ディレクティブベースのプログラミングモデルを用いた倍々精度演算の性能評価

山浦朴人† 福永晋司† 菱沼利彰‡ 藤井昭宏† 田中輝雄†

工学院大学† 株式会社科学計算総合研究所‡

1. はじめに

物理シミュレーションの核である Krylov 部分空間法は，連立一次方程式の近似解を求める反復解法である．Krylov 部分空間法では，丸め誤差の影響で発散や反復回数の増加が起きることがある．これらは高精度演算を用いることで改善できる可能性がある[1].

高精度演算の手法のひとつに倍々精度 (DD) 演算がある．DD 演算は，Bailey が提案した“Double-Double”のアルゴリズム[2]を用い，倍精度浮動小数点数を 2 つ組み合わせて 4 倍精度相当の演算を行う手法である．DD 演算の問題点は，倍精度演算と比べて演算量が 10〜20 倍であるため計算時間が増加することである．

Mukunoki らは CUDA を用いて BLAS に含まれる演算を DD で実装し，GPUによる高速化が有効であることを示している[3]．CUDA プログラムは CPU 向けのプログラムから大幅に変更する必要がある．一方，簡単な記述で GPU 向けのプログラムを開発する方法として OpenACC [4]や OpenMP Offloading（OMP Offloading）[5]のようなディレクティブベースの GPU プログラミングモデルがある．

本研究の目的は，DD 演算におけるディレクティブベースの GPU プログラミングモデルの性能を CUDA との比較により明らかにすることである．BLAS LEVEL1 の演算であるベクトル積和演算 axpy （$\boldsymbol{y} = \alpha\boldsymbol{x} + \boldsymbol{y}$）と BLAS LEVEL2 相当の演算である疎行列ベクトル積 SpMV （$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$）を用いて性能評価を行った．

2. ディレクティブベースの GPU プログラミングモデル

ディレクティブベースの GPU プログラミングモデルでは，CPU 向けのプログラムにディレクティブを挿入することでコンパイラが GPU で実行するプログラムを生成する．本研究では OpenACC と OMP Offloading という 2 つのプログラミングモデルを用いた． OMP Offloading は OpenMP (OMP)のコンパイラ拡張である．

OMP Offloading を用いた DD ベクトル積和演算 (DD-axpy) のプログラムの GPU 実行部分を図 1 に示す．図 1 の 3 行目は直下のブロックを GPU で実行するディレクティブであり，CUDA では必要となるスレッド数等の設定や GPU 実行部分の関数化は必須ではない．2 行目と 7 行目は CPU-GPU 間のデータコピーを行うディレクティブであり，CUDA では配列ごとに行っていたデータコピーを一度に行うことができる．図中の MUL，ADD はそれぞれ

```
1  DD *x, *y, alpha;
2  #pragma omp target enter data map (to : x[0:N],
       y[0:N]), alpha)
3  #pragma omp target teams distribute parallel for
       private(tmp)
4  for(int i = 0, i < N, i++){
5     MUL(&tmp, alpha, x[i]);
6     ADD(&(y[i]), tmp, y[i]);  }
7  #pragma omp target exit data map(from : y[0:N])
```

図 1 OMP Offloading を用いた DD-axpy のコード

“Double-Double”のアルゴリズム[2]に基づいた，DD の乗算と加算を行う関数である．OpenACC においてもディレクティブが異なるだけでほぼ同一のプログラムとなる．

3. 実験

3.1 実験概要

性能評価のための指標として，BLAS LEVEL1 の演算である DD-axpy，BLAS LEVEL2 相当の演算である倍精度疎行列と DD ベクトルの積（DD-SpMV)の性能を計測した．また，daxpy，倍精度疎行列ベクトル積(d-SpMV)も DD 演算の評価のための比較対象とした．

GPU プログラミングモデルとの比較のため，各演算について OMP を用いて並列化し CPU で実行した性能も計測した．DD のデータは，2 つの倍精度浮動小数点数をメンバとして持つ構造体の配列を作るデータレイアウト（Array of Structures）によって保持した．DD-SpMV において，最適化は GPU 実行時のスレッド（OpenACC ではベクトル）数とブロック （OMP Offloading ではチーム，OpenACC ではギャング）数の調整を行った．

