# NVIDIA GPU を用いた Kaltura のトランスコード高速化の検討

宇田川　暢

新潟大学 学術情報基盤機構 情報基盤センター

udagawa@cais.niigata-u.ac.jp

# A Study of Accelerating Video Transcode at Kaltura with NVIDIA GPU

UDAGAWA Mitsuru

Center for Academic Information Service, Niigata Univ.

**概要**

オープンソースの動画配信プラットフォームである Kaltura Community Edition について、GPU を使うことでビデオアップロード時のトランスコード処理の高速化を検討する。

## 1 はじめに

Kaltura は同名の企業により提供されている動画配信プラットフォームであり、有償で提供されている SaaS やオンプレミスのほか、機能制限された Kaltura Community Edition（Kaltura CE）がオープンソースとして提供されている[1]。

Kaltura CE は他のオンプレミスの動画配信プラットフォームと同様に、大学の LMS で利用されているが、大学 ICT 推進協議会 2020 年度年次大会の企画セッション[1]において「Kaltura にアップロードされた動画の処理に時間がかかる」という意見があった。

## 2 Kaltura について

### 2.1 動画のアップロード

Kaltura CE に動画をアップロードした場合、事前に決められたプロファイルに従ってエンコードされる。プロファイルはそれぞれ異なる動画の画面サイズやビットレート、動画フォーマットの組み合わせにより定義された複数のフレーバーで構成されている。複数のフレーバーが用意されている理由は、再生するデバイスに最適化された動画を用意しておくためで、このようなフレーバーに合わせるためのエンコード処理はトランスコードと呼ばれる。

### 2.2 トランスコード処理

Kaltura CE では基本的に同時に複数のトランスコードが実行されないようになっているため、同時または短時間に複数の動画がアップロードされると、順にキューに溜まり、トランスコードされた動画が利用できるまでの時間が長くなる。また、動画配信サービス側に影響を及ぼさないためか、後述するエンコードパラメータでは 4 つ以上の CPU コアを利用しない設定となっていた。このため、多数のコアを持つ CPU を用いても、トランスコード速度の向上は見込みづらい。

一つ以上のフレーバーでのエンコードが完了するまでは、アップロードした動画を利用することができなくなっており、このような仕様から、前述の動画の処理時間に関する発言がなされたと思われる。

### 2.3 FFmpeg

Kaltura CE ではトランスコード処理は FFmpeg[2] を用いて行われており、本稿執筆時の安定板である Kaltura CE 16.14 では FFmpeg 4.0.2 が利用されている。この FFmpeg は複数の動画・音声コーデックに対応できるようビルドされている。

## 3 トランスコードの高速化

### 3.1 高速化のアプローチ

Kaltura CE で動画をアップロードしてからトランスコード完了までを高速化する場合、アップロードそのものにかかる時間を除けば、その処理時間のほとんどがトランスコード処理に占められている。よって、トランスコードを高速化する方法を検討する。

[1] http://www.kaltura.org/

[2] https://ffmpeg.org/

### 3.2 CPU によるアプローチ

トランスコードを高速化するため、FFmpeg でのエンコードパラメータを変更して CPU のみで高速化できないか検討する。Kaltura CE のデフォルトで設定されているプロファイルのフレーバーのうち、最もトランスコードに時間を要する「HD/1080 – WEB (H264/4000)」を検証時の対象とする。

まず、Kaltura CE でのトランスコード処理時のパラメータを確認するため、出力されるログおよび動画のメタデータを確認した。それらから得られた情報を元に、Kaltura CE のトランスコードと同等のコマンドを記述すると図 1 のようになる。ここで、行末に¥マークが無い行は、実際には次の行に続いているものとする。

図 1　トランスコードコマンド

```
$ ffmpeg -i input.mp4 ¥
  -c:v libx264 ¥
  -subq 7 -qcomp 0.6 -qmin 10 -qmax 50 ¥
  -qdiff 4 -bf 16 -coder 1 -refs 6 ¥
  -x264opts b-pyramid:weightb:mixed-refs:8x8dct:no-
fast-pskip=0:stitchable ¥
  -vprofile high -force_key_frames expr:
'gte(t,n_forced*2)' -pix_fmt yuv420p ¥
  -b:v 4000k -s 1920x1080 -r 24 -g 48 ¥
  -aspect 1920:1080 -c:a aac ¥
  -filter_complex 'aresample=async=1:min_hard_comp
=0.100000:first_pts=0' -strict -2 ¥
  -b:a 128k -ar 44100 -ac 2 -map_chapters -1 ¥
  -map_metadata -1 -f mp4 -flags +loop+mv4 ¥
  -cmp 256 -partitions +parti4x4+partp8x8+partb8x8 ¥
  -trellis 1 -refs 6 -me_range 16 -keyint_min 20 ¥
  -sc_threshold 40 -i_qfactor 0.71 -bt 1200k ¥
  -maxrate 4000k -bufsize 8000k ¥
  -rc_eq 'blurCplx^(1-qComp)' -vsync 1 -crf 23 ¥
  -threads 4 ¥
  -y output.mp4
```

明らかに画質を落とす設定を避ける場合、エンコードスレッド数を決定する ”-threads 4” の部分に着目することになる。ハイパースレッディングを含め、多数の CPU コアを持つ CPU の場合、この値を変更することで、トランスコードの時間を短縮することが可能となると考えられる。

