B-003

Tesla V100 を用いた Tensor コアと CUDA コアの併用による行列積の高速化

Accelerating GeMM with Tensor and CUDA Cores Using Tesla V100

熊谷 和紀† Kazuki Kumagai
富永 浩文‡ Hirobumi Tominaga
中村 あすか‡ Asuka Nakamura
前川 仁孝‡ Yoshitaka Maekawa

1. はじめに

科学技術計算の分野では GPU による高速化が行われており[1]，特に Tensor コアと呼ばれる行列積和演算専用コアを搭載した NVIDIA 社 Volta 世代の GPU が，幅広い分野の計算に活用されている[2]．多くの研究では，各コアに最適化するようにチューニングをすることが一般的である[3]．一方，Tensor コアを利用するカーネルと CUDA コアを利用するカーネルを同時に起動し，各コアを同時に動作させることで，単位時間あたりの演算数の増加につながり，さらなる高速化が期待できる．そこで本研究では，行列積を対象として CUDA コアと Tensor コアを併用し，計算を各コアで分割実行する手法を実装し，その有効性を評価する．

2. Volta アーキテクチャ

図 1 に Tesla V100 の構成を示す．Tesla V100 は 80 個の SM を搭載し，各 SM は CUDA コアを 64 個，Tensor コアを 8 個搭載する．シェアードメモリは SM 当たり最大 96kB である．Tensor コアは行列積和専用のコアであり，CUDA コアと比べ行列演算性能が高い．CUDA コアは行列積和演算以外の処理も実行できるが，Tensor コアでは行列積和演算以外の処理は実行できない．また，各コアは，CUDA コアを使う命令と Tensor コアを使う命令を別々に発行し，独立して動作させることができる．

3. Tensor コアを用いた行列演算

Tensor コアを利用するために，CUDA では WMMA(Warp Matrix Multiply Accumulate)API を使用する．図 2に Tensor コアを用いる WMMA API のコードを示す．図 2は，Tensor コアを使い行列積を計算するプログラムであり，専用の API を利用して行列積を完了する．図 2中の M，N，K は，Tensor コア利用時に 1Warp で計算する行列の行と列のサイズを表す．

4. CUDA コアと Tensor コアでの分割手法

図 3(a)のように，単一 Stream では両方のコアは同時に動作しない．各コアを並列動作するための手法として，CUDA カーネル内に WMMA 命令を記述し，単一カーネルを発行するスレッドブロックレベルの併用手法と，各コアのカーネルを分けて複数カーネルを同時発行するグリッドレベルの併用手法がある．単一カーネルを用いる手法は，

† 千葉工業大学情報科学研究科情報科学専攻
Graduate School of Information and Computer Science, Chiba Institute of Technology
‡ 千葉工業大学情報科学部情報工学科
Department of Computer Science, Chiba Institute of Technology

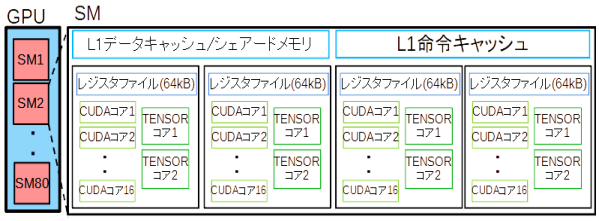


図 1　Tesla V100 Streaming Multiprocessor(SM)

```
__global__ void WMMAF16TensorCore(half *A, half *B, float *C, float *D){
    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a_frag;
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b_frag;
    wmma::fragment<wmma::accumulator, M, N, K, float> c_frag,d_flag;

    wmma::load_matrix_sync(a_frag, A , M);
    wmma::load_matrix_sync(b_frag, B , K);
    wmma::load_matrix_sync(c_frag, C , N);

    wmma::mma_sync(d_frag, a_frag, b_frag, c_frag);

    wmma::store_matrix_sync(D , c_frag, N, wmma::mem_row_major);
}
```

図 2　WMMA API コード

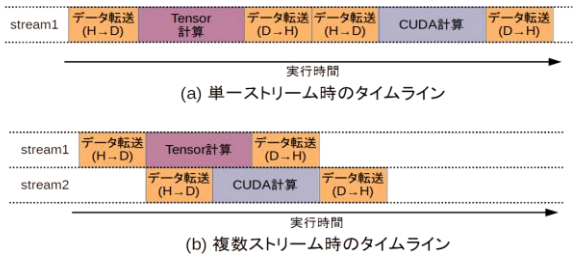


図 3　ストリーム数ごとのタイムライン

スレッドブロックを CUDA コアと Tensor コアにそれぞれ割り当てる．SM に割り当てられたスレッドブロックは，非同期に実行するため，ブロックレベルで各コアを並列に動作する．複数カーネルを用いる手法は，Stream の機能を利用し CUDA カーネルと Tensor カーネルを同時に発行する．図 3(b)に，Stream 機能を用いてカーネルを同時発行する例を示す．図のように，グリッドレベルで各コアが並列に動作し，計算とデータ転送がオーバーラップする．本研究では CUDA コアと Tensor コアを併用するために，複数カーネルを同時実行する手法を実装する．

CUDA コアと Tensor コアでは計算性能に差があるため，均等な計算量を割り当てると，片方を待つ時間が発生し，全体実行時間が増加する．このため，割り当てる行列サイズを調整することで，各コアの実行時間の均一化と待ち時間の削減を図り，全体実行時間を短縮できる．