実験には名古屋大学のスーパーコンピュータである不老 Type2 を 1 ノード使用した．GPU は NVIDIA Tesla V100 が搭載されており，GPU1 つ当たりの CUDA コア数は 5120，メモリバンド幅は 900 GB/s である[7]．各プログラミングモデルにおいて使用したコンパイラとオプションを表 1 に示す．

表 1 コンパイラおよびそのオプションの設定

Model	Compiler	Options
OpenACC	nvc 21.2	-O3 -std=c++11 -acc -tp=host -gpu=cc70
OMP Offloading	clang 13.0.0	-O3 -std=c++11 -fopenmp -fopenmp -targets=nvptx64 -Xopenmp-target -march=sm_70 -lm
CUDA	nvcc 11.2	-O3

Performance Evaluation of DD Operations using Directive-Based Programming Models
Naoto Yamaura†, Shinji Fukunaga†, Toshiaki Hishinuma‡, Akihiro Fujii†, Teruo Tanaka†
†Kogakuin University
‡Research Institute for Computational Science Co. Ltd.

いずれの演算，プログラミングモデルにおいても計算にかかった時間のみを計測し，CPU-GPU 間のデータコピーの時間は含めていない．ディレクティブベースの GPU プログラミングモデルを用いる際，元となる CPU 向けプログラムは C 言語で記述した．1 演算当たりの倍精度演算回数が異なる倍精度と DD を比較するために，演算に使用した 1 秒あたりのデータ量を性能の指標とした．本論文ではこれをメモリ性能と呼び，単位は［GB/s］で表す．

3.2 axpy

配列サイズを 5×10^4 から 6×10^6 まで 5×10^4 ずつ変化させて計測を行った．

daxpy と DD-axpy の各配列サイズにおけるメモリ性能それぞれを図 2，図 3 に示す．縦軸はメモリ性能を表し，数が大きいほど性能が高い．横軸は配列サイズを表す．

どちらの演算においても，配列サイズが 3×10^6 以上になるとディレクティブベースのプログラミングモデルと CUDA のメモリ性能に大きな差はみられなくなった．daxpy，DD-axpy ともにメモリ性能は 700～800 GB/s 程度で横ばいになり，変数型の違いによって差は出なかった．NVIDIA Tesla V100 のメモリバンド幅である 900 GB/s に対して，理論性能の 80 %程度のメモリ性能を引き出せた．

3.3 SpMV

上三角帯行列を用いて，サイズ 10^6 において帯幅を 10 から 100 まで 10 ずつ変化させて計測を行った．疎行列の格納形式は CRS を用いた．

スレッド (ベクトル)数とブロック (チーム，ギャング) 数を最適としたときの DD-SpMV の各帯幅におけるメモリ性能を図 4 に示す．縦軸はメモリ性能，横軸は帯幅を表す．DD-SpMV ではディレクティブベースのプログラミングモデル，CUDA ともに 600～700 GB/s のメモリ性能となり，理論性能の 70 %程度のメモリ性能を引き出せた．また，スレッド数ごとのメモリ性能を図 5 に示す．図 5 はサイズ 10^6，帯幅 100 の帯行列を用いた結果であり，図 5 の各プログラミングモデルにおける最も性能の高い点は図 4 の帯幅 100 の点に対応する．OMP Offloading と CUDA のスレッド数は 1024 が上限である．縦軸はメモリ性能，横軸は 1 ブロック （チーム，ギャング）あたりのスレッド (ベクトル) 数を表す．

