AI PC で将来の AI 開発を試す

Tony Mongkolsmai インテル コーポレーション

インテル® GPU 向けのオープンソース OpenAI* Triton バックエンドを構築して実行する方法

ディープラーニング（DL）AI ソリューションが大規模になるにつれて、AI 開発者が直面する最大の課題の 1 つが、パフォーマンスに優れたモデルを効率良く作成する方法です。これまで、AI モデルの開発者はカーネルを C++ で記述し、pybind を使用して Python* から実行する必要がありました。このため、AI 開発者は DL カーネルとモデルを理解するだけでなく、C++ とカーネル開発に関連する C++ テンソルの抽象化についても学ぶ必要がありましたが、この課題への取り組みとして、OpenAI* は、開発者が効率的な GPU コードを作成できるようにする、オープンソースのドメイン固有の Python* に似たプログラミング言語およびコンパイラーである Triton をリリースしました。

OpenAI* Triton の概要

Triton は、機能とパフォーマンスの点で C++ と Python* の中間層を提供します。目標は、DL 開発者が複数層のソフトウェア・スタックを実装することなく、最適化されたカーネルを構築できるようにすることです。Triton はネイティブ Python* ではありませんが、Python* コードと同じソースファイル内で開発できるため、コード管理がはるかに容易になります。

DL は本質的に大規模な並列計算を扱うため、Triton は個々の要素ではなくデータのブロックを扱うように設計されています。これにより、変数を定義することで、データのセットを表す言語の構文が簡素化されます。

```
result = x + y
```

例えば、上記のコードは `x` ベクトルと `y` ベクトルの要素単位の加算を行い、出力ベクトルを `result` 変数に書き込みます。

構文の簡素化以外の Triton の利点は、パフォーマンスです。Triton は言語であるとともにコンパイラーでもあるため、構文を特定のハードウェア設計にマップすることができます。ニューラル・ネットワークは大規模であり、実行するには大量のメモリー操作が必要です。Triton は、開発者とコンパイラー・バックエンドが、ネイティブ Python* で利用できるものよりも最適化されたキャッシュ階層とメモリーの使い方を言語の操作にマップできるよう意図的に定義されています。

これは Triton で実現される機能のほんの一部であり、ほかにも DL 開発者にとって不可欠な操作が含まれています。Triton の詳細は、OpenAI* の Triton リソース（英語）を参照してください。

インテル® Core™ Ultra プロセッサー上の Triton

OpenAI* はソフトウェア開発の課題を簡素化します。開発者は、**場所と時間に拘束されることなく**コードを開発したいと考えています。ワークステーション・レベルのラップトップを利用することもできますが、持ち運びに苦労するのが目に見えています。そこで、強力な統合 GPU とニューラル・プロセシング・ユニットを備えた、新しいインテル® Core™ Ultra プロセッサー・ベースのラップトップを選択し、新しい AI PC で Triton を試してみることにしました。

インテル® Core™ Ultra プロセッサー向け Triton のコンパイル

Triton はオープンソースですが、インテル® GPU のサポートは開発中であり、Triton のメイン・リポジトリーにまだアップストリームされていません。インテルは、ここで Triton バックエンドをオープンソースで（英語）開発しています。ほかのハードウェア・ベンダーと同様に、インテルは Triton のフォークからこのバックエンドを開発しており、この作業のアップストリームに取り組む予定です。GitHub* サイトには、このコードはインテル® データセンター GPU マックス・シリーズのカード上で開発およびテストされているという記述がありますが、任意のインテル® GPU 向けに oneAPI を使用して開発されたソフトウェアは、ほかのインテル® GPU でも動作するはずです。そこで、ソースからコードをビルドして、MSI* Prestige 16 AI Evo ラップトップでどのように動作するか確認してみることにしました（ちなみに、コンシューマーレベルのインテル® Arc™ A770 GPU 上で実行したところ、同様に動作しました）。

