TSUBAME 共同利用　平成28年度　学術利用　成果報告書

利用課題名　OpenACC の拡張によるアプリケーションの自動最適化
英文：An OpenACC Extension for Application Auto-tuning

利用課題責任者　星野哲也

所属　東京大学情報基盤センター

邦文抄録（300 字程度）
アクセラレータ向けの指示文ベースプログラミングモデルである OpenACC を用いて実装されたアプリケーションは，様々な環境において実行可能でありその高い可搬性が特徴と言える．しかし特定のデバイス向けに最適化されたアプリケーションは，別のデバイスにおける実行では十分な性能を達成できない場合がある．異なる性能特性を持つデバイスでは，有効な最適化方針が異なる場合があるためである．我々は，特にデバイスの性能特性により性能に大きく影響を与える得るプログラムのデータレイアウトに着目し，OpenACC の拡張により最適化を可能とした．実アプリケーションである UPACS と CCS-QCD へ適用し，手動による最適化と比較して，それぞれ 99.4%，92.3%の性能を達成した．
Keywords: 5つ程度
OpenACC, GPU

背景と目的

2016 年 6 月の Top500 List にて圧倒的な 1 位を獲得した Sunway TaihuLight 向けの並列プログラミングモデルにも採用されるなど，メニーコアアクセラレータ向けのプログラミングモデルとして OpenACC が脚光を浴びている．OpenACC は，マルチコア CPU 向けの並列プログラミングモデルとして一般的である OpenMP 同様，指示文ベースのプログラミングモデルであり，CPU 向けに作られた既存のアプリケーションに数行の指示文を挿入することにより，アクセラレータ上での実行を可能とする．代表的な演算アクセラレータである GPU 向けのプログラミングモデルとして，現在でも主流である CUDA が NVIDIA 社製の GPU のみを対象とする一方，OpenACC は様々なアクセラレータデバイスをターゲットとして選択することができ，その可搬性を売りとしている．しかし，ターゲットとするデバイスはそれぞれ異なる性能特性を持つことが一般的であり，デバイス毎に最適化戦略が異なることはよく知られた問題であるが，OpenACC のプログラミングモデルはその点への対応が十分とは言えない．機能的な可搬性という面から見れば，CUDA 同様広く使われている OpenCL も多数のデバイスをターゲットとしており優れているが，性能的な可搬性という面では，OpenACC と同様問題を抱えている．著者らはこれまでの研究において，CUDA・OpenACC を用いて実アプリケーションの移植・最適化を行った結果，構造体の配列を手動で展開し，配列の構造体に書き換えるという，データレイアウトの変更が最も効果的な最適化であることを確認したが，適切なデータレイアウトは実行デバイス毎に異なるため，他のデバイスで実行した際に最適な性能を得られるとは限らない．この問題は，多数の異なるデバイスを対象とする OpenACC のようなプログラミングモデルにおいては，性能の可搬性の低下原因となるため，CUDA と比較してより顕著な問題となる．

この問題を解決するため，著者らは以前より，OpenACC にデータレイアウトを抽象化する拡張指示文を追加することで，データレイアウトの自動最適化を目指している．以前の研究においては，データレイアウトを抽象化するための指示文の提案を行い，またソース-to-ソースのトランスレータを作成し，ベンチマークプログラムにてその有効性を確認した．しかし，実際のアプリケーションに適用するためには機能が不足しており，また性能最適化も十分ではなかった．本報告では，実アプリケーションに適用するための指示文拡張を行い，実アプリケーションである UPACS と CCS-QCD への適用を行うことで，トランスレータの評価を行った．その結果，手動による最適化には及ばないものの，指示文適用前のベースラインから UPACS と CCS-QCD でそれ

ぞれ 123.5%, 120.7%の性能向上を達成した詳細を報告する.

