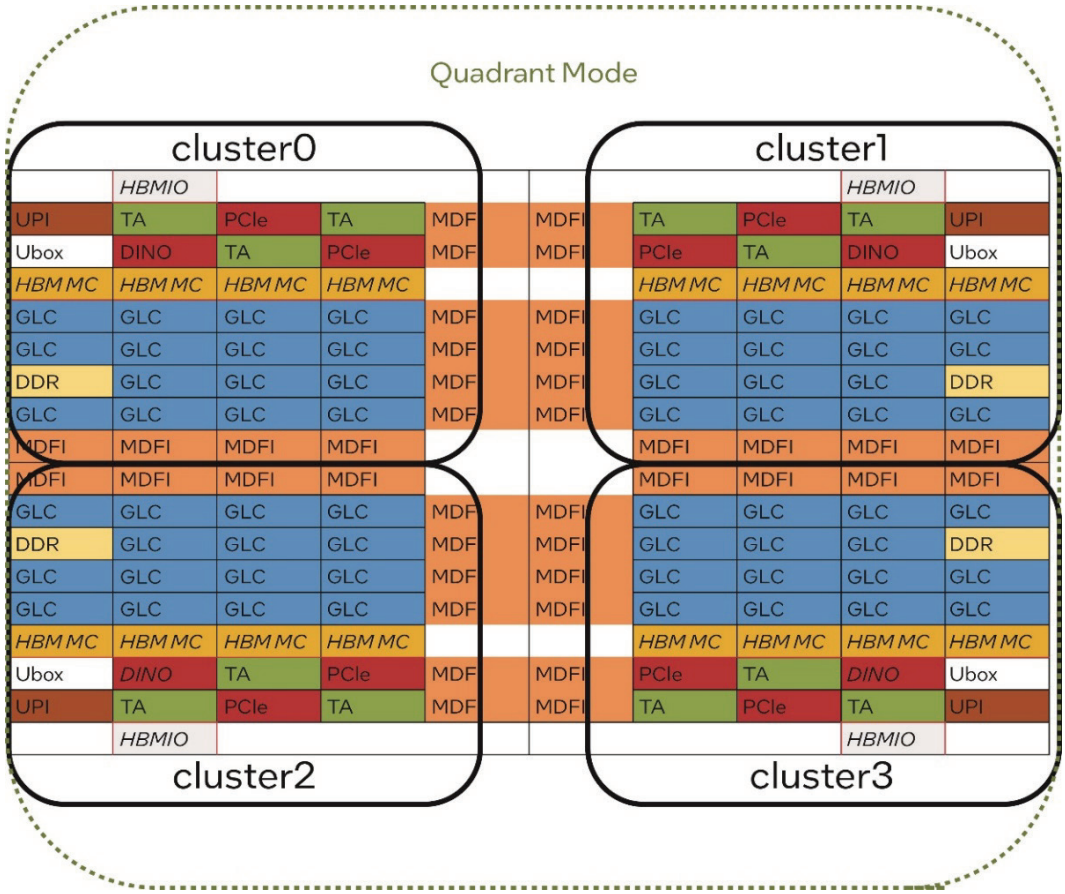

图 3. 第四代英特尔® 至强® 可扩展处理器的内部集群（cluster）架构

为实现从应用程序代码到数据通信的整体简化，PyTorch 框架支持多种分布式数据并行后端（backend），其中 MPI 后端方式能够很好地满足我们的优化需求。

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✓
recv	✓	✗	✓	?	✗	✓
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✓
scatter	✓	✗	✓	?	✗	✓
reduce_scatter	✗	✗	✗	✗	✗	✓
all_to_all	✗	✗	✓	?	✗	✓
barrier	✓	✗	✓	?	✗	✓

图 4. PyTorch 支持的多种分布式数据并行的后端（来源：PyTorch 官网[ii]）

但是，通过 pip 或 conda 来安装的预编译 PyTorch 二进制包中并未将 MPI 的后端作为缺省功能编译。因此，我们需要安装 MPI 协议工具库并通过手工编译来获得对 MPI 后端的支持。