### 3.2 GPU を使ったアプローチ

主にゲームグラフィック用に利用されている GPU を、小さなプロセッサが多数集まったものとして利用し、並列性の高い処理を効率的に行う GPGPU と呼ばれる手法が存在する。近年の GPU の中には、動画のエンコードに特化した機能を持つ GPU も存在しており、今回は NVIDIA 社製の GeForce RTX 3000 シリーズの GPU が持つ NVENC および NVDEC[3]（以下、NVENC / NVDEC）を利用して高速化を検討する。

図 1 のコマンドを NVENC / NVDEC を利用するために変更したものが図 2 となる。また、動画の B フレーム利用を指定する”-bf 16”オプションが NVENC に対応していないため削除している。

図 2　GPU 用トランスコードコマンド

```
$ ffmpeg -hwaccel cuda ¥
  -i input.mp4 ¥
  -c:v h264_nvenc ¥
  -subq 7 -qcomp 0.6 -qmin 10 -qmax 50 ¥
  -qdiff 4 -coder 1 -refs 6 ¥
  -x264opts b-pyramid:weightb:mixed-refs:8x8dct:no-
fast-pskip=0:stitchable ¥
  -vprofile high -force_key_frames expr:
'gte(t,n_forced*2)' -pix_fmt yuv420p ¥
  -b:v 4000k -s 1920x1080 -r 24 -g 48 ¥
  -aspect 1920:1080 -c:a aac ¥
  -filter_complex 'aresample=async=1:min_hard_comp
=0.100000:first_pts=0' -strict -2 ¥
  -b:a 128k -ar 44100 -ac 2 -map_chapters -1 ¥
  -map_metadata -1 -f mp4 -flags +loop+mv4 ¥
  -cmp 256 -partitions +parti4x4+partp8x8+partb8x8 ¥
  -trellis 1 -refs 6 -me_range 16 -keyint_min 20 ¥
  -sc_threshold 40 -i_qfactor 0.71 -bt 1200k ¥
  -maxrate 4000k -bufsize 8000k ¥
  -rc_eq 'blurCplx^(1-qComp)' -vsync 1 -crf 23 ¥
  -threads 4 ¥
  -y output.mp4
```

### 3.3 ベンチマークによる比較

Blender[4]により公開[5]されているアニメ動画である ”big_buck_bunny_1080p_h264.mov” を検証用動画として利用した。条件を変えながらこの動画をトランスコードすることで速度比較を行った。

今回、それぞれ異なる CPU を備えた 2 台のコンピュータを使用しており、Kaltura CE をインストールして動作確認に使用したコンピュータを

[3] https://developer.nvidia.com/nvidia-video-codec-sdk

[4] https://www.blender.org/

CPU 1、NVENC / NVDEC の検証に使用したコンピュータを CPU2 としている。CPU 2 では FFmpeg 4.0.2 を利用しているものの、libx264 などは Kaltura CE に付属するものと同一バージョンとはなっていない。

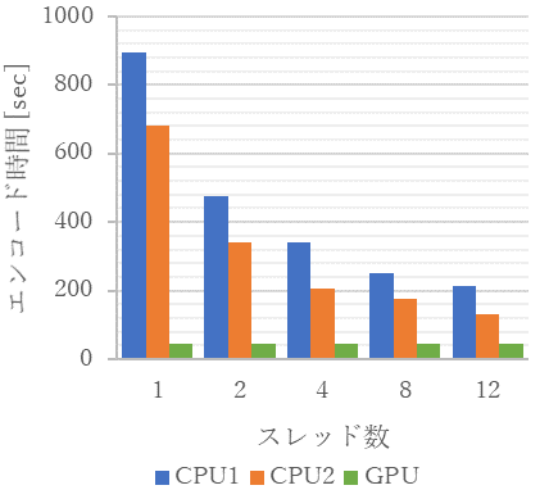
それぞれの環境に合わせ、図 1 または図 2 のコマンドを用いてベンチマークを行った結果、表 1 のようになった。表中の数字はトランスコード処理時の fps（frames per second）となっており、fps が大きいほどトランスコード速度が速いことを意味している。CPU1 は Intel Xeon E-2236、CPU2 は AMD Ryzen 5 5600X で、ともに 12 スレッドを同時に処理することが可能となっている。GPU は CPU2 のコンピュータに搭載された GeForce RTX 3600Ti を利用している。

表 1　スレッド数に対するエンコード速度(fps)

| スレッド数 | CPU1 | CPU2 | GPU |
|---|---|---|---|
| **1** | 16 | 21 | 329 |
| **2** | 30 | 42 | 328 |
| **4** | 42 | 69 | 328 |
| **8** | 57 | 82 | 330 |
| **12** | 67 | 109 | 330 |

また、動画の総フレーム数をフレームレートで割り、表 1 の内容をトランスコードに要した時間としてグラフ化したもの図 3 となる。

図 3　スレッド数に対するエンコード時間




[5] https://peach.blender.org/download/

## 4 おわりに

今回の検証の結果、GPU を利用することで大幅に Kaltura CE でのトランスコードを高速化することが可能である見込みあることがわかった。また、CPU のみの環境であっても多数の CPU コアを持つコンピュータの場合、Kaltura CE の設定を変更してある程度高速化することも可能であることが分かった。

ただし、残念ながら本稿執筆時点で Kaltura CE が動作する GPU 搭載コンピュータの環境を用意することができず、実際にどのような変更を行うことで Kaltura CE でのトランスコード高速化を実現することが可能か確認することができなかった。

## 参考文献

[1] 大西 淑雅、喜多 敏博、宮崎 誠、畠山 久、山口 真之介、梶田 将司、オンライン講義の継続に向けた OSS の利用、大学 ICT 推進協議会 2020 年度年次大会予稿集、2020、p.536-538