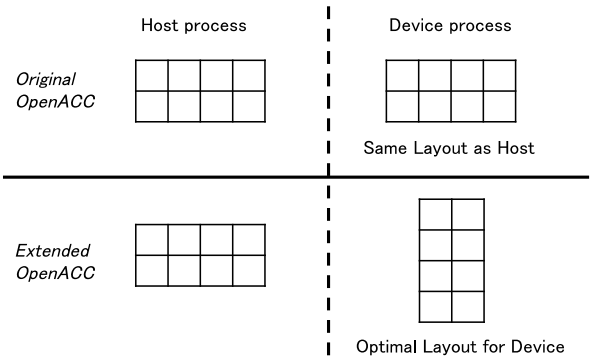

図 1

概要

OpenACC には対象とするデバイス毎に並列度を切り替えるための, device_type という指示子が導入されているが, データレイアウトを切り替える仕組みは導入されていない. 適した並列度, データの格納順序, データへのアクセス順序は密接に関係し, 性能に直結しているため, OpenACC の性能可搬性を高めるためには, 並列度を切り替える仕組みのみでは不十分であると考える. 実行デバイスごとに最適なデータレイアウトが異なるという問題は, 現在広く使われている CUDA や OpenCL といった低レベルなプログラミングモデルにおいても良く知られた問題であり, CPU 向けに書かれたプログラムをアクセラレータ向けに書き換える際の問題として良く知られている. しかし低レベルなプログラミングモデルにおいては, プログラムそのものをアクセラレータ専用に書き換えてしまうことが多いため, 比較的大きな問題にはなっていなかった. その一方で OpenACC では, 指示文を無視すれば元のプログラムとしても実行可能であり, プログラムを維持したまま移植出来る機能的な可搬性がメリットであるため, 得意とするデータレイアウトの違いによる性能可搬性の低下は解決すべき重大な問題である. 抽象化を行うことで, 低レベルなプログラミングモデルでは難しいデータレイアウトの自動最適化を可能とすることが, 本研究で達成されるべき目標である.

そこで本稿においては, OpenACC へのデータレイアウト最適化指示文の導入を提案する. OpenACC は host と device という独立したメモリ空間を持ち, データ移動指示文を用いることで host-device 間で適宜データのコピーを行い, データの一貫性を保つ. OpenACC ではこの際, device 側に持つデータは host 側と同じデータレイアウトしか許していないが, 図 1 に示すように, Device process 側のレイアウトに自由度を持たせ, 実行デバイス毎に最適化するというのが基本的な方針である.

適切なデータレイアウトを選択するための指示文として, acc transform を提案する. acc transform に与えるべき情報は以下である.

- 指示文の対象領域
- レイアウト変更をするべき配列名
- 配列のデータレイアウト
- レイアウトの変換ルール(optional)

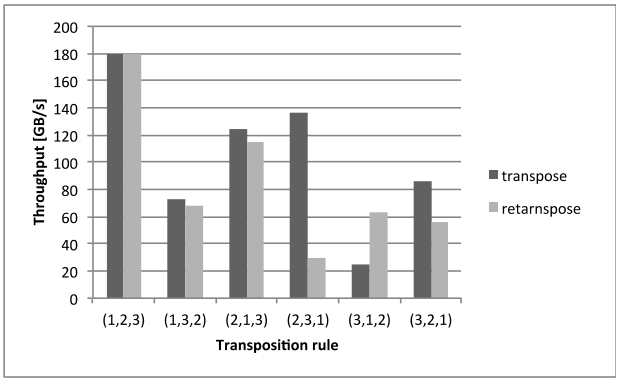

図 2

結果および考察

実装したトランスレータの評価を行うために, 以下の実験を行った.

- ランタイムライブラリ中で用いている転置カーネルを評価するためのマイクロベンチマーク.
- present_transform 指示文, transpose_create 指示子を適用した, 流体アプリケーションである UPACS による評価.
- collapse 指示子の拡張によるループ順変更とランタイムライブラリによる転置の適用を行った, CCS-QCD による評価.

また評価環境としては TSUBAME の Thin ノードを用いており, コンパイラには pgfortran version 16.4 を用いている.

ランタイムライブラリ中で用いる, 多次元配列の転置カーネルの実行時間の評価を行う. 転置を行うカーネ

ルは，ホストからデバイスへの変換を行うカーネル(transpose)と，デバイスからホストへの変換を行うカーネル(retranspose)の 2 種類がある．図 2 は，3 次元配列 array3D(1:10,1:1000,1:2000)の転置を行った際の性能である． 図 2 の左端，規則(1,2,3)は転置を行わない，単なる配列のコピーである．転置カーネルは，多次元・任意規則に対応するためナイーブな実装であるが，この中で実際に使うことの多い転置パターンは規則(2,1,3)または(2,3,1)であると考えられるため，この 2 つのパターンは個別に最適化を施している．n 次元(n > 3) の配列であっても，(2,1,3,4,5,6,...,n) や (3,4,5,...,n,1,2)といった規則であれば，3,4,5,...,n のような連続部分では次元の入れ替えは起こらないため，連続部をまとめて扱うことで，それぞれ規則(2,1,3)，(2,3,1)と同一視することができる．

array3D(1:10,1:1000,1:2000) に規則 (2,1,3) ，(2,3,1)を適用すると，それぞれ(1:1000,1:10,1:2000)，(1:1000,1:2000,1:10)のサイズを持つ配列に転置される．

