5L-07

GPUを用いた行列-行列積の実装と性能評価

榊田　匠†　　田中　輝雄†　　藤井　昭宏†
工学院大学†

1　はじめに

高い理論演算性能を持つGPU (Graphics Processing Unit) で性能を最大限に発揮させるプログラムを作成するには，多数のスレッドによる超並列やメモリへのコアレスアクセスなどGPUプログラミングの技法を駆使する必要がある[1].

本研究では，行列-行列積の実装と性能評価を行った．演算効率を極限まで高めた数値計算ライブラリcuBLAS[2]に対し，CUDAを用いた実装で高い性能を得ることを目標とした．性能を引き出すため，効率的なShared Memoryの利用しかたに着目した．Shared Memory使用量を変更することで，問題サイズごとにSM (Streaming Multiprocessor)の稼働率を最大限に上げることを試みた．

2　GPUハードウェア

本研究では，産業技術総合研究所のABCI[3]に搭載のNVIDIA社製GPU Tesla V100[4]を用いた．GPUの概要図を図1，Tesla V100の仕様を表1に示す．GPUは複数のSMを搭載しSMには演算器(CUDA Core)やShared Memoryなどが搭載されている．Shared MemoryとGlobal Memoryは階層構造であり，Shared Memoryのほうがより高速である．

Tesla V100はSM数80，SMの倍精度演算器(FP64 Core)数32である．倍精度演算の理論演算性能は約7.8TFLOPSである．

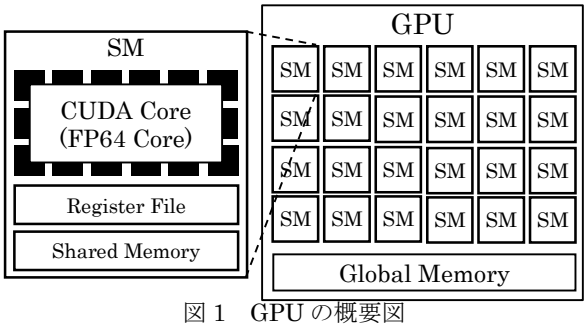

図1　GPUの概要図

表1　Tesla V100の仕様

SMs	80
FP64 Cores / SM	32
GPU Boost Clock	1530 MHz
Peak FP64 TFLOPS	7.8
Memory Size	16GB
Shared Memory Size / SM	96 KB
Register File Size / SM	256 KB

Implementation and Performance Evaluation of Matrix-Matrix Multiplication using GPU
Takumi Sakakida†, Teruo Tanaka† and Akihiro Fujii†
Kogakuin University†

表2　CUDAのThread構造とGPU構造

Thread構造	GPU構造
Thread	CUDA Core
Thread Block (Threadの集合)	SM (CUDA Coreの集合)
Grid (Thread Blockの集合)	GPU (SMの集合)

3　CUDAのマルチスレッディング

プログラム作成にはNVIDIA社のCUDAを使用した．表2にCUDAプログラミングのThread構造とGPUのハードウェア構造を示す．両者の構造が対になっていることがわかる．そのため，GPU構造を考慮しThreadを生成すれば，より高い性能を得られる．そこで本研究ではSMとThread Blockの関係について詳細に評価を行った．

4　実効Thread Block数

各SMへの処理の振り分けは，Thread Block単位で行われる．SMの稼働上限はThread数(2048)，Register使用量(256KB)，Shared Memory使用量(96KB)の3要素であり，これに収まる数だけThread Blockが同時に稼働できる．3要素すべてを半分使用するThread Blockを多数生成した場合には，各SMは同時に2Thread Blockが稼働可能となる．このように実際に同時稼働可能なThread Block数を実効Thread Block数と呼ぶ[5].

5　行列−行列積の実装

行列-行列積の実装にはNathらの実装[6]と数値計算ライブラリcuBLASのThread構成などを参考にした．図2に実装の概要図を示す．行列-行列積をC=A×Bとしたとき，Cを128行64列の部分行列に分割し，各Thread Blockがそれを計算する．Thread BlockはAをs列，Bをs行ずつShared Memoryに格納する．そして128行64列の部分行列をさらに8行8列の部分行列に分割し，各Threadが外積形式で計算する．

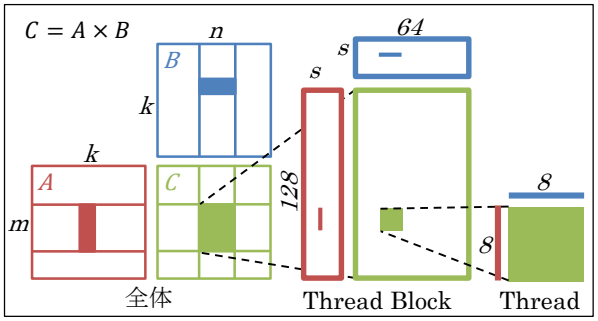

図2 行列-行列積の実装

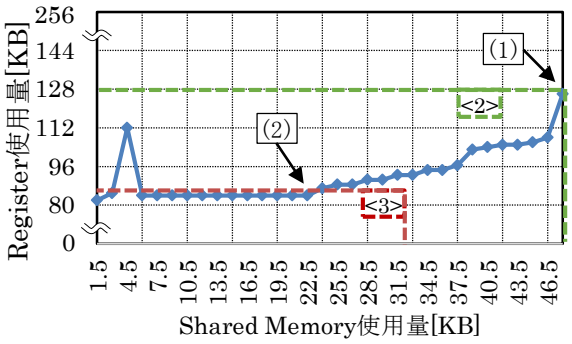

図 3　Shared Memory と Register の使用量

表 3　計測時の Thread Block 設定

	設定値		プロファイラから得た結果	
	Thread	Shared Memory	Register	実効 Thread Block 数
(1)	128	48KB	126KB	2
(2)	128	22.5KB	84KB	3