5. 評価

CUDA コアと Tensor コアを併用する手法の有効性を評価するために，正方行列の行列積を対象として，実行時間を測定する．評価環境は CPU が Intel Xeon E5-2690v4，GPU が Tesla V100，CUDA バージョンは 11.2 である．本測定の計算カーネルには，setMathMode で CUDA コアと Tensor コアを指定した cuBLAS の Sgemm を用い，各カーネルで図4に示す領域を計算する．測定条件は，行列サイズ 3 パターン，分割割合 5 パターンとする．

図5に各行列サイズごとの実行時間を示す．図中の実行時間は CPU-GPU 間のデータ転送も含めたものであり，CUDA コアで実行する割合が 0%および 100%のときは，単一のコアの実行となり，既存の実行手法と同じである．図5より，行列サイズが 2048^2 および 16384^2 のときに，提案手法が既存手法より高速であり，CUDA コアと Tensor コアを併用することの有効性が確認できる．一方，行列サイズ 256^2 においては，提案手法を用いることで実行時間が増加することが分かった．行列サイズ 256^2 は，CUDA コアで実行する割合が 100%のときよりも 0%のときの方が処理時間が長い問題である．つまり，256^2 は，行列サイズが小さいため，実際の演算時間よりも図2に示した Tensor カーネルの起動コストの方が大きい問題であることが分かる．Tensor カーネルの起動コストによるオーバヘッドは，2048^2 および 16384^2 でも Tensor カーネルに割り当てる処理の割合が小さいと提案手法の効果が小さいことからも確認できる．上記より，提案手法を実装する際には，Tensor コアへ割り当てる処理の割合は，CUDA コアへの割合よりも大きくすることが重要であると考えられる．

また，提案手法を用いることで，実際に Tensor カーネルと CUDA カーネルによる並列動作が行われたか確認するために，NVVP(Nvidia Visual profiler)を用いて，各コアのカーネルのタイムラインを解析する．図6に，図5中で最も高い高速化率が得られたパターンである行列サイズ 16384^2，CUDA コアの実行割合 25%におけるタイムラインを示す．図中の Stream1 は CUDA コア側であり，Stream2 は Tensor コア側である．図 6 より，若干ではあるが，両カーネルが同時に動作しているが，大部分でカーネルの起動がオーバーラップせずに実行されたことが分かる．この理由は，本測定で利用したライブラリでは，カーネルごとに GPU のリソースを全て活用するようなチューニングが行われており，両カーネルを同時に起動したことでリソース不足になったためと考えられる．一方で，図 3 (c)に示したデータ転送の隠蔽が行われていることを確認でき，提案手法において，データ転送の隠蔽は有効であると考えられる．また，CUDA カーネルと Tensor カーネルの実行時間が同程度になっていることから，今回測定を行った実行割合の中では，CUDA の実行割合 25%が，待ち時間を削減する上で最も良い分割割合であると考えられる．CUDA コアは行列積以外の処理も実行可能であるため，本手法は Tensor コアで行列積演算を，CUDA コアで別の計算を実行する処理に対しても適用できると考えられる．

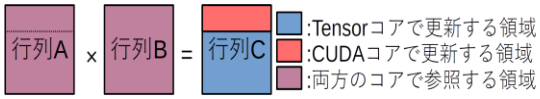
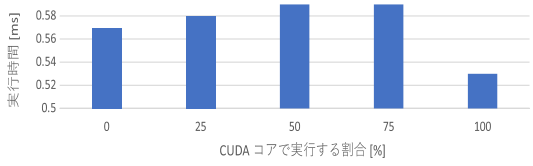
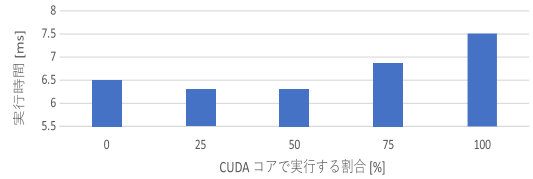


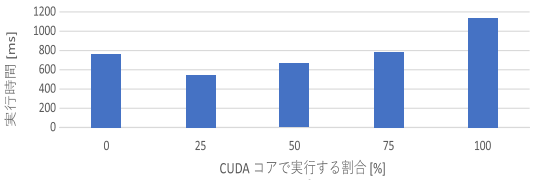
図 4　各コアへの分割領域



(a) 行列サイズ 256^2 の実行時間



(b) 行列サイズ 2048^2 の実行時間



(c) 行列サイズ 16384^2 の実行時間

図 5　各行列サイズの実行時間

図 6　NVVP タイムライン

6. おわりに

本稿では，CUDA コアと Tensor コアを併用し，計算を各コアで分割実行する手法を提案し，その有効性を評価した．行列積を対象とした評価の結果，提案手法は Tensor コアのみの実行に比べ，最大約 1.4 倍高速化することを確認した．今後の展望として，提案手法を各コアで異なる計算を実行する問題に対して評価することが挙げられる．

参考文献

[1] 大島聡史, 吉瀬謙二, 片桐孝洋, 弓場敏嗣:CPU と GPU を用いた並列 GEMM 演算の提案と実装, 情報処理学会論文誌コンピューティングシステム(ACS),Vol.47,No.SIG12(ACS15),pp. 317-328(2006).

[2] A. Sorna, X. Cheng, E. D'Azevedo, K. Won and S. Tomov, "Optimizing the Fast Fourier Transform Using Mixed Precision on Tensor Core Hardware," 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), 2018, pp. 3-7, doi: 10.1109/HiPCW.2018.8634417.

[3] Markidis,S.Chien,D,W,S.Laure,E.:NVIDIA Tensor Core Programmability,Performance&Precision,in IEEE IPDPSW(2018).