規則(2,3,1)は単なるコピーである規則(1,2,3)と比較して，transpose の性能は 75.6%であったものの，retranspose は 16.7%の性能であった．さらなる最適化は今後の課題である．ただし，実際にアプリケーションで用いる際に，この変換を行うのは，多くはホスト-デバイス間の通信の発生時であると想定される．そのため，ホスト-デバイス間の通信と比較してこのカーネルが十分に速ければ，オーバーヘッドはほとんど無視できると考えて良い．ホスト-デバイス間のデータ転送を含めた実行時間が transposition_time である．ホスト-デバイス間のデータ転送自体は必ず必要なものであるため，transpose+retranspose カーネルの実行時間が純粋なオーバーヘッドとなる．規則(2,3,1)の場合，2つのカーネルの実行時間がホスト-デバイス間の転送時間の 26.5¥%である．また本実験環境は PCI-e Gen2 であるが，最新の規格では転送速度は 2 倍程度になっているため，オーバーヘッドは相対的に大きくなり，改善が必要であると言える．

しかし，転置の発生するタイミングはホスト-デバイス間の通信時であるため，この転送時間が相対的に小さいアプリケーションにおいては，ライブラリによるオーバーヘッドも小さくなる．さらに，カーネルの実行時間をホスト-デバイス間の通信時間に隠蔽することも原理的には可能である．しかし現在の acc transform 指示文は，既にデバイス上に確保済みの配列を主な対象としているため，ホスト-デバイス間の通信がどのタイミングで行われるか関知しておらず，従って変換時間の隠蔽もできない．この問題は，acc data 指示文を拡張し，確保と変換を同時に行うよう拡張することで達成できると考えられるが，今後の課題である．

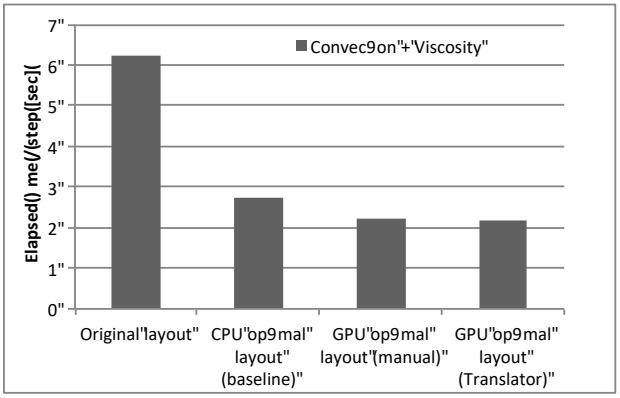

図 3

UPACS は独立行政法人宇宙航空研究開発機構 JAXA により研究開発されている，航空宇宙分野において要求される様々な流体現象の解析に用いることを目的とした，汎用的な流体アプリケーションである．ただし，今回対象としている UPACS は，JAXA の UPACS をベースとして，株式会社 IHI が開発のために独自の拡張を施したものである．

本評価には株式会社 IHI より提供された実際のデータを用いる．実験対象データは単翼周りの流れを解析するための 3 次元データであり，格子点数は約 400 万点である．この 400 万点の格子を 7 つのブロックに分割し，翼列まわりへの適合を行っている．7 つのブロックのうち最も大きいものが 83 × 96 × 120 であり，最も小さいものが 110 × 26 × 120 である．なお UPACS ではこれらのブロックは最大 7MPI 並列で計算することができるが，今回は OpenACC の拡張についての評価であるため，MPI による実行は行わず，1 プロセスが全てのブロックの計算を行う．

これまでの UPACS の OpenACC 化では，手動で書き換えることでデータレイアウト変更の効果について検証している．しかしデータレイアウトの手動による書き

換えはコストが高く，またあるデバイス向けに最適化してしまうと他のデバイスでの性能を損ねる原因となるため，自動で最適化できることが好ましい．今回は，データレイアウトの自動最適化に向けての初期評価として，UPACS の実行時間のうちの 6 割程度を占める，対流項計算部と粘性項計算部に拡張指示文を適用し，評価を行った．

ただし，今回対象とする配列は，ユーザー定義型の構造体の配列である．ユーザー定義型の構造体については expand 指示子によりサポートを予定しているが実装できておらず，直接のレイアウト変更はできなかった．そのため，オリジナルのユーザー定義型から CPU 向けに手動で書き換えたものをベースラインとする．また，今回対象とする配列は，対流項・粘性項の計算時に一時的に用いられる作業用の配列である．従って，全て transpose_create 指示子の適用で済み，転置カーネルを使う必要はない．レイアウト変換対象とした配列を引数とした関数呼び出しについては present_transform 指示文により対応した．プログラム全体に波及するデータレイアウトについては今後適用を進めていく．

図 3 に指示文の適用結果を示す．左端がユーザー定義型を用いたオリジナルのデータレイアウトであるが，拡張指示文適用のベースラインとしたのは CPU optimal layout である．指示文適用により，CPU optimal layout と比較して 23.5%の性能向上を得た．また，指示文を適用しない手動による書き換えと比較した際のオーバーヘッドも 1%未満であった．