システムの設定

次の構成のシステムを使用しました。

- Ubuntu* 23.10、カーネル 6.5.0-17
- インテル® oneAPI ベース・ツールキット 2024.0.1（インストール手順（英語））
- Anaconda*（インストール手順（英語））

Linux* をインストールした後、`/etc/default/grub` を変更する必要があることに注意してください。

```
GRUB_CMDLINE_LINUX_DEFAULT=" quiet splash"
```

を次のように変更します。

```
GRUB_CMDLINE_LINUX_DEFAULT=" quiet splash i915.force_probe=7d55"
```

ここで、`7d55` は統合 GPU の PCI ID です。アップストリームの Ubuntu* 統合ドライバーは最新の iGPU の PCI ID をまだ認識しないため、この変更が必要になります。次に、

```
sudo update-grub
```

を実行した後、再起動します。

Triton の入手とビルド

GitHub* のビルド手順に従うのは難しいため、AI PC で Triton をビルドして実行するために私が実行したコマンドを以下に示します。最初に、環境を設定します。

```
# インテルの XPU 向け Triton バックエンドをチェックアウトしてディレクトリーを変更
git clone https://github.com/intel/intel-xpu-backend-for-triton.git -b llvm-target
cd intel-xpu-backend-for-triton

# Python* 3.10 を使用して conda 環境を作成し環境をアクティベート
conda create --name triton python=3.10
conda activate triton

# oneAPI のビルドおよびランタイム環境を設定
source /opt/intel/oneapi/setvars.sh

# Triton のビルド・インフラストラクチャーに応じてコンポーネントをインストールし
# Clang でビルドするように環境変数を設定
pip install ninja cmake wheel
export TRITON_BUILD_WITH_CLANG_LLD=true
```

Triton は現在 Python* 3.11 と 3.12 では動作しないため、Python* 3.10 を使用しました。`llvm-target` はコードのメインブランチであるため、ブランチを指定しないで実行することもできます。

リポジトリーの指示に従って、ビルドを行います。

```
./scripts/compile-triton.sh
```

残念ながら、このビルドでは CUDA* が見つからないというエラーが発生します。Ubuntu* 23.10 に CUDA* ツールキットをインストールしようとすると、Ubuntu* 23.10 ではそのまま使用できない依存ライブラリーが原因でエラーになります。この問題を回避するには、`/etc/apt/sources.list` を編集して次の行を追加し、Ubuntu* 23.10 APT ソースに 23.04 のリポジトリーを追加します。

```
deb http://archive.ubuntu.com/ubuntu/ lunar universe
```

次のコマンドを実行して、CUDA* をインストールできるようにします。

```
sudo apt update
```

ビルドを再実行すると、`clang` コンパイラーと `clang++` コンパイラーが見つからないという別のエラーが発生します。clang ベースのインテル® oneAPI DPC++ コンパイラーをインストールしているため、`PATH` 変数を設定して、インテル® DPC++ コンパイラーに含まれる clang を指定します。

```
export PATH=$PATH:/opt/intel/oneapi/compiler/latest/bin/compiler
```

コンパイルスクリプトをもう一度実行します。

```
         ********************************************************************************
         Please be careful with folders in your working directory with the same
         name as your package as they may take precedence during imports.
         ********************************************************************************

  !!
    with strategy, WheelFile(wheel_path, "w") as wheel_obj:
  Building editable for triton (pyproject.toml) ... done
  Created wheel for triton: filename=triton-3.0.0-0.editable-cp310-cp310-linux_x86_64.whl size=3072 sha256=bf4fd604e7ba68134d
2d935f96a9e96d0b62865cbf74081b39653a47a1708867
  Stored in directory: /tmp/pip-ephem-wheel-cache-5z1lh_f8/wheels/98/85/b6/6ad462c1a1735d0b1af8dba5e03cd79040eac6c513748a5631
Successfully built triton
Installing collected packages: triton
  Attempting uninstall: triton
    Found existing installation: triton 3.0.0
    Uninstalling triton-3.0.0:
      Created temporary directory: /tmp/pip-uninstall-yqqxblnw
      Removing file or directory /home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/__editable__.triton-3.0.0.pth
      Removing file or directory /home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/__editable___triton_3_0_0_fin
der.py
      Removing file or directory /home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/__pycache__/__editable___trit
on_3_0_0_finder.cpython-310.pyc
      Created temporary directory: /home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/~riton-3.0.0.dist-info
      Removing file or directory /home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/triton-3.0.0.dist-info/
      Successfully uninstalled triton-3.0.0

Successfully installed triton-3.0.0
Remote version of pip: 24.0
Local version of pip:  23.3.1
Was pip installed by pip? False
Removed build tracker: '/tmp/pip-build-tracker-ts1gk_qf'
(triton) tonym@prestige:/scratch/intel-xpu-backend-for-triton$
```