英特尔® MPI 库[iii] 是一个实现 MPICH 规范的多结构消息传递库，使用该库可创建、维护和测试能够在英特尔® 处理器上实现更优性能的先进和复杂的应用。它采用 OFI 来处理所有通信，能够提供更高的吞吐量、更低的时延和更简单的程序设计。

以下是基于英特尔® MPI 库的 PyTorch 编译步骤:

#下载英特尔® MPI 库并安装

```
wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/718d6f8f-2546-4b36-b97b-bc58d5482ebf/l_mpi_oneapi_p_2021.9.0.43482_offline.sh
bash l_mpi_oneapi_p_2021.9.0.43482_offline.sh
source your_installation_path/vars.sh
```

安装 PyTorch 编译依赖包

```
pip install numpy pyyaml mkl mkl-include setuptools cmake cffi typing
pip install mkl_include
```

#下载 PyTorch 源码并完成编译、安装

```
git clone --recursive https://github.com/pytorch/pytorch.git
cd pytorch
python setup.py install
```

在获得了支持 MPI 后端的 PyTorch 后，只需按如下方法在 ChatGLM Prompt-tuning 目录下的 main.py 修改一行代码：

将 dist.init_process_group(backend='gloo', world_size=1, rank=0) 改为 **dist.init_process_group(backend='mpi')**

```
367. if training_args.do_train:
368.     checkpoint = None
369.     if training_args.resume_from_checkpoint is not None:
370.         checkpoint = training_args.resume_from_checkpoint
371.     # elif last_checkpoint is not None:
372.     #exit()
373.     model.gradient_checkpointing_enable()
374.     model.enable_input_require_grads()
375.     os.environ['MASTER_ADDR'] = 'localhost'
376.     os.environ['MASTER_PORT'] = '12355'
377.     dist.init_process_group(backend='gloo', world_size=1, rank=0)
378.     train_result = trainer.train(resume_from_checkpoint=checkpoint)
379.     # trainer.save_model()  # Saves the tokenizer too for easy upload
```

图 5. 修改前的 main.py

```
367. If training_args.do_train:
368.     checkpoint = None
369.     if training_args.resume_from_checkpoint is not None:
370.         checkpoint = training_args.resume_from_checkpoint
371.     # elif last_checkpoint is not None:
372.     #     checkpoint = last_checkpoint
373.     model.gradient_checkpointing_enable()
374.     model.enable_input_require_grads()
375.     world_size = int(os.environ['PMI_SIZE'])
376.     world_rank = int(os.environ['PMI_RANK'])
377.     dist.init_process_group(backend='mpi', world_size= world_size, rank= world_rank)
378.     train_result = trainer.train(resume_from_checkpoint=checkpoint)
379.     # trainer.save_model()  # Saves the tokenizer too for easy upload
```

图 6. 修改后的 main.py

3. **利用至强® CPU Max 系列集成的 HBM 满足大模型微调所需的大内存带宽**

基于 Transformer 的大模型，由于参数、训练数据和模型规模的复杂程度较高，因此内存复杂度通常是 O(n^2)。这意味着这些大模型需要足够大的内存带宽支持才能获得更好的运行性能。英特尔® 至强® CPU Max 系列[iv]，配备 64 GB 的 HBM2e 高带宽内存，为在 CPU 上高效运行大模型提供了高达~1TB/s 的内存带宽支持。

该 CPU 集成的 HBM，能够在 3 种模式下灵活配置：

- HBM-Only 模式——支持内存容量需求不超过 64 GB 的工作负载，具备每核 1 至 2 GB 的内存扩展能力，无需更改代码和另购 DDR，即可启动系统。
- HBM Flat 模式——可为需要大内存容量的应用提供灵活性，通过 HBM 和 DRAM 提供一个平面内存区域（flat memory region），适用于每核内存需求 >2 GB 的工作负载。可能需要更改代码。
- HBM 高速缓存模式——为内存容量 >64 GB 或每核内存需求 >2GB 的工作负载提供更优性能。无需更改代码，HBM 将用作 DDR 的高速缓存。