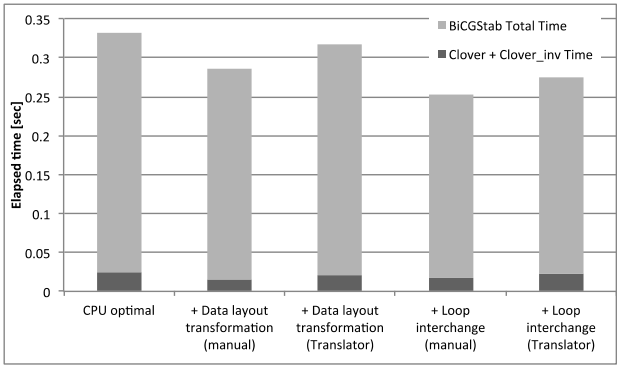

図 4

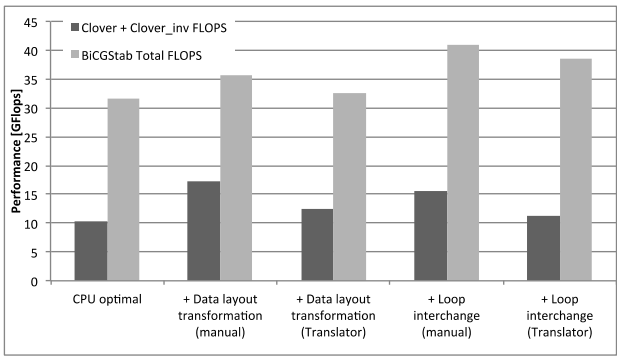

図 5

理研 AICS により整備されている，Fiber Miniapp Suite の中から，実アプリケーションに近いアプリケーションとして，CCS-QCD を用いる．

CCS-QCD は，格子 QCD 計算を行う際に現れる，粗行列向けの線形ソルバのベンチマークプログラムであり，Clover 部及び BiCGStab 部の主に 2 つのカーネルから構成される．

Fiber Miniapp Suite より提供されている CCS-QCD は既に OpenACC 化が施されており，またデータレイアウトの変更とそれに伴うループ順の入れ替えの最適化が行われている．本実験ではこの最適化を外したものをベースラインとしている．

トランスレータによるデータレイアウトの変更では，もともと施されていたレイアウト変更と同様のレイアウトになるように決定している．多くの場合，最内の 2 次元を最外もしくはその一つ内側に移動している．この場合，前述の通り、ライブラリ内部では 3 次元配列の転置とみなして転置が行われる．

図 4 と図 5 が，データレイアウトの変換，ループ順変更の適用を行った際の実行時間[sec]と性能[GFlops]である．なお，問題サイズには CLASS1(8x8x8x32)のものを使用しており，MPI による並列化は行っていない．実行時間は Clover 部と BiCGStab 部の合計であり，性能はそれぞれのものを示している．左端のグラフがベースラインである，データレイアウト変更とループ順変更の最適化を外したものである．左から 2・3 番目がそれぞれ，手動とトランスレータによるデータレイアウトの変更結果である．左から 4・5 番目が，手動とトランスレータそれぞれにより，データレイアウト変更に加えループ順の変更を行った結果である．トランスレータによりレイアウト変換とループ順変更を行った場合，手動で

行った場合と比較すると 92.3%程度の性能であったが，最適化を行わないものと比較すると，120.7%の性能を達成した．

まとめ、今後の課題

本稿では，アーキテクチャにより得意とするデータレイアウトが異なること等が，指示文ベースのプログラミングモデルである OpenACC の，異なるデバイス間における性能可搬性を損ねる原因となることに着目し，データレイアウトの抽象化を行うためのディレクティブを提案し，トランスレータを実装，実アプリケーションを用いて評価を行った．この結果，データのコピーを行う必要のない UPACS においては，手動による最適化と比較して 1%以下のオーバーヘッドであった．また，CCS-QCD による評価においては，データレイアウトの変更に加え，ループ順の変更を適切に行うことで，手動による最適化の 92.3%，最適化を行っていないものの 120.7%の性能を達成した．

しかし，現状のトランスレータには幾つかの課題も見られた．UPACS では，ユーザー定義型の配列に指示文が対応していないために，本来のオリジナルの配列に対しての適用ができなかった．また，現状のトランスレータには自動最適化機構が実装されていない．今回対象とした UPACS，CCS-QCD は，手動による最適化プログラムが既にあったため，同様の形に変更しただけであった．しかし本来，対象デバイスの最適なデータレイアウトを決定するのは難しく，しかも実行するデバイスごとに変化し得る．このような負担を軽減するために，今後は自動最適化を目指す．