6　予備実験: Register 使用量の取得

実効 Thread Block 数を決める要素のうち，Register 使用量についてはプロファイラで取得した．図 3 に Shared Memory と Register の使用量を示す．また，実効 Thread Block 数の境界線を図 3 内に破線で示す．図 3 の領域<2>は Shared Memory と Register の使用量が 4 節に示した上限の半分以下のため実効 Thread Block 数は 2 である．同様に領域<3>は実効 Thread Block 数が 3 であり，本実装では実効 Thread Block 数を 2 と 3 に設定可能である．

7　評価実験

Thread Block 内の計算量は一定のまま Thread Block 数を 1 刻みで増加させて計測を行うため，行列サイズ M を 128，K を 4096 で固定し，N を 64 から 47360 まで 64 ずつ増加させた．

表 3 に今回計測を行った 2 種類の Thread Block 設定について示す．図 3 の(1)と(2)の設定とした理由は，各実効 Thread Block 数の領域内で，最も Shared Memory，Register を使用するためである．

計測結果を図 4 に示す．横軸は生成されている Thread Block 数を表し，数が大きいほど問題の行列サイズが大きくなる．縦軸は実測性能を表す．グラフ(1)の時，実効 Thread Block 数は 2 であり，80SM 全てが 2Thread Block 稼働可能な 160Thread Block の倍数を頂点とする鋸波が確認できる．同様にグラフ(2)では 240Thread Block の倍数を頂点とする鋸波で確認できる．この結果から，生成する Thread Block 数/SM 数の倍数で実効 Thread Block 数を指定すれば，高い性能を得られるといえる．今回のように限られた実効 Thread Block 数でしか稼働できない時も，それらのうち性能の良いほうを選択し実行することで性能低下を抑制できる．さらに，複数の実効 Thread Block 数での稼働を組み合わせることでも高い性能を得られる．例として生成される Thread Block 数が 560 の時，実効 Thread Block 数 2 で 320Thread Block を，実効 Thread Block 数 3 で 240Thread Block を分割して実行を行った．結果を図 4 の菱形に示す．分割せず単一の

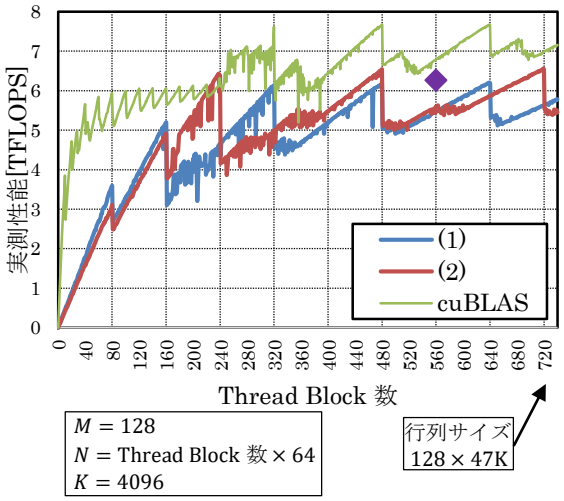

図 4　実効 Thread Block 数による性能の差

実効 Thread Block 数で実行するよりも性能が向上した．また，cuBLAS についても同じ行列サイズで計測を行った．240Thread Block 周辺でほぼ同程度の性能を得られ，9 割を超える性能を得られる Thread Block 数も確認できた．

8　おわりに

本研究では，GPU 上で行列-行列積を実装し，実効 Thread Block 数を変更することによる性能の違いを評価した．生成される Thread Block に対して適切な実効 Thread Block 数を指定することでプログラムの性能を向上させることができた．その結果，特定の問題サイズでは cuBLAS に対して同程度の性能を得られた．

今回は実効 Thread Block 数 2 と 3 の場合のみ計測することができた．その他の実効 Thread Block 数でも計測と評価を行うのが今後の課題として挙げられる．

謝辞　本研究の一部は JSPS 科研費 JP18K19782, JP18K11340, JP15K15998 の助成を受けたものです．

参考文献

[1] Hisa Ando，GPU を支える技術 技術評論社(2017)

[2] NVIDIA: cuBLAS, https://developer.nvidia.com/cublas（2019/1/11 アクセス）

[3] 産業技術総合研究所: ABCI , https://abci.ai/（2019/1/11 アクセス）

[4] NVIDIA TESLA V100 GPU ARCHITECTURE, http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf（2019/1/11 アクセス）

[5] Best Practices Guide, https://docs.nvidia.com/cuda/archive/9.2/cuda-c-best-practices-guide/ （2019/1/11 アクセス）

[6] R. Nath, S. Tomov, and J. Dongarra, “An Improved Magma Gemm For Fermi Graphics Processing Units.”, The International Journal of High Performance Computing Applications, 24(4), pp. 511–515 (2010)