インテル® Core™ Ultra プロセッサー上での Triton のテスト

最初のテストは、`intel-xpu-backend-for-triton` コードベース内の `python/tutorials` ディレクトリーをチェックアウトすることです。基本的な機能を評価するため、このディレクトリーから最初のいくつかの例を実行しています。テストコードを見ると、使用するインテル® GPU を選択するため OpenAI* Triton リポジトリーからいくつかの変更が加えられていることが分かります。次のコードのデバイス指定を、

```
x = torch.rand(size, device='cuda')
y = torch.rand(size, device='cuda')
```

次のように変更します。

```
x = torch.rand(size, device='xpu')
y = torch.rand(size, device='xpu')
```

ランタイム環境の設定

新しい Triton ビルドを試すには環境に PyTorch* を追加する必要があるため、最初のステップとして PyTorch* 向けインテル® エクステンションをインストールします。

```
python -m pip install torch==2.1.0a0 torchvision==0.16.0a0 torchaudio==2.1.0a0 intel-extension-for-pytorch==2.1.10+xpu --extra-index-url https://pytorch-extension.intel.com/release-whl/stable/xpu/us/
```

チュートリアルの例では、いくつかの追加の Python* ライブラリーも必要です。次のように `pip` を使用して簡単にインストールできます。

```
pip install matplotlib pandas
```

実行例

最初のチュートリアルは、2 つのデータブロック間の単純なベクトル加算です。出力は以下のようになります。

```
(triton) tonym@prestige:/scratch/intel-xpu-backend-for-triton/python/tutorials$ python 01-vector-add.py
/home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/torchvision/io/image.py:13: UserWarning: Failed to loa
d image Python extension: ''If you don't plan on using image functionality from `torchvision.io`, you can ignore this
 warning. Otherwise, there might be something wrong with your environment. Did you have `libjpeg` or `libpng` install
ed before building `torchvision` from source?
  warn(
No CUDA runtime is found, using CUDA_HOME='/usr/local/cuda'
tensor([1.3713, 1.3076, 0.4940,  ..., 0.8654, 1.1553, 0.4071], device='xpu:0')
tensor([1.3713, 1.3076, 0.4940,  ..., 0.8654, 1.1553, 0.4071], device='xpu:0')
The maximum difference between torch and triton is 0.0
vector-add-performance:
           size     Triton      Torch
0        4096.0   0.003057   0.003096
1        8192.0   0.006198   0.006193
2       16384.0   0.012383   0.012375
3       32768.0   0.024736   0.024673
4       65536.0   0.049506   0.049383
5      131072.0   0.098313   0.098653
6      262144.0   0.196897   0.196460
7      524288.0   0.388927   0.389345
8     1048576.0   0.769500   0.768111
9     2097152.0   1.496443   1.492719
10    4194304.0   2.858620   2.823484
11    8388608.0   5.277260   5.095062
12   16777216.0   9.116155   8.511318
13   33554432.0  14.523118  12.744209
14   67108864.0  15.574265  12.825385
15  134217728.0  14.566962  11.500764
```