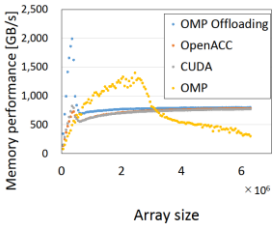

図 2　daxpy のメモリ性能

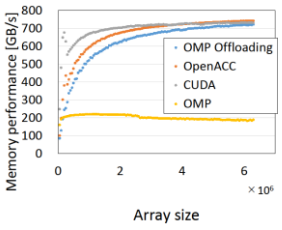

図 3　DD-axpy のメモリ性能

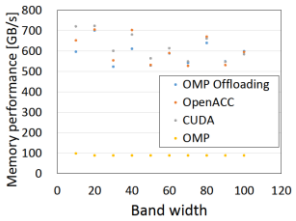

図 4　DD-SpMV のメモリ性能

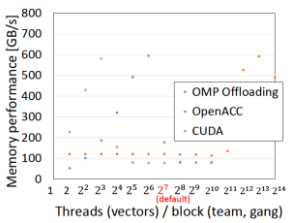

図 5　スレッド数ごとのメモリ性能

ディレクティブベースのプログラミングモデルでは CUDA におけるスレッド数やブロック数にあたる値の設定が必須ではなく，ユーザが設定せずに実行した場合は自動で設定される．DD-SpMV においてユーザが設定しなかった場合，OMP Offloading のスレッド数は 128，チーム数は（行数 / 128 + 1）であり，OpenACC のベクトル数は 128，ギャング数は（行数 / 128 + 1）となる．

スレッド数ごとのメモリ性能を見ると，OMP Offloading は 64 スレッドのときに最もメモリ性能が高く，128 スレッドのメモリ性能は 64 スレッドのメモリ性能の 30 %であった．OpenACC は 8192 ベクトルのときに最もメモリ性能が高く，128 ベクトルのメモリ性能は 8192 ベクトルのメモリ性能の 18 %となった．ディレクティブベースのプログラミングモデルにおいても，十分な性能を出すためには CUDA と同様にスレッド数などの値を適切に設定する必要がある．

4. おわりに

本研究ではディレクティブベースの GPU プログラミングモデルを用いて DD 演算を実装し，その性能を CUDA との比較により評価した．daxpy，DD-axpy では配列サイズが 3×10^6 以上になると CUDA と同程度の性能となり，理論性能の 80 %程度のメモリ性能が引き出せた．また，DD-SpMV ではサイズ 10^6 の各帯幅で CUDA と同程度の性能となり，理論性能の 70 %程度のメモリ性能が引き出せた．

ディレクティブベースの GPU プログラミングモデルを用いて DD-SpMV を実行した際，デフォルトのスレッド数では 100～200 GB/s のメモリ性能となった．しかし，スレッド数を適切に設定することで 600 GB/s 程度のメモリ性能となり，デフォルトのスレッド数で実行したときと比較して 3～5 倍のメモリ性能が引き出せた．スレッド数を適切に設定した場合のディレクティブベースのプログラミングモデルのメモリ性能は CUDA と同程度となった．

今後の課題として，自動でスレッド数やブロック数を適切に設定するプログラムを実装し利便性を高めることが挙げられる．

謝辞

本研究は JHPCN 拠点の支援（課題番号：jh210019-NAH）及び JSPS 科研費 JP18K11340 の助成により実施した．

参考文献

[1] Hasegawa, H.: Utilizing the Quadruple-Precision floating-Point Arithmetic Operation for the Krylov subspace Methods, The 8th *SIAM Conference on Applied Linear Algebra* (2003).
[2] Bailey, H., D.: High-Precision Floating-Point Arithmetic in Scientific Computation, *Computing in science & engineering*, Vol.7, No.3, pp.54-61 (2005).
[3] Mukunoki, D. and Takahashi, D.: Implementation and Evaluation of Quadruple Precision BLAS Function on GPUs, *International Workshop on Applied Parallel Computing*, pp.249-259 (2010).
[4] OpenACC-standard.org.: Homepage, 入手先〈https://www.openacc.org/〉 (参照 2021-12-25).
[5] Antao, F., S., et al.: Offloading support for openmp in clang and llvm, 2016 Third Workshop on the LLVM Compiler Infrastructure in HPC, pp.1-11 (2016).
[6] 名古屋大学：スーパーコンピュータ「不老」紹介，名古屋大学 情報連携推進本部，入手先〈https://icts.nagoya-u.ac.jp/ja/sc/overview.html〉(参照 2021-12-25).