针对 ChatGLM-6B 微调，试验结果显示：与其他两种模式相比，**HBM 高速缓存模式**在性能和使用方便性方面均更胜一筹。

在英特尔® 至强® CPU Max 系列产品上，结合之前的两项优化，我们可以通过以下命令行启动 ChatGLM-6B 微调：

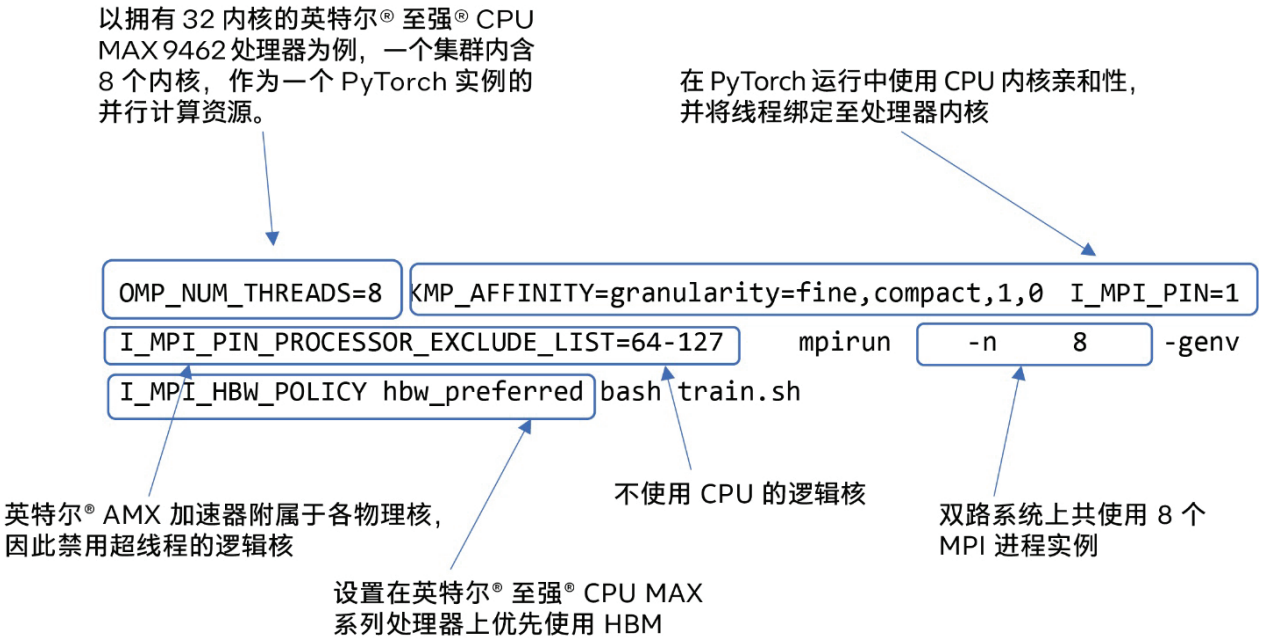

图 7. 在拥有 32 个物理核的英特尔® 至强® CPU Max 9462 双路服务器上启动微调

优化结果

通过以上简单软、硬件综合优化，无须采用昂贵的 GPU 硬件，即可实现对 ChatGLM-6B 模型的高性能微调。

注： 以上代码修改需要配合 python 工具包 accelerate 0.18.0 和 transformers 4.28.0。

作者简介：

夏磊，英特尔（中国）有限公司人工智能首席工程师，拥有近 20 年的人工智能从业经验，在软件算法、自动控制和工程管理等领域积累了丰富经验。

[i] https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors-product-brief.html
[ii] https://pytorch.org/docs/stable/distributed.html
[iii] https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/mpi-library.html
[iv] https://www.intel.cn/content/dam/www/central-libraries/cn/zh/documents/2023-03/23-cmf54-xeon-cpu-max-series-product-brief.pdf
[v] https://www.intel.cn/content/dam/www/central-libraries/cn/zh/documents/2023-03/23-cmf54-xeon-cpu-max-series-product-brief.pdf