出力は、データの正当性とベクトル加算における操作の相対パフォーマンスを測定したものです。ベクトルのサイズの増加とともに、Triton アプローチによりインテル® Core™ Ultra プロセッサーの統合 GPU のパフォーマンスが向上していることが分かります。

2 つ目のチュートリアルは、fused-softmax の実装です。softmax 関数は、ベクトルを結果の確率分布にマッピングするもので、ニューラル・ネットワークの最後の活性化関数として DL アルゴリズムでよく使用されます。softmax は入力ベクトルに対して大量の操作を実行するため、ベクトル化によりパフォーマンスが大幅に向上する可能性があります。fused-softmax 実装は、データアクセスをブロックして naïve 実装を改善することを目的としています。PyTorch* にはネイティブの fused-softmax 実装がすでに存在することに注意してください。つまり、この操作のハードウェアに最適化された C++ バージョンと比較することができます。チュートリアルを実行すると、次の出力が得られます。

```
(triton) tonym@prestige:/scratch/intel-xpu-backend-for-triton/python/tutorials$ python 02-fused-softmax.py
/home/tonym/anaconda3/envs/triton/lib/python3.10/site-packages/torchvision/io/image.py:13: UserWarning: Failed to loa
d image Python extension: ''If you don't plan on using image functionality from `torchvision.io`, you can ignore this
 warning. Otherwise, there might be something wrong with your environment. Did you have `libjpeg` or `libpng` install
ed before building `torchvision` from source?
  warn(
No CUDA runtime is found, using CUDA_HOME='/usr/local/cuda'
/scratch/intel-xpu-backend-for-triton/python/tutorials/02-fused-softmax.py:184: UserWarning: The grad mode is detecte
d as torch.no_grad() is NOT enabled. In this mode on XPU, please expect NO graph and fusion optimization will be appl
ied.
 (Triggered internally at /build/intel-pytorch-extension/csrc/gpu/jit/fusion_pass.cpp:829.)
  ms, min_ms, max_ms = triton.testing.do_bench(lambda: naive_softmax(x), quantiles=quantiles)
softmax-performance:
          N    Triton  Torch (native)  Torch (jit)
0     256.0  0.508614        0.520325     0.485650
1     384.0  0.738521        0.779249     0.711792
2     512.0  0.976950        1.029912     0.916427
3     640.0  1.210549        1.273200     1.120450
4     768.0  1.441097        1.514099     1.302048
..      ...       ...             ...          ...
93  12160.0  8.014298       13.657079     3.345754
94  12288.0  8.042449       13.338677     3.417873
95  12416.0  8.148614       13.899843     3.363594
96  12544.0  8.153968       13.621084     3.355575
97  12672.0  8.267947       15.834839     4.628440

[98 rows x 4 columns]
```

ここで、softmax の 3 つの異なる実装のパフォーマンスを確認できます。最初の列は Triton バージョン、2 つ目の列は C++ 最適化バージョン、3 つ目の列は naïve バージョンです。インテル® Core™ Ultra プロセッサーでは、Triton バージョンは naïve バージョンよりも高速ですが、C++ ネイティブバージョンのコードよりも遅くなっています。Torch C++ 実装はすでに大幅に最適化されているため、この結果は当然のことです。しかし、Python* のみを使用した naïve 実装よりもはるかに高速なコードを作成できることは、開発者として興味深いことです。

まとめ

Triton フレームワークは、DL カーネル開発者から強く支持されています。単純な並列構文と、基盤となるハードウェア上のデータアクセスをすべて高水準の Python* コードから最適化する機能は、DL カーネル開発の簡素化に大いに役立ちます。このフレームワークをインテル® Core™ Ultra プロセッサー・ベースの AI PC と組み合わせることにより、必要なときに必要な場所で DL カーネル開発を行うための前途有望な道を切り拓くことができるでしょう。`intel-xpu-backend-for-triton` の開発は現在進行中です。このエキサイティングなハードウェアとソフトウェアの組み合わせの恩恵を受けられることを楽しみにしています。